18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS 2018, August 23–24, 2018, Helsinki, Finland

Edited by

Ralf Borndörfer
Sabine Storandt

Helsinki · Finland
23-24 August 2018
ATMOS2018

OASICS

Editors

Ralf Borndörfer
Zuse Institute Berlin/Freie Universität Berlin
Berlin
Germany
`borndoerfer@zib.de`

Sabine Storandt
Universität Würzburg
Würzburg
Germany
`storandt@informatik.uni-wuerzburg.de`

ACM Classification 2012
Theory of computation → Design and analysis of algorithms, Mathematics of computing → Discrete mathematics, Mathematics of computing → Combinatorics, Mathematics of computing → Graph Theory, Mathematics of computing → Mathematical optimization, Applied computing → Transportation

ISBN 978-3-95977-096-5

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-096-5.

Publication date
August, 2018

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/OASIcs.ATMOS.2018.0

ISSN 2190-6807

http://www.dagstuhl.de/oasics

Contents

Preface

Running and optimizing transportation systems give rise to very complex and large-scale optimization problems requiring innovative solution techniques and ideas from mathematical optimization, theoretical computer science, and operations research. Since 2000, the series of Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS) workshops brings together researchers and practitioners who are interested in all aspects of algorithmic methods and models for transportation optimization and provides a forum for the exchange and dissemination of new ideas and techniques. The scope of ATMOS comprises all modes of transportation.

The 18th ATMOS workshop (ATMOS'18) was held in connection with ALGO'18 and hosted by Aalto University in Helsinki, Finland, on August 23–24, 2018. Topics of interest were all optimization problems for passenger and freight transport, including, but not limited to, demand forecasting, models for user behavior, design of pricing systems, infrastructure planning, multi-modal transport optimization, mobile applications for transport, congestion modelling and reduction, line planning, timetable generation, routing and platform assignment, vehicle scheduling, route planning, crew and duty scheduling, rostering, delay management, routing in road networks, traffic guidance, and electro mobility. Of particular interest were papers applying and advancing techniques like graph and network algorithms, combinatorial optimization, mathematical programming, approximation algorithms, methods for the integration of planning stages, stochastic and robust optimization, online and real-time algorithms, algorithmic game theory, heuristics for real-world instances, and simulation tools.

There were twenty-nine submissions from eighteen countries. All of them were reviewed by at least three referees in ninety-one reviews, among them five external ones, and judged on their originality, technical quality, and relevance to the topics of the workshop. Based on the reviews, the program committee selected sixteen submissions to be presented at the workshop (acceptance rate: 55%), which are collected in this volume in the order in which they were presented. Together, they quite impressively demonstrate the range of applicability of algorithmic optimization to transportation problems in a wide sense. In addition, Dennis Huisman kindly agreed to complement the program with an invited talk on *Railway Disruption Management: State-of-the-Art in Practice and New Research Directions*.

Based on the reviews, Ralf Borndörfer, Marika Karbstein, Christian Liebchen, and Niels Lindner won the Best Paper Award of ATMOS'18 with their paper *A Simple Way to Compute the Number of Vehicles That Are Required to Operate a Periodic Timetable*. In addition, we awarded Tomas Lidén the Best VGI Paper Award of ATMOS'18 for his paper *Reformulations for Integrated Planning of Railway Traffic and Network Maintenance*.

We would like to thank the members of the Steering Committee of ATMOS for giving us the opportunity to serve as Program Chairs of ATMOS'18, all the authors who submitted papers, Dennis Huisman for accepting our invitation to present an invited talk, the members of the Program Committee and the additional reviewers for their valuable work in selecting the papers appearing in this volume, our sponsors MODAL, TomTom, and VGIscience for their support of the prizes, and the local organizers for hosting the workshop as part of ALGO'18. We acknowledge the use of the EasyChair system for the great help in managing the submission and review processes, and Schloss Dagstuhl for publishing the proceedings of ATMOS'18 in its OASIcs series.

August 2018 Ralf Borndörfer
Sabine Storandt

18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018).
Editors: Ralf Borndörfer and Sabine Storandt
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Program Committee

Markus Bohlin	RISE SICS, Sweden
Ralf Borndörfer (co-chair)	Zuse Institute Berlin/Freie Universität Berlin, Germany
Stefan Funke	Universität Stuttgart, Germany
Natalia Kliewer	Freie Universität Berlin, Germany
Spyros Kontogiannis	University of Ioannina, Greece
Alexander Kröller	TomTom, Germany
Jesper Larsen	Technical Universtity of Denmark, Denmark
Carlo Mannino	SINTEF, Norway
Matthias Müller-Hannemann	Martin-Luther-Universität Halle-Wittenberg, Germany
Güvenç Şahin	Sabancı Üniversitesi, Turkey
Peter Sanders	Karlsruhe Institute of Technology, Germany
Marie Schmidt	Erasmus University Rotterdam, Netherlands
Sabine Storandt (co-chair)	University of Würzburg, Germany
Pieter Vansteenwegen	Katholieke Universiteit Leuven, Belgium
Renato Werneck	Amazon, USA

Steering Committee

Alberto Marchetti-Spaccamela	Universita di Roma “La Sapienza”, Italy
Anita Schöbel	Georg-August-Universität Göttingen, Germany
Dorothea Wagner	Karlsruhe Institute of Technology (KIT), Germany
Christos Zaroliagis (chair)	University of Patras, Greece

List of Subreviewers

Lukas Bach	SINTEF, Norway
Valentin Buchhold	Karlsruhe Institute of Technology, Germany
Stefan Funke	Universität Stuttgart, Germany
Sascha Witt	Karlsruhe Institute of Technology, Germany
Andreas Linscheid	TomTom, Germany
Dario Pacciarelli	Universita di Roma “La Sapienza”, Italy

Local Organizing Committee

Parinya Chalermsook	Aalto University, Finland
Petteri Kaski	Aalto University, Finland
Jukka Suomela	Aalto University, Finland

Reformulations for Integrated Planning of Railway Traffic and Network Maintenance

Tomas Lidén
Linköping University, Department of Science and Technology
Norrköping SE-601 74, Sweden
tomas.liden@liu.se
https://orcid.org/0000-0002-1643-6365

Abstract

This paper addresses the capacity planning problem of coordinating train services and network maintenance windows for a railway system. We present model reformulations, for a mixed integer linear optimization model, which give a mathematically stronger model and substantial improvements in solving performance – as demonstrated with computational experiments on a set of synthetic test instances. As a consequence, more instances can be solved to optimality within a given time limit and the optimality gap can be reduced quicker.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases Railway scheduling, Maintenance planning, Optimization

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.1

Funding This research is funded by the Swedish Transport Administration with the grant TRV 2013/55886 and conducted within the national research program "Capacity in the Railway Traffic System".

Acknowledgements The work was performed during a research visit to University of Newcastle, Australia, where Thomas Kalinowski and Hamish Waterer gave invaluable help, ideas and inspiration.

1 Introduction

Scheduling access to a railway infrastructure, commonly termed *capacity planning*, is the core tactical planning problem for all railway systems and can be seen as a resource planning for the infrastructure components (stations, lines, yards, tracks, switches, signalling blocks etc). Capacity planning includes producing a timetable for the train traffic and access (or possession) plans for maintenance and work tasks. Timetables and possession plans will in turn form the basis for other resource plans, such as rolling stock plans and crew schedules for the train operators as well as equipment and work force plans for maintenance and renewal contractors.

Train services and maintenance tasks should ideally be planned together, but have mostly been treated as separate planning problems. While planning of train operations has been extensively studied in the research literature [2, 3, 4, 5, 6], there has been much less focus on maintenance planning [8]. As for the joint planning of train services and network maintenance there are a few examples, which consider the introduction of a small number of work possessions into an existing train timetable [7, 11] or operative plan [1], by allowing different types of adjustments to the trains.

This research focuses on the long term tactical coordination of a large volume of maintenance windows and train services on a railway network. An example of how such plans

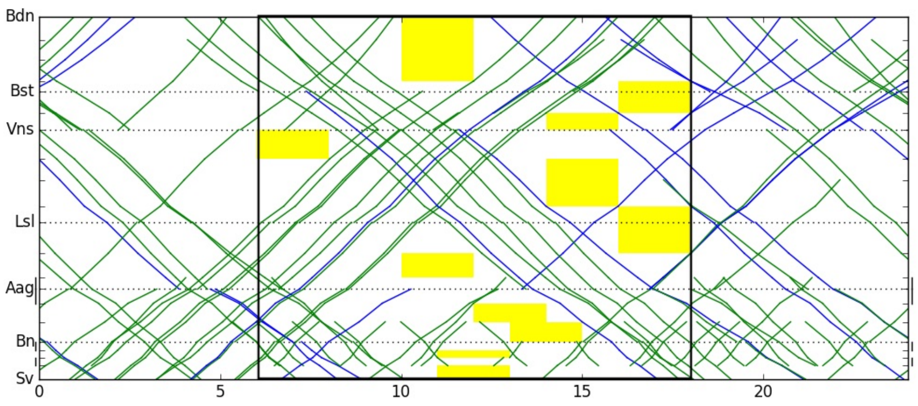


Figure 1 Train and work graph example.

might look is given in Figure 1, as a train and work graph. The geographic distance and sectioning into links is shown on the vertical axis while time is on the horizontal axis. Train services are shown as tilted lines while maintenance windows are shown as yellow boxes. In this case no trains are allowed to run "through" the maintenance windows.

An initial MILP model that solves this coordination problem to optimality has been presented in [9]. Model extensions for assigning maintenance crew resources and considering their costs and limitations regarding spatial availability as well as work and rest time regulations are treated in [10]. The latter model also has a stronger formulation for the train and maintenance window scheduling, but that paper only briefly summarizes these model improvements. In this paper we describe and compare the two formulations more closely.

The original model, which we here denote with ORG, uses cumulative train entry/exit variables (for each link and time period) and implicit link usage variables for the train scheduling. The improved model, which we denote with IMP, instead uses binary train entry/exit detection variables and explicit link usage variables. These changes increase the number of variables and decrease the number of constraints. The linear relaxation does not become tighter, but the MILP solver benefits from having more binary variables to branch on, a better linking of constraints and some possibilities for pruning due to the binary restrictions.

The main improvement in IMP concerns the maintenance scheduling part. First of all, some coupling constraints have been aggregated, but more importantly a tighter formulation has been used for the maintenance work and window start variables – according to the modelling for bounded up/down sequences as presented in [12, section 11.4, pp 341–343] and mathematically analysed in [13]. These improvements do make the linear relaxation tighter and in addition the MILP solver presolve method is able to reduce the size more effectively.

The net effect of these model improvements is that the solution performance gets better, more instances can be solved to optimality within a given time limit and the optimality gap can be reduced quicker.

The remainder of the paper is organised as follows: Section 2 gives the mathematical formulation by first introducing the necessary notation and giving an overview of the model structure. Then the reformulation for the train scheduling part is described, followed by the changes for the maintenance window scheduling part. The computational experiments are presented in Section 3 after which some concluding remarks are made.

2 Mathematical formulation

2.1 Notation and model structure

The railway network is modelled by a link set L, where a subset of links $L^M \subseteq L$ shall have maintenance windows. The scheduling problem has a planning horizon of length H, divided into a sequence $T = \{0, \dots, H-1\}$ of unit size time periods $t \in T$, each covering real-valued event times between t and $t+1$.

For each link $l \in L^M$, a required number of time periods shall have maintenance windows. The scheduling shall be done according to a set W_l of window options where each option $o \in W_l$ is defined by a tuple $o = (\eta_o, \theta_o)$ that gives the required number η_o of maintenance windows to schedule, and the window length θ_o expressed as an integer number of time periods. As an example, $W_l = \{(1,3), (2,2)\}$ means that either one window of length three or two windows of length two shall be scheduled on link l.

For the train traffic we have a set S of train services. Each train service $s \in S$ has a set R_s of possible routes. Each route $r \in R_s$ implies a sequence L_r of links, and the set L_s of all possible links that train service s can traverse is given by the union of the sets L_r for all $r \in R_s$. The scheduling of trains shall be done by selecting one route $r \in R_s$ and deciding entry and exit times for each link in that route, such that all event times are within the scheduling window defined by $T_s \subseteq T$.

The model has two groups of variables. The main variables for scheduling train services are:

z_{sr}	route choice: whether train service s uses route r or not
e^+_{sl}, e^-_{sl}	event time: entry(+)/exit(−) time for service s on link l
x^+_{slt}, x^-_{slt}	link entry/exit: whether train service s enters/exits link l in time period t or not
u_{slt}	link usage: whether train service s uses link l in time period t or not
n^h_{lt}	number of train services traversing link l in direction h during time period t

The ORG formulation uses cumulative x variables, which we denote by $\bar{x}^+_{slt}, \bar{x}^-_{slt}$, and implicit u variables, while the IMP formulation uses binary x variables and explicit u. The variables for scheduling maintenance windows are:

w_{lo}	maintenance window option choice: whether link l is maintained with window option o or not
y_{lt}	maintenance work: whether link l is maintained in time period t or not
v_{lot}	work start: whether maintenance on link l according to window option o is started in time period t or not

The model can be summarized as follows:

$$\text{minimize } c\left(\mathbf{z}, \mathbf{e}, \mathbf{y}, \mathbf{v}\right) \tag{1}$$

$$\text{subject to } \mathbf{A}(\mathbf{z}, \mathbf{e}, \mathbf{x}, \mathbf{u})^{\,route} \tag{2}$$

$$\mathbf{A}(\mathbf{z}, \mathbf{e})^{\,trains} \tag{3}$$

$$\mathbf{A}(\mathbf{w}, \mathbf{y}, \mathbf{v})^{\,maintenance} \tag{4}$$

$$\mathbf{A}(\mathbf{u}, \mathbf{n}, \mathbf{y})^{\,capacity} \tag{5}$$

$$\text{variable types and bounds}$$

where $c(..)$ is the objective function and $\mathbf{A}(..)$ are linear constraint functions over one or more of the indicated variables. The objective (1) is a linear combination of the train and maintenance scheduling variables, while the constraints enforce: (2) correct (feasible) bounds

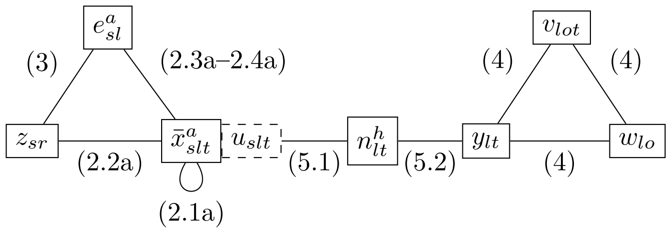


Figure 2 Variable and constraint graph – ORG formulation.

on the train events and linking of entry / exit and usage variables according to the selected route, (3) sufficient travel durations and dwell times along the chosen route, (4) sufficient maintenance windows scheduled according to the chosen option, and (5) that the available network capacity is respected.

The ORG and IMP formulations differ regarding constraints (2) and (4), which will be described in the following sections. For the details regarding the objective function and other constraints, we refer to [10].

2.2 Train scheduling

The ORG formulation uses cumulative variables $\bar{x}^+_{slt}, \bar{x}^-_{slt}$, which takes value 1 if train service s has entered/exited link l in time period t or earlier. The link usage is given by the implicit variables $u_{slt} := \bar{x}^+_{slt} - \bar{x}^-_{sl,t-1}$, with the convention that $\bar{x}^a_{sl,t-1} = 0$ for $t = 0$.

The constraint set (2) for ORG is:

$$\bar{x}^a_{slt} \geq \bar{x}^a_{sl,t-1} \qquad \forall s \in S, l \in L_s, t \in T_s, a \in \{+,-\} \tag{2.1a}$$

$$\bar{x}^a_{sl,\text{last}(T_s)} = \sum_{r \in R_s : l \in L_r} z_{sr} \qquad \forall s \in S, l \in L_s, a \in \{+,-\} \tag{2.2a}$$

$$e^a_{sl} \geq \sum_{t \in T_s} LB^a_t (\bar{x}^a_{slt} - \bar{x}^a_{sl,t-1}) \qquad \forall s \in S, l \in L_s, a \in \{+,-\} \tag{2.3a}$$

$$e^a_{sl} \leq \sum_{t \in T_s} UB^a_t (\bar{x}^a_{slt} - \bar{x}^a_{sl,t-1}) \qquad \forall s \in S, l \in L_s, a \in \{+,-\} \tag{2.4a}$$

where LB^a_t and UB^a_t are the lower and upper bound time values for entry and exit in time period t. The constraints enforce: (2.1a) the cumulative property, (2.2a) that all links in the selected route will be visited, and (2.3a–2.4a) correct lower and upper bounds for the event variables.

The structure of this model is illustrated in Figure 2 as a constraint (or co-occurrence) graph with vertices for the variables and edges connecting variables that occur in the same constraint. The constraints correspond to cliques in the graph as indicated in the figure.

The IMP formulation uses binary detection variables x^+_{slt}, x^-_{slt} to track whether service s enters or exits link l in time period t or not. These variables correspond to the cumulative $\bar{x}$ variables in the ORG formulation as follows

$$x^a_{slt} = \bar{x}^a_{slt} - \bar{x}^a_{sl,t-1}$$

This relation is illustrated in Figure 3, both for the binary case and for a linear relaxation. Using the expression

$$\bar{x}^a_{slt} = \sum_{t' \in T_s : t' \leq t} x^a_{slt'}$$

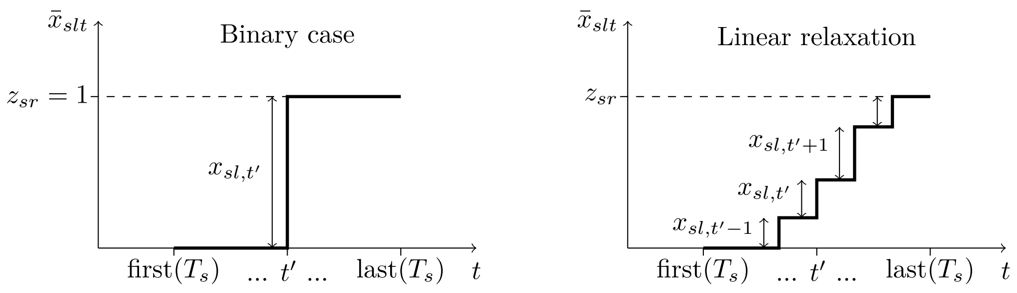


Figure 3 Relation between cumulative variables $\bar{x}$ and detection variables x.

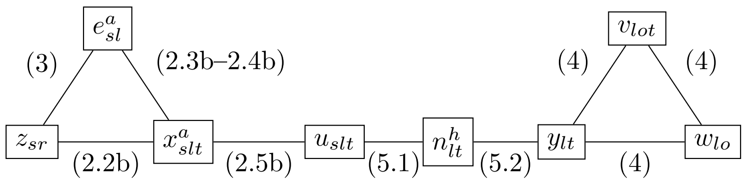


Figure 4 Variable and constraint graph – IMP formulation.

we transform constraints (2.2a–2.4a) and introduce explicit u variables to get the IMP formulation of (2):

$$\sum_{t \in T_s} x^a_{slt} = \sum_{r \in R_s : l \in L_r} z_{sr} \qquad \forall s \in S, l \in L_s, a \in \{+,-\} \tag{2.2b}$$

$$e^a_{sl} \geq \sum_{t \in T_s} LB^a_t x^a_{slt} \qquad \forall s \in S, l \in L_s, a \in \{+,-\} \tag{2.3b}$$

$$e^a_{sl} \leq \sum_{t \in T_s} UB^a_t x^a_{slt} \qquad \forall s \in S, l \in L_s, a \in \{+,-\} \tag{2.4b}$$

$$u_{slt} = \sum_{t' \in T_s : t' \leq t} x^+_{slt'} - \sum_{t' \in T_s : t' \leq t-1} x^-_{slt'} \qquad \forall s \in S, l \in L_s, t \in T_s \tag{2.5b}$$

Note that the cumulative constraints disappear, but that we have new constraints (2.5b) for the usage variables. The IMP formulation will have $|S|\,|L_s|\,|T_s|$ more variables but $|S|\,|L_s|\,|T_s|$ less constraints as compared to ORG. Also the train counting constraints ($n^h_{lt} = \sum u_{slt}$) will operate on the explicit u variables and hence contain fewer elements in IMP as compared to ORG.

The increase in variables might be a drawback, but also gives the MILP solver an opportunity for more pruning (due to the binary restriction of u) and another set of variables to branch on during the branch and bound procedure.

The constraint graph for the IMP formulation is shown in Figure 4.

2.3 Maintenance scheduling

In the following we study the formulation differences between ORG and IMP for the maintenance scheduling constraints (4). In the ORG formulation we have

$$\sum_{o\in W_l} w_{lo} = 1 \qquad \forall l \in L^M \qquad (4.1)$$

$$\sum_{t\in T} v_{lot} \geq \eta_o w_{lo} \qquad \forall l \in L^M, o \in W_l \qquad (4.2)$$

$$v_{lot} + 1 \geq y_{lt} - y_{l,t-1} + w_{lo} \qquad \forall l \in L^M, o \in W_l, t \in T \qquad (4.3a)$$

$$v_{lot} \leq w_{lo} \qquad \forall l \in L^M, o \in W_l, t \in T \qquad (4.4)$$

$$v_{lot} \leq y_{lt} \qquad \forall l \in L^M, o \in W_l, t \in T \qquad (4.5a)$$

$$v_{lot} \leq 1 - y_{l,t-1} \qquad \forall l \in L^M, o \in W_l, t \in T \qquad (4.6a)$$

$$\sum_{t'=t}^{t+\theta_o} y_{lt'} \geq \theta_o v_{lot} \qquad \forall l \in L^M, o \in W_l, t \in T \qquad (4.7a)$$

Constraint (4.1) ensures that exactly on window option is used, while (4.2) ascertain a sufficient number of maintenance windows. Constraints (4.3a–4.6a) ensure the correct coupling of work start variables (v_{lot}), window choice (w_{lo}) and work variables (y_{lt}), while constraint (4.7a) imposes the required maintenance window lengths.

The coupling constraints (4.3a,4.5a,4.6a) can be aggregated, by utilising the fact that exactly one window option must be selected. Hence, since only one v_{lot} variable for each l, t combination can be non-zero, IMP uses summations over the window options, as follows:

$$\sum_{o\in W_l} v_{lot} \geq y_{lt} - y_{l,t-1} \qquad \forall l \in L^M, t \in T \qquad (4.3b)$$

$$\sum_{o\in W_l} v_{lot} \leq y_{lt} \qquad \forall l \in L^M, t \in T \qquad (4.5b)$$

$$\sum_{o\in W_l} v_{lot} \leq 1 - y_{l,t-1} \qquad \forall l \in L^M, t \in T \qquad (4.6b)$$

Next, we make use of a model for bounded on/off sequences, presented in [13], where the formulation describes the convex hull. Thus there is no tighter formulation for that set of variables. We extend this model with the window option choice w_{lo} and can then replace (4.7a) with the following constraints:

$$\sum_{o\in W_l} \left[\sum_{t'=t+1-\theta_o}^{t} v_{lot'} \right] \leq y_{lt} \qquad \forall l \in L^M, t \in T \qquad (4.7b)$$

$$\sum_{t'=t+1-\theta_o}^{t} v_{lot'} + 1 \geq y_{lt} + w_{lo} \qquad \forall l \in L^M, o \in W_l, t \in T \qquad (4.8b)$$

$$\sum_{o\in W_l} v_{lot} \leq 1 - y_{l,t-1} \qquad \forall l \in L^M, t \in T \qquad (4.9b)$$

$$\sum_{t'=t+1}^{t+MS_o} v_{lot'} \geq w_{lo} - y_{lt} \qquad \forall l \in L^M, o \in W_l, t = 1, \ldots, H - MS_o \qquad (4.10b)$$

The constraints (4.7b)-(4.8b) enforce that each window should span exactly θ_o time periods. Constraint (4.9b) enforces at least one time period between two maintenance windows, but since it is precisely the same as (4.6b) and hence redundant it is not needed. If there would be requirements for larger separation between windows on each link, the LHS should be a suitable forward going sum over v_{lot}. Constraint (4.10b) will make sure that the maximum separation (MS_o) between windows is respected. This constraint and (4.8b) will only be active for the chosen window option. Here we can only aggregate over the window options for constraints (4.7b) (and (4.9b)).

3 Computational results

The same set of synthetic test instances as in [9] has been used for evaluating the efficiency of the different formulations. The data instances are available as JSON files together with a set of Python parsers at `https://github.com/TomasLiden/mwo-data.git`. The test set consists of nine line instances (L1–L9) and five network instances (N1–N5), having a planning horizon of five hours to one week divided into 1 h periods and with 20 to 350 train services. All line instances except one (L4) are single track, while the network instances have a mixture of single and double track links. The trains are uniform with no runtime or cost differences – only the preferred departure times differ. These simplifications, which make the trains almost indistinguishable for the solver, are used in order to test the scalability and solvability of the models. Real-life instances will of course use more realistic settings for costs and runtimes.

The evaluations have been done in two steps. First, various alternatives for the train scheduling formulation have been tested. These tests were made with Gurobi 6.0.5 as MIP solver on a MacBook Pro with a 2,6 GHz Intel Core i5 processor, 8 GB 1600 MHz DDR3 memory and OSX 10.10.5. The formulation with the best performance, as presented in Section 2.2, was then used when evaluating various alternatives for the maintenance scheduling part. The latter tests were made with Gurobi 6.5 as MIP solver on a Dell PowerEdge R710 rack server with dual hex core 3.06GHz Intel Xeon X5675 processors and 96GB RAM running Red Hat Enterprise Linux 6. For all tests, a maximum computation time of 3600 seconds have been used. Most tests have a relative MIP gap tolerance of 0.001 (0.1%) while some of the smaller instances (L1–L5 and N1–N2) have 0.01%. All other options have been left at their default values.

The computational results are presented in Table 1, where the left part gives instance properties, the middle part lists solution statistics for the four alternatives

(a) ORG - the original formulation, run with Gurobi 6.0.5
(b) T60 - the improved train scheduling formulation with Gurobi 6.0.5
(c) T65 - the improved train scheduling formulation with Gurobi 6.5
(d) IMP - the complete improved formulation with Gurobi 6.5

and the right part lists the absolute improvement in initial LP value as compared to the ORG formulation.

There is a clear improvement in solving performance – both solution times and remaining MIP gaps are reduced. Also, we see that IMP is tighter since the initial LP value is increasing (the small increases for some T65 instances are caused by a more efficient pre-solve).

To further illustrate the improvements, performance profile plots are used which show the accumulated number of instances (on the vertical axis) reaching a certain level of quality measure (on the logarithmic horizontal axis) – normalised as a factor of the best outcome for all alternatives. Thus an alternative have better performance when being above (= more instances) and to the left (= better quality) of another curve. In Figure 5 the time for

Table 1 Instances, properties, performance (time (seconds) to reach optimality or else remaining gap after 3600 seconds) and improvement in initial LP value vs ORG.

	Properties			Performance (sol. time / rem. gap)				LP improvement		
Case	$\|L\|$	$\|T\|$	$\|S\|$	ORG	T60	T65	IMP	T60	T65	IMP
L1	4	5	20	66	15	3	4	0	0	0
L2	4	5	20	181	18	4	4	0	0	0
L3	4	12	40	0.15%	255	102	46	0	0	0.6
L4	4	12	40	2	4	2	1	0	0	0.4
L5	9	24	40	0.16%	2878	3415	1796	0	0	0.75
L6	9	48	80	0.77%	0.50%	0.21%	0.13%	0	0	0.95
L7	18	24	80	31.0%	1.24%	0.90%	0.53%	0	0	1.35
L8	18	96	160	32.0%	17.6%	1.62%	1.14%	0	0	0.55
L9	25	168	350	265%	201%	170%	64.4%	0	0	2.76
N1	9	5	20	7	9	5	3	0	0.02	0.02
N2	9	24	50	42	25	6	5	0	0.12	0.27
N3	9	48	100	97	179	193	23	0	0	0.98
N4	9	96	200	1209	535	435	264	0	0.01	0.28
N5	9	168	350	0.14%	0.13%	3600	907	0	0	0.92

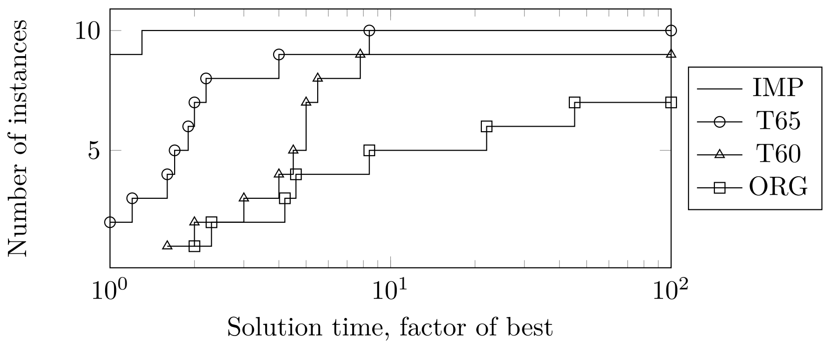


Figure 5 Performance profile – time to reach optimality.

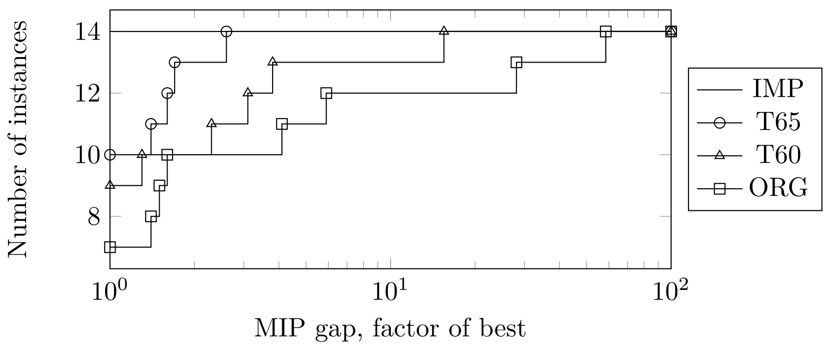


Figure 6 Performance profile – final MIP gap.

reaching optimality is the quality measure, while Figure 6 shows the remaining MIP gap for those instances that have not been solved to optimality. The improvement obtained for each step is clear – including the performance gain when changing solver version and computer platform.

The net result is that three more instances (L3, L5, N5) are solved to optimality, the optimal solutions are reached quicker (with a speed up between 2 and 10 times) and two more instances (L7, L8) are solved to a MIP gap $< 1.5\%$, which can be considered an acceptable solution quality for the cost factors being used.

4 Concluding remarks

We have investigated and found reformulations that substantially improve the solving performance for an optimization model that jointly schedules train services and network maintenance windows. The reformulations include the removal of cumulative variables, making implicit variables explicit, using aggregation where appropriate but most importantly to use a tighter model for bounded up/down sequences (according to [13]).

These improvements have made it possible to extend the model with maintenance resource considerations (see [10]), and in recent work the models have also been applied to real world problems of realistic size – which will be reported in the presentation. In the latter work cyclic scheduling is used, which unfortunately destroys the integral properties of the previously mentioned model for bounded up/down sequences. Hence, the mathematical properties of cyclic on/off sequences are currently being studied with the aim of finding methods for strengthening such models.

References

1 A. R. Albrecht, D. M. Panton, and D. H. Lee. Rescheduling rail networks with maintenance disruptions using problem space search. *Computers & Operations Research*, 40(3):703–712, 2013. doi:10.1016/j.cor.2010.09.001.

2 V. Cacchiani, D. Huisman, M. Kidd, L. Kroon, P. Toth, L. Veelenturf, and J. Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, 2014. doi:10.1016/j.trb.2014.01.009.

3 V. Cacchiani and P. Toth. Nominal and robust train timetabling problems. *European Journal of Operational Research*, 219(3):727–737, 2012. doi:10.1016/j.ejor.2011.11.003.

4 A. Caprara, L. Kroon, M. Monaci, M. Peeters, and P. Toth. Passenger railway optimization (ch 3). In *Handbooks in Operations Research and Management Science*, volume 14, pages 129–187. Elsevier, 2007. doi:10.1016/S0927-0507(06)14003-7.

5 A. Caprara, L. Kroon, and P. Toth. Optimization problems in passenger railway systems. *Wiley Encyclopedia of Operations Research and Management Science*, 2011. doi:10.1002/9780470400531.eorms0647.

6 F. Corman and L. Meng. A review of online dynamic models and algorithms for railway traffic management. *IEEE Transactions on Intelligent Transportation Systems*, 16(3):1274–1284, 2014. doi:10.1109/TITS.2014.2358392.

7 M. Forsgren, M. Aronsson, and S. Gestrelius. Maintaining tracks and traffic flow at the same time. *Journal of Rail Transport Planning & Management*, 3(3):111–123, 2013. doi:10.1016/j.jrtpm.2013.11.001.

8 T. Lidén. Railway infrastructure maintenance - a survey of planning problems and conducted research. *Transportation Research Procedia*, 10:574–583, 2015. doi:10.1016/j.trpro.2015.09.011.

9 T. Lidén and M. Joborn. An optimization model for integrated planning of railway traffic and network maintenance. *Transportation Research Part C: Emerging Technologies*, 74:327–347, 2017. `doi:10.1016/j.trc.2016.11.016`.

10 T. Lidén, T. Kalinowski, and H. Waterer. Resource considerations for integrated planning of railway traffic and maintenance windows. *Journal of Rail Transport Planning & Management*, 8.1:1–15, 2018. `doi:10.1016/j.jrtpm.2018.02.001`.

11 X. Luan, J. Miao, L. Meng, F. Corman, and G. Lodewijks. Integrated optimization on train scheduling and preventive maintenance time slots planning. *Transportation Research Part C: Emerging Technologies*, 80:329–359, 2017. `doi:10.1016/j.trc.2017.04.010`.

12 Y. Pochet and L. Wolsey. *Production planning by mixed integer programming.* Springer series in operations research and financial engineering. Springer, 2006. `doi:10.1007/0-387-33477-7`.

13 M. Queyranne and L. Wolsey. Tight MIP formulations for bounded up/down times and interval-dependent start-ups. *Mathematical Programming*, 164(1-2):129–155, 2017. `doi:10.1007/s10107-016-1079-2`.

Large Scale Railway Renewal Planning with a Multiobjective Modeling Approach

Nuno Sousa
INESC-Coimbra, Coimbra, Portugal
Department of Sciences and Technology, Open University, Lisbon, Portugal
R. da Escola Politécnica 141-147, 1269-001 Lisboa, Portugal. Phone +351 213 916 300
nsousa@uab.pt
https://orcid.org/0000-0002-2681-5035

Luís Alçada-Almeida
INESC-Coimbra, Coimbra, Portugal
Faculty of Economics, University of Coimbra, Coimbra, Portugal
Av. Dr. Dias da Silva 165, 3004-512 Coimbra, Portugal. Phone: +351 239 790 500
alcada@fe.uc.pt
https://orcid.org/0000-0003-1272-9053

João Coutinho-Rodrigues
INESC-Coimbra, Coimbra, Portugal
Department of Civil Engineering, Faculty of Sciences and Technology, University of Coimbra, Coimbra, Portugal
Rua Luís Reis Santos – Polo II, 3030-788 Coimbra, Portugal. Phone +351 239 797 100
coutinho@dec.uc.pt
https://orcid.org/0000-0001-9311-5584

Abstract

A multiobjective modeling approach for managing large scale railway infrastructure asset renewal is presented. An optimized intervention project schedule is obtained considering operational constraints in a three objectives model: evenly spreading investment throughout multiple years, minimizing total cost, minimizing work start postponements on higher priority railway sections. The MILP model was based on a real world case study; the objectives and constraints specified by an infrastructure management company. Results show that investment spreading greatly influences the other objectives and that total cost fluctuations depend on the overall condition of the railway infrastructure. The model can produce exact efficient solutions in reasonable time, even for very large-sized instances (a test network of similar size to the USA railway network, the largest in the world). The modeling approach is therefore a very useful, practical methodology, for generating optimized solutions and analyzing trade-offs among objectives, easing the task of ultimately selecting a solution and produce the works schedule for field implementation.

2012 ACM Subject Classification Theory of computation → Integer programming

Keywords and phrases Rail infrastructure, Renewal maintenance, Multiobjective modeling

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.2

Funding Portuguese Foundation for Science and Technology, project grant UID/MULTI/00308/2013.

Acknowledgements The authors would like to thank Vitor Dias da Silva for his help preparing the manuscript.

1 Introduction

The railway has recognized economic, energy and environmental benefits [2], as well as lower operating externalities when compared to road infrastructure [13]. In recent years, the need to provide for a rising demand of rail services has prompted infrastructure managers to intensify maintenance actions, leading to a range of planning problems [1, 4, 6, 9, 11, 12, 14].

The European Commission, in view of these advantages, has been taking measures to increase the use of this mode of transport, by opening up the market to competition, creating new infrastructure and improving the interoperability and safety of existing networks. Ensuring the safety of people and goods, as well as the normal running of rail services, requires maintenance of the existing railway, much of it degraded after decades of disinvestment. In the context of maintenance, it is important to distinguish between current maintenance and renewal interventions. Current maintenance refers to frequent minor works aiming at maintaining an adequate level of service of the infrastructure, whereas renewal actions are typically more extensive and restore (or modernize) the infrastructure [5].

In this article a multiobjective methodology to plan renewal interventions in the railroad is presented, taking into account three objectives: to spread out investment expenses, as evenly as possible, over project years; to minimize the total renewal costs; to minimize work start postponements on the higher priority railway lines. Equitable distribution is required since large-scale renewal actions require a very considerable financial effort from the infrastructure management company, and it is desirable that this effort is diluted as much as possible over multiple years. Achieving a balanced annual investment plan, without compromising the total financial effort or excessively postponing the execution of the priority works, was the motivation for pursuing the research which is now presented. For recent research concerning other aspects (not just financial) of resource levelling in project management see e.g. [3, 8]. It should be noted that the objectives, as well operational constraints to be respected, were defined by an infrastructure management company operating at national scale, which also provided field data for one of the case studies, as well as model parameter calibrations. Indeed, the proposed model stemmed from interaction between a research institution and a railway infrastructure management company, and therefore authors are strongly convinced of its practical usefulness, given it provides a scientific methodology to deal with a real problem in corporate asset management.

2 Multi-objective model

Following the terminology of [7], "renewal" refers to background interventions subsequent to the natural wear and tear of the infrastructure, "line" refers to major railway lines connecting principal stations, and "section" to parts of a line between two geographic landmarks. These marks are usually stations or junctions but may also be mere kilometer points. Sections are often heterogeneous, in which case they are divided into homogeneous subsections. Sections are what undergoes renewal works.

The model is suitable for treating renewal actions which do not involve prolonged track closure or re-routing of the circulation through multiple alternative routes. Typically these are large-scale, extensive interventions on rails, ballasts, sleepers, etc. and may involve upgrading rail assets. Interventions on other asset types (e.g. catenaries, sub-base) may be included provided they do not lead to prolonged blockades. While a section is under intervention, trains must run at reduced speed, causing delays in services. The model cannot, therefore, allow for an accumulation of works on the same line which may cause excessively large delays. Similarly, the lines do not all have the same socio-economic importance or service intensity, making it is necessary to prioritize the sections to be renewed. The model takes these issues into account and considers two periods of accounting as well, monthly and annual, the first to schedule the field works and the second for budgeting. Both can be changed without affecting the structure of the model.

Indices:

$i = 1, \ldots, M$ railway line sections to be renovated.
$j = 1, \ldots, N$ spanning months.
$k = 1, \ldots, P$ spanning years; $N = 12P$.
$l = 1, \ldots, Q$ railway lines. Each section belongs to a railway line.

Parameters: (units)

$\boldsymbol{C_i^R}$ cost of renewing section i (monetary unit MU).
$\boldsymbol{C_{ij}^{EM}}$ extra maintenance cost of section i if it is not renewed as of month j (MU). These costs are active until the repair works end.
$\boldsymbol{P_i}$ priority for renewing section i (adimensional). Active until repair works on that section are completed. This can also be seen as service inconvenience of not renewing the section.
$\boldsymbol{T_i}$ time span needed for renewing section i (months).
$\boldsymbol{D_i}$ delay caused to railway traffic from having section i under renewal(minutes).
$\boldsymbol{B_{il}}$ 1 if section i belongs to line l, 0 otherwise (binary). Note: in the case studies, no section belongs to two lines, but that is not forbidden.
$\boldsymbol{M_l}$ max delay tolerable for line l (minutes).

Decision variables:

$\boldsymbol{x_{ij}}$ 1 if section i starts to be renewed in month j, 0 otherwise (binary).
$\boldsymbol{F}$ maximum yearly investment (real positive variable).

Auxiliary variables:
$\boldsymbol{A_{ij}}$ 1 if section i is being renewed in month j, 0 otherwise (binary).
$\boldsymbol{U_{ij}}$ 1 if the renewal of section i is not yet finished by month j, 0 otherwise (binary).

Model:

$$\min \ O_1 = F \tag{1}$$

$$\min \ O_2 = \sum_i C_i^R + \sum_{ij} C_{ij}^{EM} U_{ij} \tag{2}$$

$$\min \ O_3 = \sum_{ij} P_i U_{ij} \tag{3}$$

Subject to:

$$\sum_j x_{ij} = 1, \quad \forall i \tag{4}$$

$$x_{ij} = 0, \quad \forall i : j > N - T_i \tag{5}$$

$$A_{ij} = \sum_{j'=j-T_i+1, j' \geq 1}^{j} x_{ij'}, \quad \forall ij \tag{6}$$

$$U_{ij} = \sum_{j'=j-T_i+1, j' \geq 1}^{N} x_{ij'}, \quad \forall ij \tag{7}$$

$$\sum_{j=12(k-1)+1}^{12(k-1)+12} \left[\sum_i \left(\frac{C_i^R}{T_i} A_{ij} + C_{ij}^{EM} U_{ij} \right) \right] \leq F, \quad \forall ij \tag{8}$$

$$\sum_i D_i A_{ij} B_{il} \leq M_l, \quad \forall jl \tag{9}$$

Objective O_1 is implemented by equations formulas (1) and (8), where the 1st member of (8) is the annual investment. The extra costs C_{ij}^{EM} are active until the end of the work, but these costs can be considered in other ways, such as e.g. being active up until halfway the work completion. Objective O_2 has a fixed and a variable part and was thus defined to give the decision maker a better notion of the final values. As for O_3, sections accumulate priority values, month after month, until their respective renewal is complete. The more a high-priority work is postponed, the more it builds up in O_3. Equations (4) and (5) enforce that the works are started at some stage, and in time to finish before the last year ends. Equations (6) and (7) define auxiliary variables and equation (9) are operational constraints which avoid excessive delays in train circulation when a line undergoes multiple works at the same time.

It should be noted that the structure of the operational restrictions (9) allows to model some cases of track closure, namely those in which the movement of people and goods along the closed track section is made by alternative transportation. The only modification is the D_i value, which is usually higher than that caused by reduced speed circulation. In highly congested lines, or lines with feeder branches, the D_i delays may eventually cause knock-on effects (bottlenecks) in circulation. This does not happen in case study 1, but if such effects are plausible in other instances, modifications to (9) might need to be considered.

3 Case studies and results

3.1 Case study 1 – real data

Case study 1 consists of $M = 20$ sections to be renewed, over $P = 5$ years ($N = 60$ months) and belonging to $Q = 17$ lines. The parameters that characterize the sections were obtained by averaging values of their constituent homogeneous subsections, weighted by the length of the latter. The infrastructure management company provided all the data and validated the parameterization mentioned below.

The extra maintenance cost structure considers a negative exponential degradation of the infrastructure, which leads to extra maintenance costs of +3.5% per year on the current maintenance cost, for each year in which the renewal exceeds the recommended term, i.e. for every month j belonging to year k one has $C_{ij}^{EM} = C_{base} \times \left[(1 + 0.35)^{(\alpha_i - 1 + k) \times \theta(\alpha_i - 1 + k)} - 1\right]$, with α_i the number of years for which renewal is overdue and $\theta(x)$ the unit step function. In the case study α_i was 10 years, on average.

Priorities were defined considering the type of service provided by the line (TS) to which each section belongs, the sections present conservation status (CS) and freight traffic volume (FT). Values of 100/90/75/50 for TS and CS, and 100/90/75/50/40 for FT were considered and the final value for priorities was defined by $P_i = 0.5TS + 0.3CS + 0.2FT$. All these parameter values were suggested by the infrastructure management company.

Finally, delays in circulation were calculated considering the length of the sections and maximum train speed under works. Maximum values M_l and works duration T_i were obtained directly from the infrastructure management company.

The Pareto front of the case study was obtained by the epsilon-constraint method (Cohon, 1978) using the IBM CPLEX 12.7 solver, running on a quad-core @ 2.6 GHz CPU. Starting from solutions with O_1 restricted to its smallest possible value and gradually relaxing this value until reaching unrestricted O_1, two solutions were generated for each O_1 value, respectively minimizing O_2 and O_3. Solutions near O_1 optima took a few hours to derive, and were used as starting point for sequent runs, which gradually finished faster, down to just a few seconds per solution. The total CPU time was less than 1 day, for 312 runs. It was found that in all the solutions obtained, the value of O2 never exceeded its optimum by more than 1%, so this objective was discarded, giving rise to the front of Fig. 1 below (values in percentage, for confidentiality reasons, with optimum = 100%):

As can be seen, the front shows a relatively regular behavior, allowing the decision maker to analyse the trade-offs between equitably distributing the investment and accelerating the renewals. The non-dominated solutions that form the front may, for field works planning purposes, be displayed as Gantt schedules. Fig. 2 below shows the schedule for the solution with O_1 <120%, min O_3. Several non-dominated solutions, including this one, were presented to the infrastructure management company and are currently under evaluation for field implementation.

3.2 Case study 2 – large-sized theoretical problem

A large instance was generated, reflecting a problem of size similar to the USA railway network. This is the largest network in the world [10] so it is not expected that considerably larger problems appear in real life. In practice the US market is highly fragmented, i.e. split into several, independent infrastructure management companies, so this instance is purely hypothetical. It was carried out not only to stress-test the model in terms of CPU times, and thus unravel eventual limits to the computational performance of the model, but also to find out under what circumstances objective O_2 becomes important. Field data associated to railway network was randomly generated and the same parameterization of case study 1 was used. However, for case study 2 the α_i were distributed so as to have an average of 25 years backlog and a $P = 10$ years of project horizon was considered. Despite the very large increase in the number of decision variables (now about 600000), the CPU time increase was not very significant, with most runs taking in the range of seconds and runs close to O_1 optimum taking more CPU time (in fact only 4 solutions required more than 20 seconds: 20.7, 22.3, 415.6 and 1412.6 seconds), which was already the case for case study 1. This is a reasonable increase for a problem that is almost 200 times as large. It is thus expectable that just about any real-life problem can be treated in a modern computer, regardless of size.

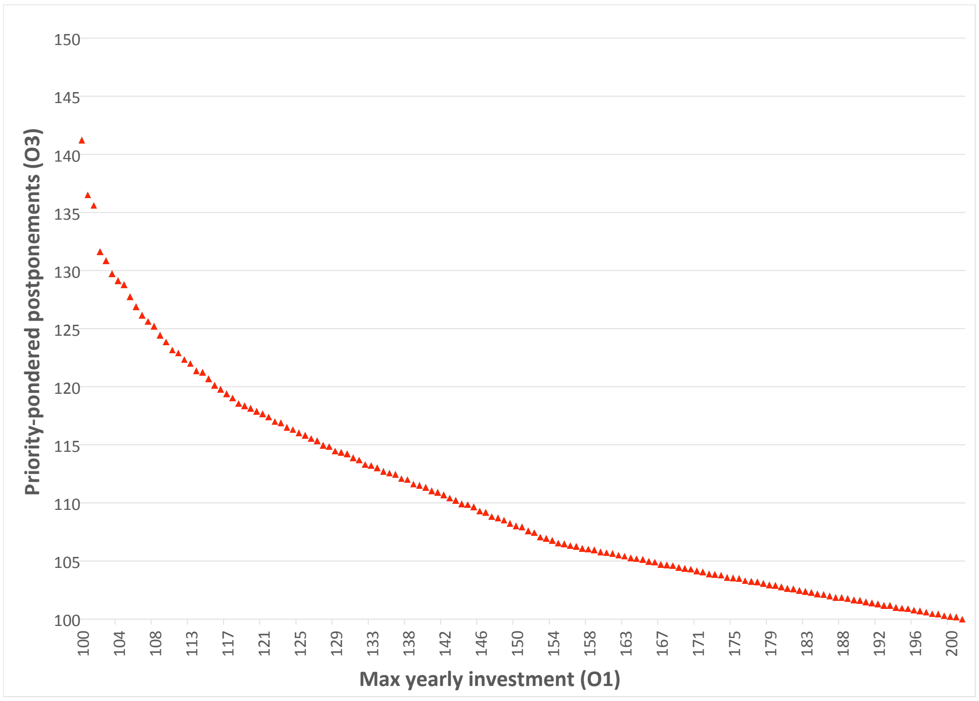


Figure 1 Pareto front for the case study (O_2 not displayed).

As compared to case study 1, in case study 2 optimizing O_1 now leads to greater (percent-wise) degradation of O_2 and O_3, whereas optimizing O_2 and O_3 lead to similar pay-off values. Objective O_2 is now relevant, fluctuating between 100% and 210% (rather than just the 1% of case study 1), showing all objectives are important when the infrastructure is ageing, and the backlog is large. Indeed, if the railway infrastructure is very degraded, objective O_2 should be included in the analysis, especially if the renewal plans span for many years.

Figure 3 shows that if the decision maker allows some increase in max yearly investment (i.e. degradation of O_1), solutions improve considerably in the remaining two objectives. It also shows that, for each value of the O_1 restriction, O_2 and O_3 can only fluctuate in a narrow range of values, making O_1 a very important objective, whose value has a big influence on the two other.

4 Conclusions and summary

In this paper, a multiobjective methodology was proposed for renewal of railway networks planning. The model is linear, soluble in reasonably time and provides a range of solutions for the analysis of trade-offs by the decision maker, each one being translatable in Gantt schedules for later implementation on the field. The methodology is strongly inspired by a real case study and reflects the practice of an infrastructure management company, so it may be especially useful as an asset management tool. It is also easily generalizable to other types of infrastructure, such as highways.

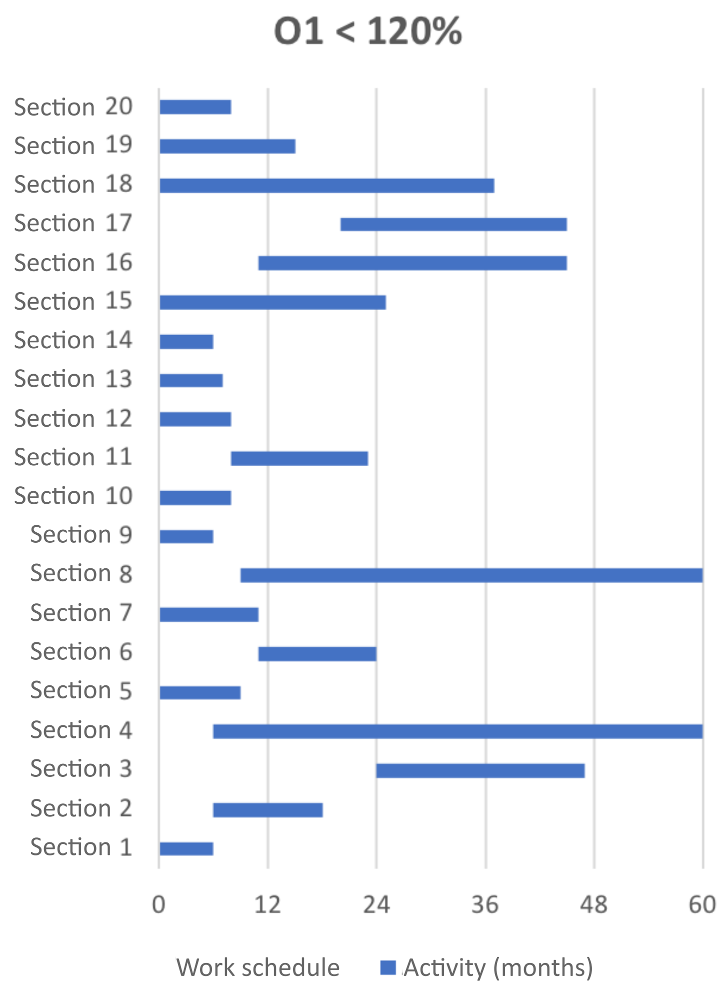


Figure 2 Gantt chart for solution $\min O_3$ with $O_1 < 120\%$.

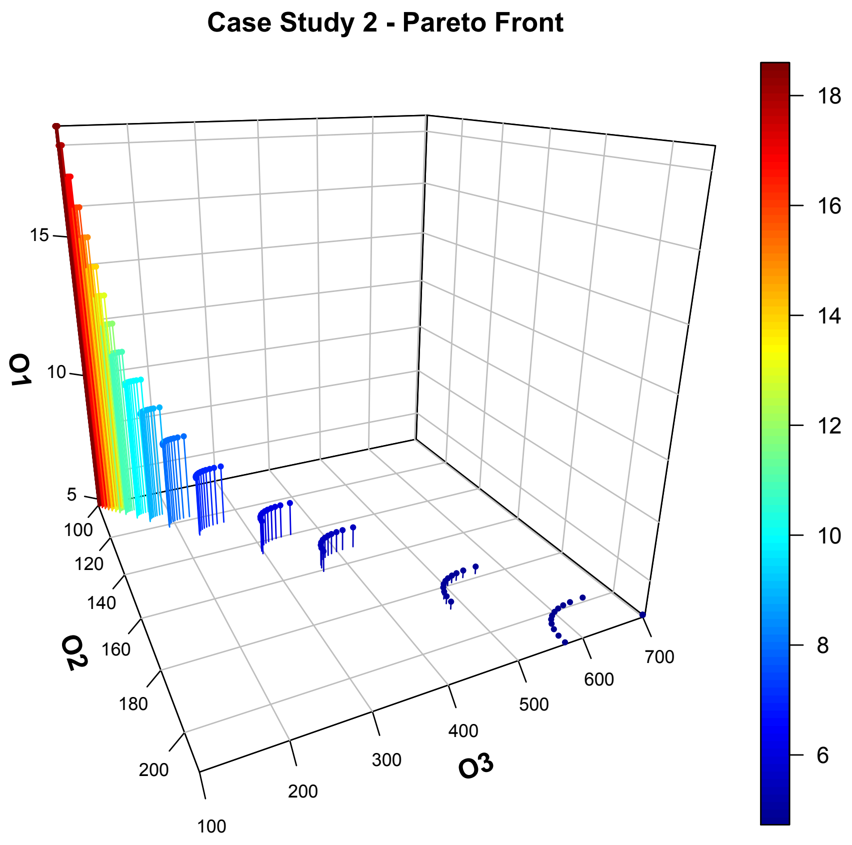


Figure 3 Results for the large-sized instance.

References

1 António R. Andrade and Paulo F. Teixeira. Biobjective optimization model for maintenance and renewal decisions related to rail track geometry. *Transportation Research Record: Journal of the Transportation Research Board*, 2261:163–170, 2011. doi:10.3141/2261-19.

2 David Banister and Mark Thurstain-Goodwin. Quantification of the non-transport benefits resulting from rail investment. *Journal of Transport Geography*, 19(2):212–223, 2011. doi:10.1016/j.jtrangeo.2010.05.001.

3 Lucio Bianco, Massimiliano Caramia, and Stefano Giordani. Resource levelling in project scheduling with generalized precedence relationships and variable execution intensities. *OR Spectrum*, 32(2):405–425, 2016. doi:10.1007/s00291-016-0435-1.

4 Gabriella Budai, Dennis Huisman, and Rommert Dekker. Scheduling preventive railway maintenance activities. *Journal of the Operational Research Society*, 57(9):1035–1044, 2006. doi:10.1057/palgrave.jors.2602085.

5 Marc Gaudry, Bernard Lapeyre, and Émile Quinet. Infrastructure maintenance, regeneration and service quality economics: A rail example. *Transportation Research Part B: Methodological*, 86:181–210, 2016. doi:10.1016/j.trb.2016.01.015.

6 IMPROVERAIL. Deliverable 10: Project handbook. improved tools for railway capacity and access management, 2003. URL: http://www.transport-research.info/sites/default/files/project/documents/20060727_145926_44007_IMPROVERAIL_Final_Report.pdf.

7 RailNetEurope. Glossary of terms related to railway network statements, 2016. URL: http://www.rne.eu/rneinhalt/uploads/RNE_NetworkStatementGlossary__V8_2016_web.pdf.

8 Julia Rieck, Juergen Zimmermann, and Thorsten Gather. Mixed-integer linear programming for resource leveling problems. *European Journal of Operational Research*, 221(1):27–37, 2012. doi:10.1016/j.ejor.2012.03.003.

9 René Schenkendorf, Joern C. Groos, and Lars Johannes. Strengthening the rail mode of transport by condition based preventive maintenance. *IFAC-PapersOnLine*, 48(21):964–969, 2015. 9th IFAC Symposium on Fault Detection, Supervision andSafety for Technical Processes SAFEPROCESS 2015. doi:10.1016/j.ifacol.2015.09.651.

10 Statista. Length of railroad network in selected countries around the world as of 2015, 2018. URL: https://www.statista.com/statistics/264657/ranking-of-the-top-20-countries-by-length-of-railroad-network.

11 Sander van Aken, Nikola Bešinović, and Rob M.P. Goverde. Designing alternative railway timetables under infrastructure maintenance possessions. *Transportation Research Part B: Methodological*, 98:224–238, 2017. doi:10.1016/j.trb.2016.12.019.

12 Min Wen, Rui Li, and Kim B. Salling. Optimization of preventive condition-based tamping for railway tracks. *European Journal of Operational Research*, 252(2):455–465, 2016. doi:10.1016/j.ejor.2016.01.024.

13 Allan Woodburn. An analysis of rail freight operational efficiency and mode share in the british port-hinterland container market. *Transportation Research Part D: Transport and Environment*, 51:190–202, 2017. doi:10.1016/j.trd.2017.01.002.

14 Zhang Xueqing and Gao Hui. Determining an optimal maintenance period for infrastructure systems. *Computer-Aided Civil and Infrastructure Engineering*, 27(7):543–554, 2012. doi:10.1111/j.1467-8667.2011.00739.x.

How to Measure the Robustness of Shunting Plans

Roel van den Broek
Department of Computer Science, Utrecht University
Utrecht, The Netherlands
r.w.vandenbroek@uu.nl

Han Hoogeveen
Department of Computer Science, Utrecht University
Utrecht, The Netherlands
j.a.hoogeveen@uu.nl

Marjan van den Akker
Department of Computer Science, Utrecht University
Utrecht, The Netherlands
j.m.vandenakker@uu.nl

Abstract

The general problem of scheduling activities subject to temporal and resource constraints as well as a deadline emerges naturally in numerous application domains such as project management, production planning, and public transport. The schedules often have to be implemented in an uncertain environment, where disturbances cause deviations in the duration, release date or deadline of activities. Since these disruptions are not known in the planning phase, we must have schedules that are robust, i.e., capable of absorbing the disturbances without large deteriorations of the solution quality. Due to the complexity of computing the robustness of a schedule directly, many surrogate robustness measures have been proposed in literature. In this paper, we propose new robustness measures, and compare these and several existing measures with the results of a simulation study to determine which measures can be applied in practice to obtain good approximations of the true robustness of a schedule with deadlines. The experiments are performed on schedules generated for real-world scheduling problems at the shunting yards of the Dutch Railways (NS).

2012 ACM Subject Classification Computing methodologies → Planning under uncertainty

Keywords and phrases robustness, resource-constrained project scheduling, partial order schedule, uncertainty, Monte Carlo simulation, train shunting

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.3

1 Introduction

For the shunting yards operated by the *Dutch Railways (NS)*, the largest passenger railway operator in the Netherlands, human planners create daily *shunting plans* describing all the activities that have to be performed, such as cleaning, maintenance, parking, and movements of trains. The objective for the planners is to construct a schedule in which all service activities on a train are completed before its deadline, which is the scheduled departure time. However, since arriving trains might be delayed, and performing a service activity can take longer than expected in practice, a shunting plan that is feasible with respect to the nominal timetable and activity durations may become infeasible during operation if a departure from the shunting yard is delayed due to disturbances.

Since the exact disturbances that will occur during the execution of a schedule are not known in advance, a practical strategy to handle the uncertainty is to construct a baseline schedule and a simple scheduling policy that adapts the initial schedule to the disruptions. The baseline schedule of a shunting plan of the NS is a partial order schedule of all activities, with precedence relations to ensure that any execution of the plan is *resource feasible*. The scheduling policy for the shunting plans is the *Earliest Start Time (EST)* or *Right Shift* policy, which assigns activities to their earliest start time in the baseline schedule, and, in case of disruptions during operation, delays activities that have not yet started as much as necessary while maintaining the ordering in the baseline schedule. Operational disturbances that cannot be absorbed by the shunting plan with the EST policy have to be handled by the human planners. The Dutch Railways prefers robust shunting plans that require little rescheduling during the operational phase, as the ad-hoc modifications to the schedule made by human planners often have a cascading effect in other parts of the shunting plan. A quantitative metric of this preference is the probability that the execution of a baseline plan will result in a delayed train departure.

In order to find robust baseline schedules for scheduling problems with deadlines, we have to determine a priori whether a schedule will perform well in the uncertain operational environment. However, the robustness of a schedule depends heavily on the available knowledge of the uncertainty: which elements in the scheduling problem can be disrupted by uncontrollable factors, and what are the distributions of those events? Often, the uncertain elements are known during the planning stage, but data on the distribution of the uncertainty are lacking. As a result, the robustness of a schedule is hard to compute in general, and assumptions on the uncertainty have to be made.

An approach often used to estimate the robustness is to simulate the performance of the schedule in many different scenarios sampled from the (assumed) distributions of the uncertainty. Although simulation is a powerful and versatile tool that gives an accurate estimate if a sufficient number of samples is used, it tends to be a computationally expensive technique. As solution methods for scheduling problems typically evaluate a large number of schedules to find the (near-)optimal solution, using simulation as a subroutine in the solution method might not be feasible. Therefore, several robustness measures that act as a surrogate for the sampled robustness of a schedule have been developed in the past few decades.

The contribution of this paper is to identify robustness measures that both properly predict the robustness of a schedule subject to deadlines, and can be evaluated efficiently. To this end, we generate a large number of schedules for real-world instances of the shunting yards operated by the NS. We perform a Monte Carlo simulation of schedules with uncertainty to obtain a good approximation of the robustness, and compare the outcome with the predictions of the robustness measures to determine if any of the estimations show a strong correlation with the sampled robustness. We base the comparison on two performance metrics, which are the fraction of delayed schedules, and the average lateness of the schedules.

The remainder of this paper is organized as follows. We start in Section 2 with a review of related work on robustness in resource-constrained project management, followed by a brief summary of the common concepts and notation in this paper in Section 3. We provide in Section 4 an overview of several robustness measures from literature, and propose some new, path-based measures. The instances provided by the NS as well as the setup of the Monte Carlo simulation study are discussed in Section 5. We compare the predictions of the robustness measures with the results of the simulation study in Section 6, and finish with concluding remarks and potential directions for further research in Section 7.

2 Literature overview

Most of the robustness measures proposed in literature are for resource-constrained project scheduling problems, where the standard objective is to minimize the makespan of the schedule. These measures are mainly based on the concept of *slack*. The *total slack* is defined as the the maximum amount of time that we can delay an activity without increasing the makespan of the total schedule, whereas *free slack* is the amount of time by which an activity can be delayed without delaying any other activity in the schedule.

A simple slack-based robustness estimation, proposed by [6], is to compute the average of the total slacks of all activities. By simulating many realizations of job shop schedules, [6] showed that a large percentage of the variation in the realized makespan was explained by the average slack of the schedule. Similarly, [1] proposed the sum of free slacks as a robustness measure.

Based on the observation that, in addition to the total amount of slack, the distribution of the slack over the schedule affects the robustness as well, [3] proposed several variants of the sum of free slacks. These robustness measures weigh the free slack by the number of successors, and substitute the free slack with a binary slack indicator function or an upper bound on the slack based on the activity duration.

The relation of a number of existing and newly proposed robustness measures to the fraction of feasible schedule realizations in a Monte Carlo simulation has been investigated by [5]. For instances of the discrete time/cost trade-off problem, they reported high values (> 0.91) of the coefficient of determination for the sum of total slacks measure and successor-weighted variants of it.

A similar comparison of robustness metrics in a Monte Carlo simulation was performed by [2]. In contrast to the work of [5], their results showed that summing the unweighted slack of the activities has at best a weak correlation with the expected makespan of the schedule.

When scheduling activities subject to deadlines, the primary objective is to find a feasible schedule. However, the concepts of free and total slack do not fully capture the slack of a schedule with respect to its deadline. To quantify this type of slack, we can view a schedule with deadlines as a special case of a *Simple Temporal Network (STN)*, which is a directed graph with both minimum and maximum time lags on the arcs that was introduced by [4]. For this type of graph, there are *flexibility metrics* that aggregate the slack with respect to all the temporal constraints, including the deadlines. The *naive flexibility* of an STN is the sum of the difference between the latest and earliest start time of each activity, i.e., the total slack relative to the deadline instead of the makespan of the schedule. Analog to the free slack of an activity, [15] proposed the *concurrent flexibility* metric, which is based on interval schedules. An interval schedule specifies for each activity an interval such that every activity can start at any time within its interval independently of the other events, and without exceeding the deadline of the schedule. The concurrent flexibility of an STN is defined as the maximal sum of the interval lengths over all possible interval schedules. A linear programming formulation was proposed by the authors to compute the concurrent flexibility. It was shown in [14] that a schedule with a high flexibility is not always robust to disruptions.

The limitations of the sum of free slacks metric were discussed by [8], and they proposed to use the minimum free slack over all activities as a robustness measure for schedules with a deadline, and provided an algorithm that maximizes the minimum free slack by distributing the free slack evenly over the schedule. Their approach is essentially the concurrent flexibility metric, proposed by [15], with as objective to maximize the minimum interval length instead of the sum of the intervals.

An extensive comparison of robustness measures can be found in the paper of [7]. They investigated the correlation between the surrogate robustness measures and the probability that the completion time of a schedule exceeds its nominal makespan, which was approximated using a Monte Carlo simulation. Their results showed that the strongest correlation ($R^2 > 0.64$) with the robustness performance metric in the simulation was achieved with a robustness measure that computes the *slack sufficiency*, which is based on the ratio between the free slack and the processing time of an activity.

Despite the many surrogate robustness measures in literature, there is no consensus on which of these provides a good approximation of the true robustness of a schedule. In the simulation studies of [6], [5], [2] and [7], only schedules without deadlines are considered, focusing mainly on the expected makespan and related performance metrics. However, a good expected makespan of a schedule constrained by a deadline does not necessarily imply that the schedule will respect its deadline. Therefore, additional research is required to verify their results for schedules with deadlines.

3 Preliminaries

In this paper, we consider the general resource-constrained scheduling problem with deadlines. For a scheduling problem with activities 1 to n, and a deadline at time T for all activities, we define a *baseline schedule* σ as a pair $(S_\sigma, \mathcal{POS}_\sigma)$, where $S_\sigma = \{0, \dots, n+1\}$ is the *activity set*, and $\mathcal{POS}_\sigma$ a *partial order schedule* of these activities:

$$\mathcal{POS}_\sigma = \{i \prec j \mid \forall i, j \in S_\sigma : i \text{ directly precedes } j \text{ in } \sigma\}.$$

In this schedule, the activities 0 and $n+1$ are dummy activities representing the start and end of the schedule, respectively; the precedence relations needed to ensure that all activities take place between the start and the end activity are contained in the partial ordering.

Each activity i has a nominal processing time $p_i \in \mathbb{R}_+$, with $p_0 = p_{n+1} = 0$. We assume that the release date of each activity is equal to 0, and that all activities, in particular $n+1$, have to be finished before the deadline T. Note that scheduling problems with an individual release date or deadline of activity i can still be modeled in the schedule by adding a dummy activity between i and the start or end activity, respectively.

From the baseline schedule σ, we can compute for each activity i the time window in which it has to be processed. The *earliest start time* est_i^σ is the earliest possible time at which all predecessor activities can be finished. Similarly, the *latest finish time* lft_i^σ is equal to the latest possible completion time of activity i such that the schedule remains feasible with respect to the deadline. The *latest start time* lst_i^σ and *earliest finish time* eft_i^σ can be derived from the latest or earliest counterpart by subtracting or adding the processing time p_i, respectively. The earliest finish time of activity $n+1$ is known as the *makespan* or $C_{\max}$.

The concept of *slack* is commonly used to quantify the robustness of a schedule. We define the *total slack* ts_i^σ of activity i in schedule σ as the maximum amount of time by which we can delay the activity such that no deadlines are exceeded in the schedule. Equivalently, the slack is the difference between the earliest and latest start time of the activity, $ts_i^\sigma = lst_i^\sigma - est_i^\sigma$. Note that this definition differs slightly from the formulation given in 2, where the total slack is computed with respect to the makespan instead of the schedule deadline. However, for each activity in the schedule, the difference in slack between the two definitions is a constant, namely $T - C_{\max}$. A different type of slack is the *free slack* fs_i^σ, which is the maximum amount of time that activity i can be delayed without affecting any other activity. That is, we define the free slack as

$$fs_i^\sigma = \min_{j \in succ_\sigma(i)} \{est_j\} - eft_i, \tag{1}$$

where $succ_\sigma(i)$ are the successor activities of i in the schedule.

An intuitive graph-based representation of the partial ordering can be constructed by modeling each activity i in activity set as a vertex v_i, and adding for every precedence relation $i \prec j \in \mathcal{POS}$ an arc from v_i to v_j to the graph. The result is a digraph $G_\sigma = (V_\sigma, A_\sigma)$, in which a directed path represents a set of activities that have to be performed sequentially. Similar to slack of an activity, we define the slack of a path $\pi = (\pi_1, \ldots, \pi_k)$ as

$$s_\pi^\sigma = lft_{\pi_k}^\sigma - est_{\pi_1}^\sigma - \sum_{i \in \pi} p_i,$$

which is the maximal amount of time we can delay activities on the path without exceeding the deadline of π_k.

Shunting plans

The schedules that we use our experiments are solutions to a scheduling problem of the Dutch Railways that arises at shunting yards, which are networks of tracks connected by switches that contain facilities providing services such as cleaning and maintenance to the trains. In this scheduling problem, which is a variant of the *Train Unit Shunting Problem* described in [13], we have a number of train units that arrive during the evening on the shunting yard. These arrivals happen according to a static timetable, which lists the arrival time as well as the train – a sequence of coupled train units – in which each train unit arrives. The train units have to leave the shunting yard the next morning, again based on the timetable. Note that the arrival train of a train unit is not necessarily the same as the departure train. During their stay at the shunting yard, the train units have to move through different facilities to receive service tasks such as cleaning and maintenance, and must be parked on an appropriate track to wait until departure.

The scheduling problem at the shunting yards is then to construct a *shunting plan*, which is a schedule that describes all the activities on the shunting yard such as coupling and decoupling train units, service tasks and train movements, such that the service tasks of each train unit in a departing train are completed before the departure time, and none of the resource capacity constraints are exceeded in the shunting plan. A more in-depth description of the scheduling problem can be found in [13].

4 Robustness measures

In this section we discuss the robustness measures that we will compare in our experiments. Surrogate robustness measures that rely heavily on the exact distribution of the uncertainty in a schedule might produce accurate predictions of the robustness, but their applicability to real-world scheduling problems is limited, since quantitative data of the uncertainty are often scarce in practice. Therefore, robustness measures with a low dependency of the available knowledge of the uncertainty are preferred.

Robustness measures are usually created with the assumption that the nominal or expected processing time of the activities is known. If a robustness measure does not rely on any other information about the uncertainty, the robustness is solely estimated from the structure of the baseline schedule. The two robustness measures applied most often in literature, namely the sum of total slacks ([6]),

$$RM_1(\sigma) = \sum_i ts_i^\sigma, \tag{2}$$

and the sum of free slacks ([1]),

$$RM_2(\sigma) = \sum_i fs_i^\sigma, \tag{3}$$

are examples of measures that depend only on the nominal activity durations. Furthermore, the standard resource-constrained scheduling objective, the makespan or, equivalently, the minimum total slack,

$$RM_3(\sigma) = \min_i ts_i^\sigma, \tag{4}$$

can be viewed as an expectation-only robustness measure as well.

Another robustness measure of this type that was shown by [7] to provide good estimations of the robustness of a schedule was based on *slack sufficiency*, which compares the free slack of an activity to a fraction of the duration of that activity or one of its predecessors in the schedule. In the work of [7], this robustness measure is defined as

$$RM_4(\sigma) = \sum_i |\{j \mid j \in prec_\sigma(i) \cup \{i\}, fs_i \geq \lambda p_j\}| \tag{5}$$

where $prec_\sigma(i)$ are the predecessors of activity i in the schedule and $0 < \lambda < 1$. The authors suggested that λ should be set to the expected deviation from the nominal processing time of the activities due to disruptions.

A more complex robustness measure depending only on the expected activity duration is the interval schedule based approach of [15]. It finds the maximal assignment of intervals to activities such that each activity i can be scheduled within its interval (e_i, l_i) independently of the other activities. To achieve this, the intervals are computed with the linear program

$$\begin{aligned} RM_5(\sigma) = {} & \max \sum_i (l_i - e_i) \\ & \text{subject to} \\ & \quad est_i^\sigma \leq e_i \leq s_i \leq lst_i^\sigma \qquad \forall i \\ & \quad l_i + p_i \leq e_j \qquad\qquad\quad\ \forall i \prec j \in \mathcal{POS}_\sigma. \end{aligned} \tag{6}$$

As an alternative to solving this linear program, [10] formulated a matching problem based on the dual problem.

Analog to the work of [8], we can change the objective of the linear program of [15] to maximize the minimum interval, which will result in a more evenly distributed interval schedule. The linear program then becomes

$$\begin{aligned} RM_6(\sigma) = {} & \max \min_i (l_i - e_i) \\ & \text{subject to} \\ & \quad est_i^\sigma \leq e_i \leq l_i \leq lst_i^\sigma \qquad \forall i \\ & \quad l_i + p_i \leq e_j \qquad\qquad\quad\ \forall i \prec j \in \mathcal{POS}_\sigma. \end{aligned} \tag{7}$$

In contrast to the previous surrogate robustness measures, the measure of [15] focuses on the entire graph structure instead of just the slack of the individual activities. However, an optimal solution to the linear program is an interval schedule that assigns large intervals to concurrent activities, and, consequently, only small intervals to sequential activities, thus overemphasizing parallel activities.

A compromise between activity-based and schedule-based measures is to predict the robustness of a schedule from the paths in the partial ordering. Without any knowledge of the uncertainty, it is reasonable to assume that the likelihood of a disruption on a path increases with the number of activities of the path. Therefore, we propose to use the minimum over all paths of the path slack divided by the number of activities on the path as a robustness measure,

$$RM_7(\sigma) = \min_{\pi} \left\{ \frac{s_\pi^\sigma}{|\pi|} \right\}. \tag{8}$$

Although the number of paths can be exponentially large, we can evaluate this robustness measure efficiently by computing for each $k = 1$ to $n + 2$ the shortest path in the schedule with exactly k activities.

In many cases, a reasonable estimate of the variance of the uncertainty can be made as well, even if the exact distribution of the uncertainty is unknown. We can exploit this additional information by making the assumption that the duration of each activity is normally distributed, as normal distributions can be characterized solely by their mean and variance. Although this assumption might be wrong for the distribution of the duration of a single activity, if follows from the central limit theorem that the sum of activity durations does resemble a normal distribution. Therefore, we can approximate the uncertainty in the duration of a path in the schedule.

We can utilize this approximation as the basis for several robustness measures. Firstly, we propose another path-based robustness measure. Analog to the minimum weighted path slack in RM_7, we use the minimum probability that a path can be completed within the deadline, computed over all the possible paths in the graph. That is, we compute for each path π the normal distribution approximation X_π of the duration of the path by summing the processing time distributions of activities on that path, and report the minimum probability of completion before the deadline:

$$RM_8(\sigma) = \min_{\pi} \left\{ P\left(X_\pi \leq T\right) \right\} \tag{9}$$

Although the paths in the schedule are connected by precedence relations, they are assumed to be independent by this robustness measure.

In contrast to RM_7, we might have to evaluate all the paths in the schedule to compute the distribution-based robustness measure RM_8, since the usual graph-theoretical properties of paths, such as the property that any sub-path of a shortest path is again a shortest path, do not hold in this case. To keep the computation tractable, we construct in topological order for each activity i the set of paths in schedule σ ending at i, from the paths ending at the immediate predecessors of i:

$$\Pi_i = \left\{ (\pi^1, \ldots, \pi^k, i) \mid \exists\, j \prec i \in \mathcal{POS}_\sigma : (\pi^1, \ldots, \pi^k) \in \Pi_j \right\}. \tag{10}$$

Furthermore, we compute the distribution X_π of the duration of each $\pi \in \Pi_i$ by summing the normal distributions of the activities on the path. We can then reformulate RM_8 to

$$RM_8(\sigma) = \min_{\pi \in \Pi_{n+1}} P\left(X_\pi \leq T\right). \tag{11}$$

To avoid the exponential growth of Π_i, we repeatedly remove the path π from Π_i with the smallest probability of exceeding any other path in Π_i, until the set of paths is at most size K. Ties are broken randomly in this pruning procedure, and a maximum size of $K = 8$ was shown to be sufficiently large to achieve a good robustness estimation in preliminary experiments.

Another approach is to estimate the distribution of total makespan of the schedule. Many approximation algorithms have been proposed in literature, see [9] for a comparison of several techniques. An efficient method to construct an approximation of the makespan distribution is to evaluate the activities in topological order, computing the makespan distribution Y_i up to each activity i as the distribution of the maximum over the makespan distributions of the immediate predecessors of i

$$Y_i = \max_{j \prec i \in \mathcal{POS}_\sigma} \{Y_j\} + D_i, \tag{12}$$

where D_i is the normal approximation of the activity duration of i, and the maximum over the predecessor distributions is approximated with a normal distribution as proposed by [11]. The robustness measure, which is proposed in the work of [12], is then

$$RM_9(\sigma) = P\left(Y_{n+1} \leq T\right). \tag{13}$$

5 Experimental setup

The main application of surrogate robustness measures is in the comparison of schedules, since these can estimate the true robustness of a schedule far more efficiently than other approaches such as simulation. However, we need to investigate whether the estimations correctly reflect the relative ordering of schedules according to their robustness to verify that the robustness estimators are actually suitable for this purpose. To accomplish this, we construct empirical makespan distributions of a set of realistic schedules in a simulation study, and search for robustness estimators that show a strong correlation with the empirical results.

We have selected two real-world instances of the shunting problem described in Section 3 as the basis of our simulation study. The first one originates from *"Kleine Binkchorst (KBH)"*, which is a shunting yard of the NS near the central station of The Hague. It consists of a single night during which 19 train units arrive at the yard. These train units need to receive internal cleaning and a maintenance inspection; three of them need to be washed as well. Due to all the necessary train movements, shunting plans of problem instance typically have close to 160 activities, with 250 to 300 precedence relations. The other instance is obtained from a shunting yard near Utrecht, named *"OZ"*, which contains, contrary to the KBH, many dead-end tracks. As a result, the main difficulty in the scheduling problem is the parking order of the trains. This instance has 16 train units and a total of 27 service activities. The number of activities in the corresponding shunting plans ranges from 140 to 160 activities, and roughly 300 precedence relations.

For each of these two scheduling problems, we generated 500 feasible shunting plans with a local search algorithm that, starting with an infeasible initial solution, iteratively alters the current solution to resolve conflicts in the shunting plan. The meta-heuristic used in the local search framework is *simulated annealing*, which is a stochastic optimization technique that accepts with a small probability some deteriorations in the solution quality – the number of conflicts – due to local modifications. Initial solutions are generated by scheduling the service activities in a random order and assigning the trains to random parking tracks. Furthermore, the objective that is optimized by the local search consists only of the sum of the (weighted) conflicts, and the search process stops when a feasible shunting plan is found. Due to the stochastic nature of the solution method, this generation process produces a diverse set of shunting plans. See [13] for more details of the local search algorithm.

The main components of the uncertainty in the execution of a shunting plan are the arrival time of trains and the duration of service activities and train movements. Disturbances in the arrival time of a train are modeled with a uniform distribution with the mean equal to the scheduled arrival time, and an interval size of 10 minutes. The service activities and train movements always have a nonnegative duration, and the size of the disruptions are usually proportional to the duration of the activities. Therefore, we model the uncertainty in these activities with log-normal distribution with the nominal duration as the mean, and a standard deviation equal to 0.1 times the nominal duration. Robustness measures RM_1 to RM_7 use only the nominal durations in their computation, while RM_8 and RM_9 require the standard deviation of the distributions as well. Although for RM_4, the slack sufficiency measure, we can pick any value between zero and one for the fraction λ, we set it equal to the standard deviation of the uncertainty of the service and movement activities, i.e., $\lambda = 0.1$, as is suggested in [7].

The schedules are then evaluated by each of the surrogate robustness measures listed in Section 4 to generate their predictions of the robustness of the schedules. The predictions are compared with the results of a Monte Carlo simulation, which is a technique that repeatedly draws samples from the distribution of the uncertainty – thus simulating different realizations of the scheduling problem – to approximate the makespan distribution of the schedule. To obtain an accurate empirical distribution of the makespan, we collect 20000 samples per schedule.

Since the objective of the shunting yard planners at the NS is to find feasible shunting plans that minimize the probability of delayed departures, we use the fraction of samples in which the schedule realization resulted in the delay as the primary performance metric. Additionally, we compute the average lateness of the empirical makespan distribution to get a better understanding of the problem structure.

The correlation between the performance metrics and the robustness measures is investigated in the following section by computing both the *Pearson correlation coefficient* (r), and *Spearman's rank correlation coefficient* (ρ). If the robustness of a schedule can be approximated by a robustness measure, then a high value of the measure should indicate a low delayed fraction and average lateness, and we expect that the robustness measure will have a correlation coefficient close to -1 with either of the performance metrics. A coefficient of -1 for the Pearson correlation means that there is a perfect linear relation between the robustness measure and the performance metric, and a robustness measure with a Spearman correlation of -1 will rank the schedules perfectly according to the performance metric. The Spearman correlation coefficient is particularly suitable for our experiments, since the purpose of the robustness measures is to compare schedules efficiently.

In addition to the two robustness performance metrics, we record the time required by each robustness measure to evaluate the schedules to compare the computational efficiency of the measures. To obtain reliable estimates of these computation times, we evaluated each of the 500 schedules 100 times with every robustness measure, and compute the average computation time per evaluation.

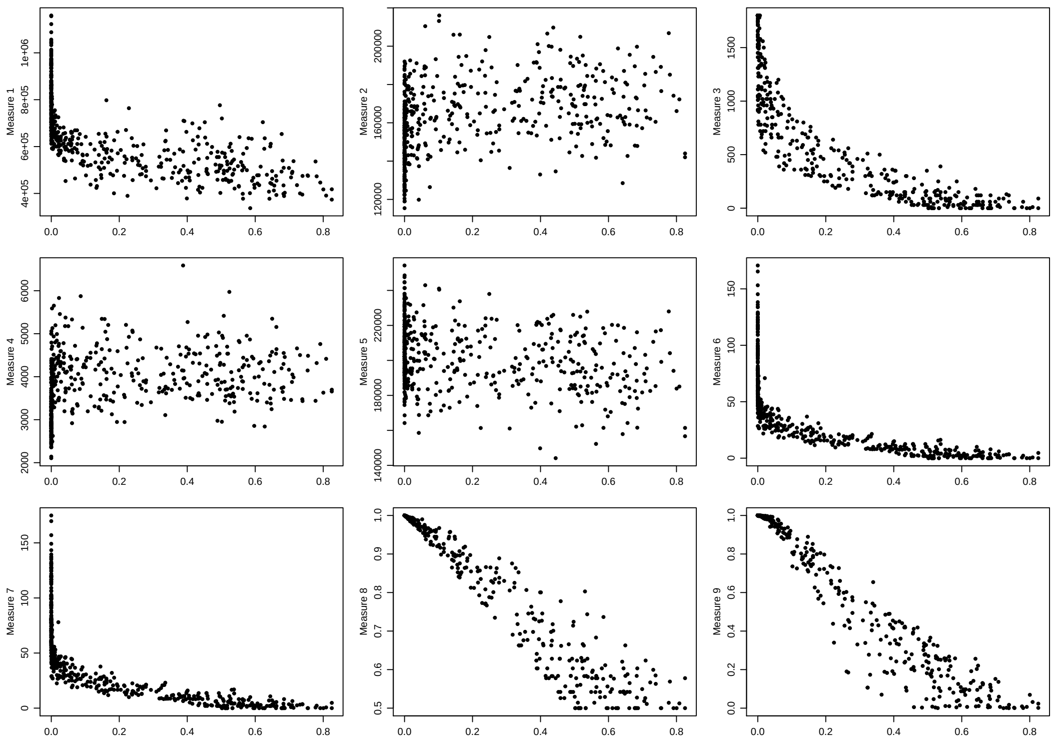


Figure 1 Scatter plots for the 9 robustness measures, showing the computed value of the measure (vertical axis) and the fraction of delayed samples (horizontal axis) of the KBH instances.

Table 1 The Spearman (ρ) and Pearson (r) correlation coefficients, as well as the average computation time, for the KBH instances. Coefficients close to -1 indicate that the robustness measure is a good approximation of the schedule robustness.

	Fraction delayed		**Average Lateness**		**Computation**
	ρ	r	ρ	r	**Time (ms)**
RM_1	-0.840	-0.663	-0.840	-0.828	0,01
RM_2	0.456	0.357	0.470	0.491	0,02
RM_3	-0.955	-0.838	-0.972	-0.990	0,01
RM_4	0.457	0.290	0.479	0.507	0,54
RM_5	-0.298	-0.293	-0.321	-0.326	3,95
RM_6	-0.964	-0.718	-0.955	-0.889	6,64
RM_7	-0.963	-0.719	-0.953	-0.887	0,87
RM_8	-0.982	-0.969	-0.972	-0.835	0,57
RM_9	-0.981	-0.971	-0.971	-0.846	0,23

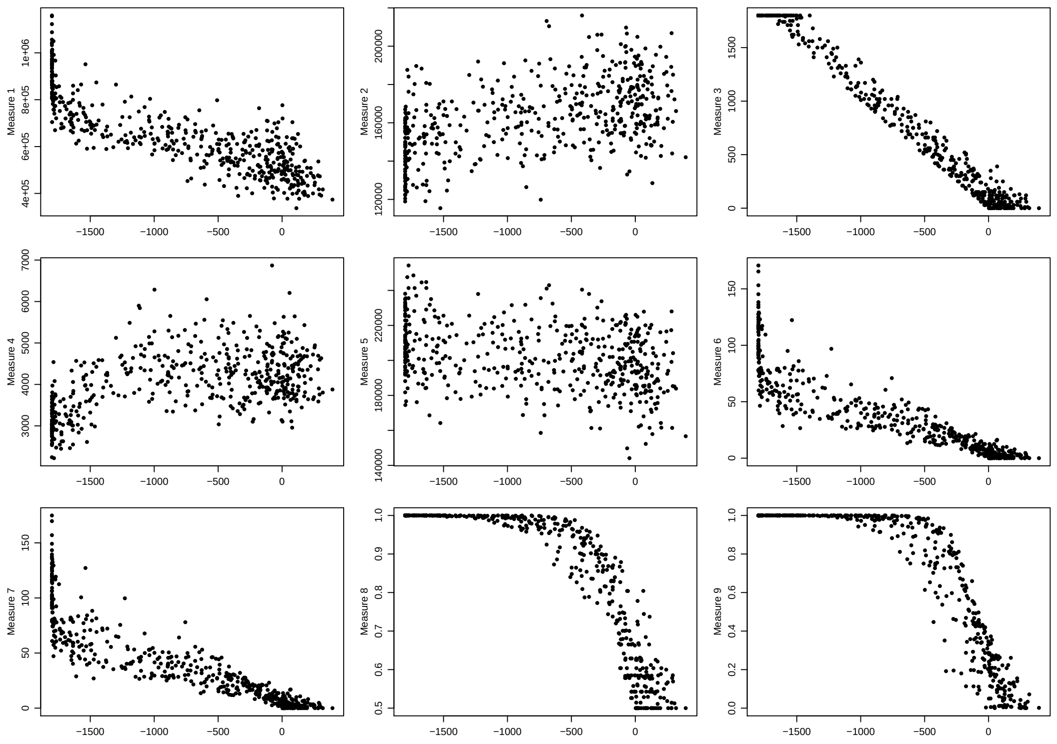


Figure 2 Scatter plots for the 9 robustness measures, showing the computed value of the measure (vertical axis) and the average lateness over the samples (horizontal axis) of the KBH instances.

Table 2 The Spearman (ρ) and Pearson (r) correlation coefficients, as well as the average computation time, for the OZ instances. Coefficients close to -1 indicate that the robustness measure is a good approximation of the schedule robustness.

	Fraction delayed		Average Lateness		Computation
	ρ	r	ρ	r	Time (ms)
RM_1	-0.933	-0.831	-0.943	-0.962	0,01
RM_2	0.544	0.510	0.542	0.606	0,01
RM_3	-0.975	-0.876	-0.980	-0.989	0,01
RM_4	0.156	0.132	0.161	0.270	0,53
RM_5	-0.118	-0.117	-0.127	-0.151	3,81
RM_6	-0.969	-0.840	-0.976	-0.972	6,28
RM_7	-0.968	-0.841	-0.975	-0.972	0,83
RM_8	-0.977	-0.959	-0.971	-0.818	0,60
RM_9	-0.972	-0.910	-0.960	-0.896	0,25

6 Empirical results

The relations between the robustness measures and the fraction of samples in which trains departed with a delay in the simulation study are shown in Figure 1 for the *Kleine Binckhorst* test set. Figure 2 shows the results for the average lateness performance metric of same instances. Tables 1 and 2 list the correlation coefficients of the robustness measures and the two performance metrics, as well as the average computation time, of the KBH and OZ instances.

The two robustness measures based on normal approximations, RM_8 and RM_9, appear to have the strongest rank correlation with both the performance metrics, clearly showing the advantage of exploiting the additional information of the variance of the uncertainty. Furthermore, both measures show a high Pearson correlation coefficient with the delayed fraction metric, as can be seen in Figure 1. If we take the computation time into account as well, then the approximation of makespan distribution, RM_9, would be the preferred robustness measure in a practical application.

When knowledge of the variance is not available, robustness measures that rely only on the nominal processing time of activities have to be used. Of those measures, RM_3, RM_6 and RM_7 are good choices in practice due to their high correlation with both performance metrics. In particular, the minimum total slack RM_3, which is equivalent to the makespan of a schedule, shows a remarkably strong Spearman correlation with the robustness performance metrics, and the correlation appears to be linear with the average lateness metric, which is an alternative formulation of the expected makespan. Given that the makespan can be computed more efficiently than the normal approximation methods, this robustness measure will most likely be sufficient to obtain robust solutions to scheduling problems with deadlines.

Contrary to the result of [7], the robustness measure RM_2, RM_4 and RM_5, which are based on maximizing the sum of the free slacks, correlate poorly to either of the performance metrics. This result is supported by the random scattering of the three measures in Figures 1 and 2. In the case of RM_2 and RM_4, the probability of delays in the schedule actually increases with the total amount of free slack in the schedule. Although the cause of this relation remains to be investigated, one possible explanation might be that free slack in these shunting plans mostly arises when a train is scheduled to wait until a route or a resource is available for its movement or service activity. Therefore, if a shunting plan contains many waiting trains, then the infrastructure or resources at the shunting yard are not used effectively, and the schedule will likely have a large makespan.

7 Conclusion

In this paper, we have studied robustness measures for shunting plans, which are solutions to the scheduling problem with deadlines that arises at shunting yards. The goal of the research is to identify measures that can accurately and efficiently estimate the robustness of a shunting plan, which is the likelihood that all trains depart on time from the shunting yard when disruptions occur in the operational phase. To achieve this goal, we have proposed new path-based robustness measures, and compared these, as well as several existing measures, with the results of a Monte Carlo simulation study on shunting plans for two real-world shunting problems of the Dutch Railways. We have shown that the new and existing robustness measures that utilize normally distributed approximations of the activity durations are strongly correlated with robustness of the schedules. Despite its simplicity, the makespan is also a good indicator of the robustness for schedules with deadlines. Contrary to earlier results on schedules without deadlines, the free slack has a poor predictive value of the robustness of shunting plans. Further research should be conducted to investigate the differences in robustness in scheduling problems that are subject to deadlines, and those that require the minimization of the makespan.

References

1 Mohammad A. Al-Fawzan and Mohamed Haouari. A bi-objective model for robust resource-constrained project scheduling. *International Journal of production economics*, 96(2):175–187, 2005.

2 Louis-Claude Canon and Emmanuel Jeannot. Evaluation and optimization of the robustness of dag schedules in heterogeneous environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):532–546, 2010.

3 Hédi Chtourou and Mohamed Haouari. A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. *Computers & industrial engineering*, 55(1):183–194, 2008.

4 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1-3):61–95, 1991.

5 Öncü Hazır, Mohamed Haouari, and Erdal Erel. Robust scheduling and robustness measures for the discrete time/cost trade-off problem. *European Journal of Operational Research*, 207(2):633–643, 2010.

6 V. Jorge Leon, S. David Wu, and Robert H. Storer. Robustness measures and robust scheduling for job shops. *IIE transactions*, 26(5):32–43, 1994.

7 Mohamed Ali Khemakhem and Hédi Chtourou. Efficient robustness measures for the resource-constrained project scheduling problem. *International Journal of Industrial and Systems Engineering*, 14(2):245–267, 2013.

8 Przemysław Kobylański and Dorota Kuchta. A note on the paper by Ma Al-Fawzan and M. Haouari about a bi-objective problem for robust resource-constrained project scheduling. *International Journal of Production Economics*, 107(2):496–501, 2007.

9 Arfst Ludwig, Rolf H. Möhring, and Frederik Stork. A computational study on bounding the makespan distribution in stochastic project networks. *Annals of Operations Research*, 102(1-4):49–64, 2001.

10 Simon Mountakis, Tomas Klos, and Cees Witteveen. Temporal flexibility revisited: Maximizing flexibility by computing bipartite matchings. In *ICAPS*, pages 174–178, 2015.

11 Saralees Nadarajah and Samuel Kotz. Exact distribution of the max/min of two gaussian random variables. *IEEE Transactions on very large scale integration (VLSI) systems*, 16(2):210–212, 2008.

12 Guido Passage, Han Hoogeveen, and Marjan van den Akker. Combining local search and heuristics for solving robust parallel machine scheduling. Master's thesis, Utrecht University, 2016. https://dspace.library.uu.nl/bitstream/handle/1874/334269/thesis.pdf.

13 Roel van den Broek, Han Hoogeveen, Marjan van den Akker, and Bob Huisman. A local search algorithm for train unit shunting with service scheduling, 2018. Manuscript submitted for publication.

14 Michel Wilson. *Robust scheduling in an uncertain environment*. PhD thesis, TU Delft, 2016.

15 Michel Wilson, Tomas Klos, Cees Witteveen, and Bob Huisman. Flexibility and decoupling in simple temporal networks. *Artificial Intelligence*, 214:26–44, 2014.

Robustness as a Third Dimension for Evaluating Public Transport Plans

Markus Friedrich
Institut für Straßen- und Verkehrswesen, Universität Stuttgart,
Pfaffenwaldring 7, D 70569 Stuttgart, Germany
markus.friedrich@isv.uni-stuttgart.de

Matthias Müller-Hannemann
Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg,
Von-Seckendorff-Platz 1, D 06120 Halle (Saale), Germany
muellerh@informatik.uni-halle.de
https://orcid.org/0000-0001-6976-0006

Ralf Rückert
Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg,
Von-Seckendorff-Platz 1, D 06120 Halle (Saale), Germany
rueckert@informatik.uni-halle.de

Alexander Schiewe
Institut für Numerische und Angewandte Mathematik, Universität Göttingen,
Lotzestr. 16-18, D 37083 Göttingen, Germany
a.schiewe@math.uni-goettingen.de

Anita Schöbel
Institut für Numerische und Angewandte Mathematik, Universität Göttingen,
Lotzestr. 16-18, D 37083 Göttingen, Germany
schoebel@math.uni-goettingen.de

Abstract

Providing attractive and efficient public transport services is of crucial importance due to higher demands for mobility and the need to reduce air pollution and to save energy. The classical planning process in public transport tries to achieve a reasonable compromise between service quality for passengers and operating costs. Service quality mostly considers quantities like average travel time and number of transfers. Since daily public transport inevitably suffers from delays caused by random disturbances and disruptions, robustness also plays a crucial role.

While there are recent attempts to achieve delay-resistant timetables, comparably little work has been done to systematically assess and to compare the robustness of transport plans from a passenger point of view. We here provide a general and flexible framework for evaluating public transport plans (lines, timetables, and vehicle schedules) in various ways. It enables planners to explore several trade-offs between operating costs, service quality (average perceived travel time of passengers), and robustness against delays. For such an assessment we develop several passenger-oriented robustness tests which can be instantiated with parameterized delay scenarios. Important features of our framework include detailed passenger flow models, delay propagation schemes and disposition strategies, rerouting strategies as well as vehicle capacities.

To demonstrate possible use cases, our framework has been applied to a variety of public transport plans which have been created for the same given demand for an artificial urban grid network and to instances for long-distance train networks. As one application we study the impact of different strategies to improve the robustness of timetables by insertion of supplement times. We also show that the framework can be used to optimize waiting strategies in delay management.

18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018).
Editors: Ralf Borndörfer and Sabine Storandt; Article No. 4; pp. 4:1–4:17
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Applied computing → Transportation, Mathematics of computing → Graph algorithms, Theory of computation → Network optimization

Keywords and phrases robustness, timetabling, vehicle schedules, delays

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.4

Funding This work has been supported by DFG grants for the research group FOR 2083.

1 Introduction

The design of attractive and efficient public transport services is a challenging problem of fundamental importance. The overall planning process is complex and involves many stages. We here focus on the planning stage where the basic infrastructure (stops, available tracks or roads) is already fixed and planners are interested in the design of a *public transport plan*, i. e. the design of a line network with a corresponding timetable and vehicle schedule. Traditionally, the primary optimization goals for public transport plans are operating costs on the one side, and service quality criteria like perceived travel time and number of transfers on the other. Robustness issues addressing the effect of possible disturbances on passengers, are often not considered at this stage of the planning process.

Goals and contribution. In this paper, we provide a general framework for the analysis of public transport plans, applicable to transport networks of all scales (city, regional, and long-distance) and different means of public transport (trains, trams, busses). Based on given passenger demands, our goal is to analyze robustness indicators which allow for a comparison of different line plans, timetables and vehicle schedules with respect to their vulnerability to delays. The results of robustness tests shall provide planners with a detailed account of the strengths and weaknesses of their public transport plans with respect to three dimensions: operating costs, service quality, and robustness. We provide an extension of the preliminary robustness tests introduced in ATMOS 2017 [12]. In that work, we considered three different types of isolated delay scenarios: delays of individual vehicles, delays caused by slow-downs on segments, and delays caused by blockings at stops. In this paper, we complement these robustness tests to cover specific characteristics of a public transport in a more realistic way. The overall robustness test framework has been designed to model public transport in a fairly realistic way. Important features and enhancements include the following:

- Passengers choose routes according to a generic cost model for perceived travel times.
- We consider vehicle schedules in two ways: as base for computing operating costs and for the propagation of delays over consecutive trips of the same vehicle.
- We apply a more realistic model by considering limited vehicle capacities. In our simulations, passengers are forbidden to enter fully occupied vehicles.
- In practice, waiting and disposition strategies are used to reduce passenger delays. Typical strategies can be studied within our framework.
- We compare timetables that have been optimized with different strategies to increase robustness by inserting time supplements (buffer times).
- We apply random delays in simulations which are based on historical observations.

We aim at answering the following questions:

- Do we observe a trade-off between robustness and travel times on the basis of timetables which are optimized with respect to travel time for the respective line plan?
- Can we detect shortcomings with respect to robustness of specific transport plans?

- What is the benefit of time supplements? Where should supplements be inserted and how large should they be to get a reasonable compromise between robustness and travel time?
- How different are the outcomes of the robustness tests? If we rank all solutions decreasingly by their robustness, do we obtain consistent rankings for the proposed robustness tests?

Related work. Parbo et al. [20] provide a recent literature review on passenger perspectives in railway timetabling. They argue that there is a gap between passengers' perception of railway operations and the way timetables are designed. In particular, they observe that a discrepancy exists between how rail operations are planned with the main focus being on the trains and how passengers actually perceive and respond to railway performances. Punctuality of public transport is of high importance for passengers. Therefore, a plenitude of methods exist to quantify the deviation of the realized schedule from the planned schedule. Reports often provide the percentage of arrivals on time, where *being on time* is defined as to arrive not later than within a given margin (e.g., 5 or 15 minutes for long-distance trains) of the planned arrival time. In a Dagstuhl seminar in 2016 (`http://www.dagstuhl.de/16171`), Dennis Huisman coined the phrase "passenger punctuality 2.0" for measuring the (weighted) total passenger delay at the destination for all passengers. The latter definition has been used by [24, 15, 9, 8, 10] and others. Less sophisticated indicators include the mere number of delayed departure and arrival events [7]. Alternatively, Acuna-Agost et al. [1] propose to count every time unit of delay at every planned stop and at the last stop. Robust planning has been studied intensively, see Lusby et al. [18] for a very recent survey, and Josyula and Törnquist Krasemann [16] for a review of passenger-oriented railway rescheduling strategies. In robust timetabling it is desirable from an operational point of view that a timetable can absorb delays and recover quickly (thus avoiding penalties for the operator). To achieve this, inserting time supplements may help to reduce the effect of disturbances, but may have a negative effect on average travel times. Not only the total amount of supplements times, but also their distribution along the line routes is important. These aspects have been studied intensively in operations research. For example, Kroon et al. [17] use stochastic optimization to allocate time supplements to make the timetable maximally robust against stochastic disturbances. They use the expected weighted delay of the trains as indicator. To increase delay tolerance, Amberg et al. [2, 3] consider the redistribution and insertion of supplement times in integrated vehicle and crew scheduling for public bus transport. Using mixed integer linear programming, Sels et al. [28] improve punctuality for passenger trains in Belgium by minimizing the total passenger travel time as expected in practice. Bešinović et. al. [5] optimized the trade-off between travel times and maximal robustness using an integer linear programming formulation which includes a measure for delay recovery computed by an integrated delay propagation model. In these works, the line network is usually already fixed. Robustness of timetables was empirically investigated (with respect to different robustness concepts) in [14], robustness of lines has been studied in [13] and [27]. A general survey of line planning in public transport can be found in [25]. A recent integrated approach combines line planning, timetabling, and vehicle scheduling, but without considering robustness [26].

Overview. In Section 2, we present our algorithmic framework and motivate different robustness tests for public transport plans. To evaluate them, we explore in Section 3 four transport plans for German long-distance trains. Moreover, we consider 60 different transport plans for an artificial grid network. All test instances mainly differ in the strategy by which supplement times are incorporated into the timetable. Finally, we summarize and conclude with future work.

2 Simulation framework and robustness tests

In this section, we first sketch the basics of our simulation framework. Afterwards, we introduce several robustness tests.

2.1 Simulation framework

Basic definitions. In this paper we use the following definitions to describe the movements of vehicles and passengers:

- *Service run*: Movement of a vehicle between the start and terminal node of a line route. A service run has a scheduled arrival and departure time at every stop.
- *Route*: Movement of a passenger between an origin and a destination stop. A route (a.k.a. itinerary or connection) describes a passenger trip consisting of a sequence of trip legs, i. e. transfer-free segments of a trip. Every trip leg uses one particular service run and has a defined departure and arrival time. A route requires transfers between trip legs.
- *Routing*: A process to determine the route of a passenger. Rerouting is necessary, if the planned route of a passenger fails. This happens if service runs are late or overloaded.

Event-activity network. To represent a public transport timetable with its corresponding vehicle schedule, we use a so-called *event-activity network (EAN)* $N = (V, A)$, i.e. a directed acyclic graph with vertex set V and arc set A. The vertices of the network correspond to the set of all arrival and departure events of the given timetable. Each event is equipped with several attributes: its type (arrival or departure), the id of the corresponding service, the stop, and several timestamps. In this context we distinguish between the planned event time according to schedule, and the realized time after the event has occurred. In an online scenario, one also has to consider the estimated event times with respect to the current delay scenario. Arcs of the network model order relations between events. We distinguish between different types of arcs ("activities"):

- driving arcs, modeling the driving of a specific vehicle from one stop to its very next stop,
- dwelling arcs, modeling a vehicle standing at a platform and allowing passengers to enter or leave it,
- transfer arcs, modeling the possibility for passengers to change from one vehicle to another, and
- vehicle circulation arcs, modeling the usage of the same physical vehicle in subsequent services.

Every arc (activity) has an attribute specifying its minimum duration. For driving arcs we thereby model the catch-up potential between two stops under optimal conditions. For dwelling arcs the minimum duration corresponds to the minimum time needed for boarding and deboarding. For transfer arcs, the minimum duration models the time which a passenger will need for the transfer. For vehicle circulation arcs, the minimum duration specifies the time needed between two services.

Disposition policies. For all non-direct travelers, the effect of some delay on their arrival time at the destination depends on the chosen delay management policy of the responsible transport operator. Waiting time rules specify how long a vehicle will wait at most for a delayed feeder service. Such rules may depend on the involved lines, the time when to be applied, and other criteria. Our basic framework follows in spirit those of PANDA [22], a tool originally developed for optimized passenger-friendly train disposition. It can be instantiated in a flexible way with almost arbitrary fixed waiting time strategies, in particular with the extreme cases of NO-WAIT and ALWAYS-WAIT.

Delay propagation. Delay scenarios are specified by a set of source delays. Given some source delay, the delay is propagated from the current event to forthcoming events of the same service, and possibly to subsequent services of the same vehicle. Depending on the disposition policy (waiting strategies), it may also influence services provided by other vehicles. Delay propagation due to capacity restrictions of the infrastructure is not considered. New timestamps for events are derived through a propagation in breadth-first search order in the event activity network [19].

Vehicle capacities. An important optional feature of our framework is to incorporate vehicle capacities into our simulations. Every vehicle has a maximum capacity for transporting passengers. When capacity limitations are applied, passengers can only board if the given capacity is not exceeded. Otherwise, they have to wait for the next service or to look for an alternative route. If too many passengers compete for the remaining capacity, we choose randomly who can enter the vehicle.

Passenger routing and rerouting. In our framework, we assume that passengers prefer shortest routes with few transfers. We use a generic cost function to evaluate travel time on routes which penalizes every transfer by an equivalent of five minutes of extra travel time in the grid network and ten minutes in the long-distance train network, subsequently referred to as *perceived travel time*. In our model passengers behave always rational and have access to full information about all current delays. That means, passengers can send route queries to an online server, but it seems reasonable to assume that they check their route only in certain specific situations:

1. Whenever passengers wait at a stop and the next vehicle they intend to board is late, they also check for a better connection and take a new route if this choice reduces their travel time compared to the delayed previous choice.
2. Likewise, if passengers sit in a vehicle and notice that it has caught some delay, they will actively check the feasibility of the current route and switch to a new route if necessary.
3. While we assume that a central server has full information about current delays, the passenger load in each vehicle is unknown. Therefore, it may happen that passengers choose a route which later turns out to be infeasible due to limited capacities. In such cases, passengers notice that a particular vehicle is full and cannot be used only when they try to board it. As a consequence, they also have to adapt their route.

Table 1 summarize the different possibilities which require routing requests. Our framework is flexible in the sense that some rerouting actions can be switched off (last column of Table 1).

In some cases passengers miss the last connection of a day. Such passengers are treated separately. They either have to use a different means of transportation (for example, a taxi) or they have to spend a night in a hotel in case of a long-distance journey. We penalize such cases with a fixed delay of four hours. We assume for simplicity that passengers choose alternative routes again with the same principle (minimum perceived travel time). Such routes can efficiently be computed by some variant of Dijkstra's algorithm, see [4] for a recent survey on fast approaches. For large networks with many origin-destination pairs, a sufficiently fast method is needed to achieve reasonable simulation times.

Composition of framework. The framework consists of several modules which can be instantiated in a flexible way. Figure 1 provides an overview of the overall robustness test framework, its modules and interfaces.

Table 1 Classification of cases for passenger rerouting.

Situation	Point in time for next rerouting	Rerouting mandatory?
first vehicle on route is late	planned time for departure	no
next vehicle to board is late	current time of arrival event before transfer	no
future transfer is invalid	current time of arrival node before next transfer	no
current transfer invalid	current time of arrival node before transfer	yes
current capacity while boarding invalid	current time of departure event	yes

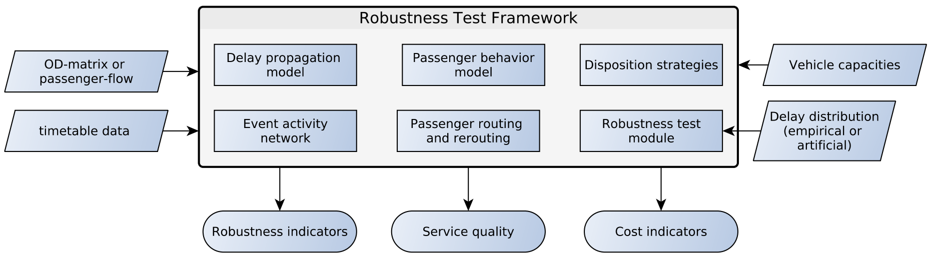


Figure 1 Modules of the robustness framework.

2.2 Robustness tests

As discussed in the introduction, robustness of transport plans can be measured in many ways. In the following, we focus on small to moderate delays and take a passenger-oriented view. That means, we want to quantify the effect of delays and disturbances on passengers. We propose the following four robustness tests.

Robustness test RT-1: Delays of single service runs (initial vehicle delay). The first robustness test considers the effect of the delay of a single service run in isolation. We evaluate many distinct scenarios, one for each service run of the given timetable. Every service run is delayed by x_1 minutes at its first stop. Such a delay may occur due to technical problems of some specific vehicle or due to the late arrival of some feeder vehicle causing a departing vehicle to wait for changing passengers.

Robustness test RT-2: Slow-down of single network sections (track delay). A second robustness test models scenarios where some network section invokes a certain delay. For example, temporary speed restrictions may occur because of construction work or for safety reasons. Whenever a service run passes this section, it catches a delay of x_2 minutes. Optionally, this test can be refined by either considering bidirectional or unidirectional delays on the network section.

Robustness test RT-3: Temporary blocking of single stop (stop disruption). We model the temporary blocking of a whole stop. Such a disturbance has a starting point t_{start} and a duration x_3. During the blocking phase, we assume that vehicles may still enter the stop but no vehicle can leave it. For simplicity, we further assume that the capacity to hold all vehicles at the stop is sufficient. For long-term blockings, a more detailed model would be required. When the blocking phase is over, vehicles restart from this stop one after another in the order of their scheduled departure with a constant headway of *headway* minutes.

Robustness test RT-4: Empirical delay distributions. We use data from past observations for source delay distributions on network sections and start delays of trips. For the initial departure event of each service and for all intermediate arrival events, we draw random independent source delays from these distributions. These source delays are then propagated through the network. Since the resulting delay scenarios are randomly drawn, the tests have to be repeated many times. From several pretests we learned that 50 repetitions are sufficient to yield stable means of our test indicators in practice.

3 Experiments

In order to evaluate the proposed robustness tests, we perform a number of experiments. In this section, we will first describe the chosen test instances and how they have been generated. Then, we discuss the choice of parameters used within the robustness tests. Then, in the main part we present the results. As indicators for robustness we use the total delay and the fraction of affected passengers. By total delay we refer to the sum of delays at their destinations experienced by all passengers across all separate parts of one robustness test.

3.1 Test data and parameters

In this paper, we use two types of instances: (1) instances based on a simplified version of the German long-distance (high-speed) train network, and (2) variations of artificial instances on a grid network as proposed in [11] for studying different planning strategies.

Construction of transport plans. Various public transport plans have been created using the LinTim-framework [13, 23]. For choosing the lines and their frequencies, a cost-based approach was chosen, see [25]. This approach starts with a line pool and assigns a frequency to each line in the pool. Lines with a frequency of zero are not chosen. The objective is to minimize the costs, i.e., to cover the demand, but using as few lines and as low frequencies as possible. From the resulting lines, we construct an event-activity network in which the timetabling step is performed. To this end, every driving, dwelling and transfer activity receives a minimum duration to which we add supplement times. To increase robustness we require for specific activities a certain minimum supplement (details below). For computing the timetable we used the fast MATCH approach introduced in [21]. After rolling out the periodic timetable for the day, a vehicle schedule is computed using a basic IP-based approach, see [6], to minimize the overall vehicle scheduling costs.

Long-distance train instances. We study four instances based on a simplified version of the German long-distance train network containing major stations in Germany and stations of neighbouring countries (Figure 2). For this network we used an artificial demand containing 380k passengers. This matches the average number of passengers travelling on long-distance trains in Germany. The four instances are based on different minimum time supplements, specified for driving sections or for dwelling activities at busy (i.e. highly used) stops:

- (A) no minimum supplement times (“no_buffer”)
- (B) supplement time is at least 3 minutes at busy stops (“3_min_busy_stops”)

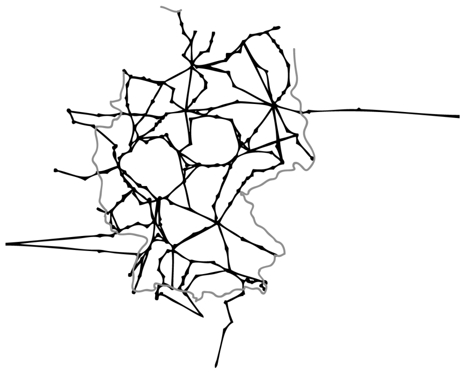
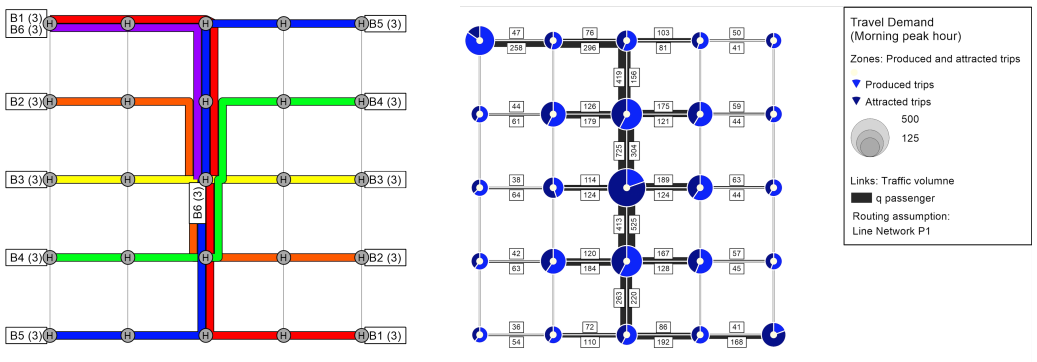


Figure 2 Long-distance train network used in this paper.



(a) Line network. **(b)** Travel demand.

Figure 3 Grid instances: Common line network of all instances with line frequencies per hour (in brackets) and the passengers' travel demand.

- (C) supplement time is at least 5% of driving time ("5_percent_drive_buffer")
- (D) supplement time is at least 10% of driving time ("10_percent_drive_buffer")

Note that every periodic timetable has some intrinsic slack (due to the periodicity), i.e., usually there will be some arcs with a larger slack than the required time supplement.

Grid instances. In [12] many different line plans were tested concerning their robustness to similar tests. Based on a comparatively robust line plan (Figure 3a) we created 60 schedules that implement different strategies to distribute supplement times to further improve robustness. Figure 3b shows the travel demand (in morning peak hours) common to all instances. The demand is defined for every pair of origin and destination and every hour of the day containing 19500 passengers in total. Each vehicle is assumed to operate with a capacity of 65 passengers. All instances were created in an effort to minimize perceived travel time, number of transfers and operating costs.

Recall that we aim at studying to which extent we can increase robustness by inserting additional time supplements. With this goal in mind, we created ten different classes of instances described in Table 2. For every class we created six instances with different time supplements concerning their circulation time supplement (commonly also referred to as free layover time or supplement on turn-around). Namely, these six instances of each class

Table 2 Time supplement classes for grid network.

Class	constraints for driving and dwelling times	undisturbed mean travel time
A	no supplements	20.08 min
B	3 minutes supplements at the 5% most frequently used stops	20.19 min
C	1 minute supplement for driving sections of central north-south axis	20.31 min
D	2 minutes supplement for driving sections of central north-south axis	20.54 min
E	3 minutes supplement for driving sections of central north-south axis	20.76 min
F	3 minutes supplements at 10% most frequently used stops	20.84 min
G	3 minutes supplements at 15% most frequently used stops	21.06 min
H	3 minutes supplements at the 5% most frequently used driving sections	22.61 min
I	3 minutes supplements at the 10% most frequently used driving sections	23.48 min
J	3 minutes supplements at the 15% most frequently used driving sections	23.85 min

have at least circulation time supplements of 0,1,3,5,7 and 9 minutes. The circulation time supplement influences the vehicle schedule. The larger this supplement, the more vehicles are required. The timetable, however, remains unchanged for each instance of the same class. In Table 2, the instance classes are ordered increasingly with respect of the mean undisturbed travel times of passengers.

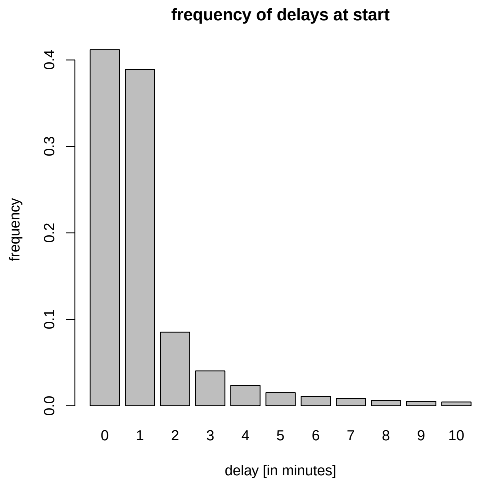
The operating costs of a transport plan are calculated in a simple model as the sum of two components as follows. For each used vehicle we consider the amount of travelled distance in kilometers (including empty trips) and apply a cost factor of 1.5€$/km$. For each used vehicle, the operating time (regular and empty trips) is multiplied with a cost factor of 50€$/h$ which includes personal costs, depreciation and maintenance of the vehicle.

3.2 Setup of the robustness tests

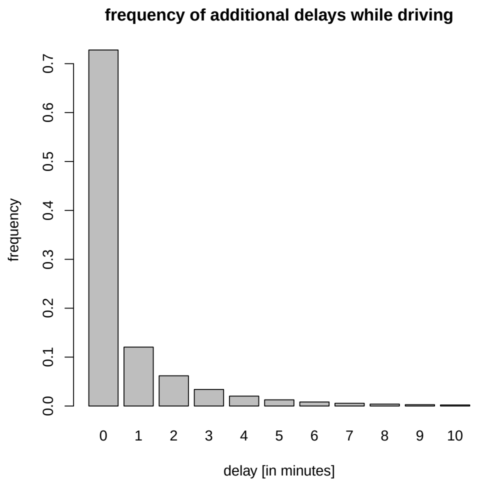
For each robustness test, a certain range of parameters has to be chosen. We decided to vary the parameters in the following ranges.

- Test RT-1: $x_1 = 1..18$ min initial vehicle delay
- Test RT-2: $x_2 = 1..10$ min delay for crossing disturbed edge
- Test RT-3: $x_3 = 10..20$ min blocking time for disturbed station
- Test RT-4 applies an empirical delay distribution. We use observed delay data of German long-distance trains from a dataset of 2016-2017 containing over 28 million events for ICE and IC trains. Based on these data, we derive empirical, discrete delay distributions for two types of source delays. The first type is the starting delay of a vehicle at its first departure of a trip. The second type of delay is the additional delay of a vehicle on any driving edge (see Figures 4a and 4b). Both delay distributions are truncated to the interval between 0 and 10 minutes of delay.

Let us briefly discuss the rationale behind our parameter choices for the experiments with the grid instances. For the first test RT-1, an initial delay of more than 20 minutes would result in a simultaneous departure of the delayed vehicle and the next trip of the same line. Therefore, we choose 18 minutes as maximal delay. For experiments with the second robustness test RT-2 we choose 10 minutes as a maximum delay for a similar reason. Even larger delays would significantly decrease the number of service runs per day (unless there is enough catch-up potential). For experiments with robustness test RT-3 we choose upper



(a) Start delays.



(b) Delays on driving sections.

Figure 4 Empirical delay distributions for long-distance trains.

limits of a disruption to be 20 minutes. If the duration of the disruption is too high and every vehicle in the network passing a central station is hit by it, the propagation of delays would be too large and thus making an evaluation of resulting effects on passengers pointless.

3.3 Experiments with long-distance train network

Next we present the results of applying our robustness tests to the four instances of long-distance train networks. In a first experiment, we are interested in the impact of different delay parameters in the robustness tests RT-1 to RT-3. For robustness test RT-1, Figure 5 (left) shows the total delay of all passengers depending on different delay parameters x_1. The total delay seems to increase almost linearly with the size of the vehicle delay parameter for all instances. When we look at the results of RT-1, it seems as if there are clear differences between the four instances. Indeed, in terms of total delay instance (A) without time supplements is always worst. The slowest increase rate occurs for instance (B) with supplement times for busy stations. The ranking of the other instances changes when increasing the delay parameter. However, it is very important to note that the total delay is fairly small in absolute terms for all instances. Thus, all instances are quite robust against delays of single vehicles. Robustness tests RT-2 and RT-3 show a linear dependence on the delay parameters and yield the same ranking for all tested parameters (Figures 5 and 6). The relative gap between the instance with no time supplement and the three other instances in RT-2 (Figure 5, right) is significantly larger than in the two other experiments, and the gap is increasing with the size of the delay. For the robustness test RT-4 based on empirical delay distributions, the resulting delays for passenger routes are quite significant. We observe an average delay of 12 minutes per passenger in our instance (A) having no time supplements, about 8 minutes for instance (B), about 6 minutes for instance (C) and only 4.5 minutes for instance (D). Figure 6 (right part) shows that instance (C) has the best tradeoff between mean delay per passenger and planned travel time.

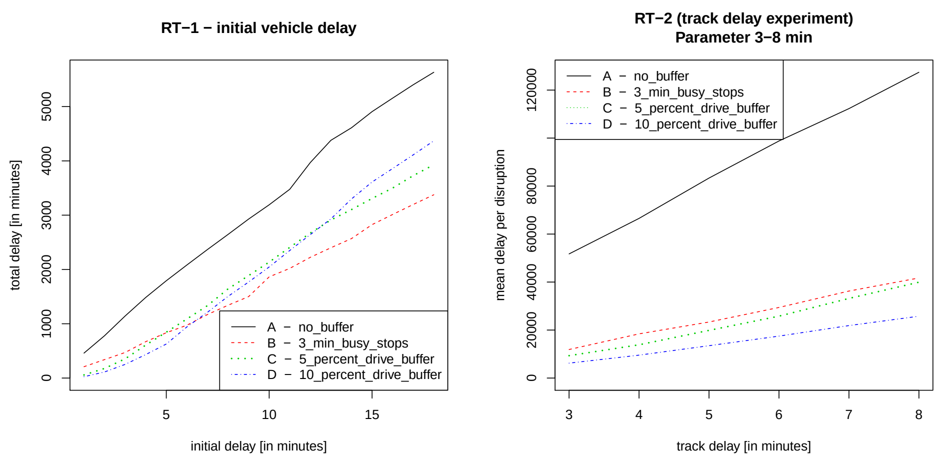


Figure 5 Robustness tests RT-1 and RT2 on long-distance train network.

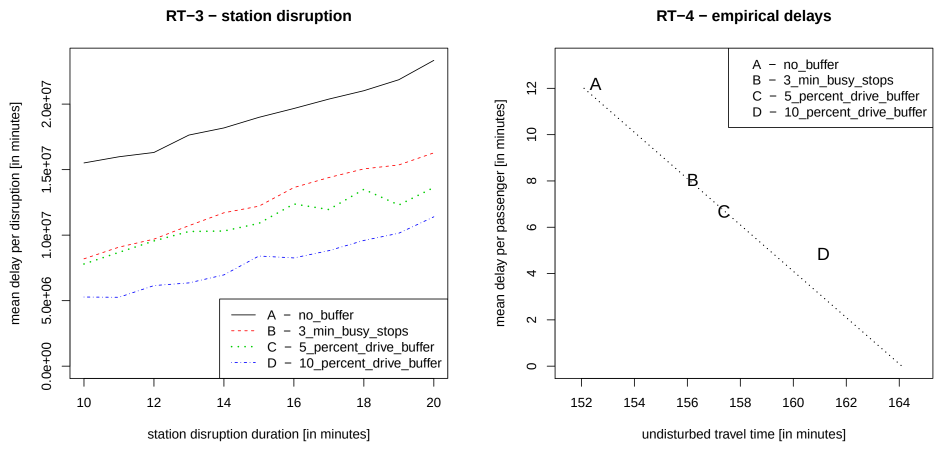


Figure 6 Robustness test RT-3 and RT-4 on long-distance train network. For RT-4, the stroked line is a visual guide for a linear tradeoff between undisturbed travel time and expected travel time.

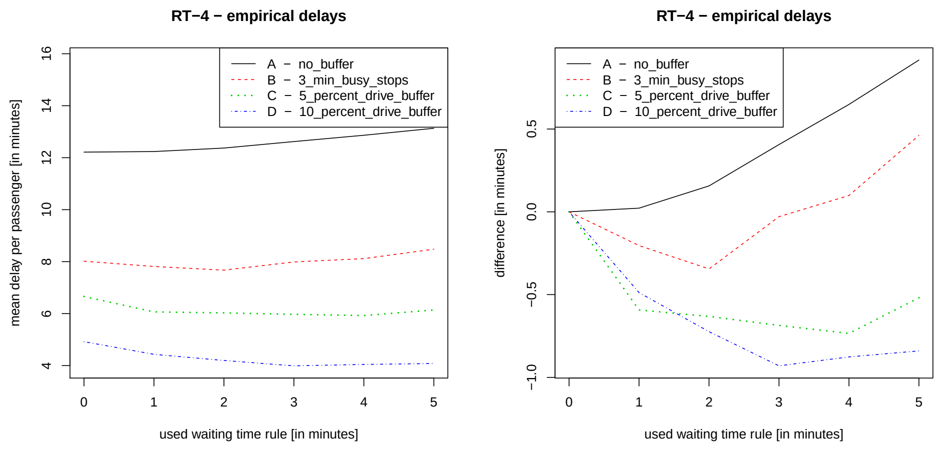


Figure 7 Robustness test RT-4 on long-distance train network with different fixed waiting time rules.

Disposition strategies. The next experiment compares fixed waiting strategies for the timetables of the long-distance train network. By a *fixed waiting time strategy* we mean the following. In case of a delayed feeder train, the departing trains waits for up to x minutes if this helps to save a planned transfer for at least one passenger. In our experiment, we compare different strategies with $x \in \{0, 1, 2, 3, 4, 5\}$ where $x = 0$ means NO-WAIT.
Figure 7 shows several interesting findings:

- The mean delay per passenger differs significantly for instances (A)-(D): the smallest delay occurs for timetable (D) with 10% time supplement on driving sections, second best is the timetable (C) with 5% time supplement on driving sections, followed by timetable (B) with a 3-minute-supplement on busy stations. Not surprisingly, the instance (A) without extra supplement times performs worst.
- The best performing fixed waiting strategy depends on the timetable. The larger the time supplements are, the longer one can afford to wait for delayed trains.
- Using a fixed waiting time strategy the mean delay per passenger can be reduced by up to about 25% in comparison with NO-WAIT strategy.

3.4 Experiments with grid instances

Comparison of supplement time strategies. For each of the robustness tests RT1-RT3, we obtain a ranking of the 60 grid instances with respect to the observed total delay (with rank 1 being the best). An obvious question is to ask whether we do observe different or consistent rankings for different parameters and robustness tests. If we compare the rankings of the robustness shown in Figure 8, one can see that the instances where the time supplements are placed on the driving arcs have best ranks among all classes independent from the circulation time supplement. The class of instances where the time supplements are placed at every stop rank second and the instances where the supplements are placed at the most frequently used stops rank third. We expect that circulation time supplements become more important for delay scenarios with larger delay parameters. This is confirmed in Figure 8 where rankings of instances with 9 minutes of required circulation time supplements clearly outperform those without such a required supplement. Instances with a circulation time supplement of 9 minutes in most cases improve their ranking with increasing disturbances, while instances with no circulation time supplement tend to fall off in their ranking.

Trade-off between travel time and robustness. We also have to compare the average time it takes passengers to reach their destination on an undisturbed day of traffic with the results from the robustness test. We can see this trade-off for a fixed robustness test and parameter on Figure 9 (left). Although the instances (H)-(J) have superior robustness, their tradeoff between robustness and perceived travel-time is worse than the tradeoff between the other classes of instances. In Figure 9 (right) one can see, that the costs of these solutions are significantly higher than for instances (A)-(F). These evaluations were made for one specific set of delay parameters, but tests with other sets of parameters yield similar results.

Choosing turn-around time supplements. One other question we would like to answer is: How to choose a good turn-around time supplement? And what is the trade-off between the corresponding increased costs and reduced delays? Figure 9 can provide several insights to this question for a fixed delay parameter of $x_2 = 5$ minutes in RT-2. For the relatively cheap classes of schedules (A) to (F), it seems worth to invest 3 to 5 minutes of turn-around supplements. Schedules within the class (G) to (J), however, do not require such turn-around supplements. These schedules can already make use of other time supplements. The magnitude of regular

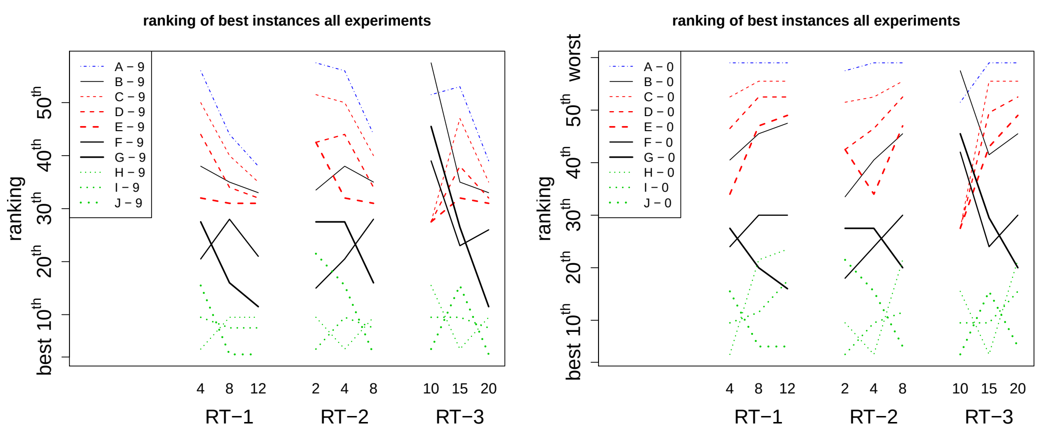


Figure 8 Comparison of timetables with respect to robustness ranking for different tests. Instances are denoted by class (A-J) and required circulation time supplement (0 or 9 minutes).

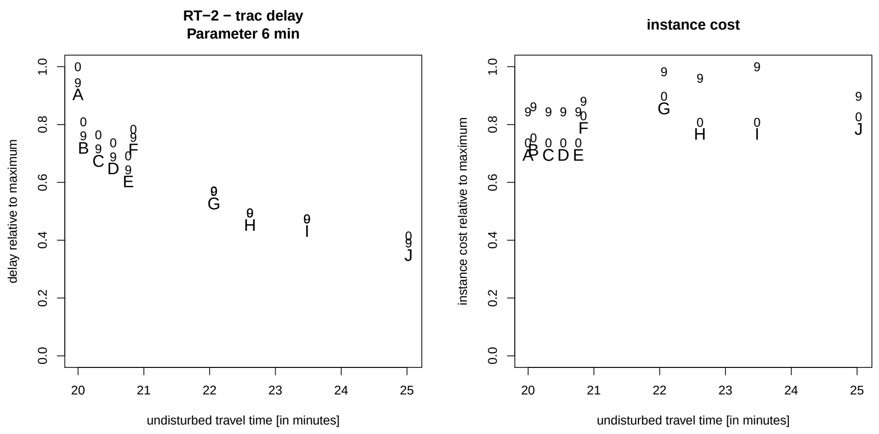


Figure 9 Instances are denoted by class (A-J) and the required minimum circulation time supplement of 0 or 9 minutes. Left: We compare the robustness trade-off for all instance classes of the grid instances with respect to robustness test RT-2 with a slowdown parameter of $x_2 = 5$ minutes. The y-axis shows the observed delay relative to the instance with maximum delay. Right: We show the tradeoff between average undisturbed travel time (x-axis) and relative operating costs.

disruption determines the need for a turn-around time supplement. However, for small to medium disruptions the trade-off between benefit and cost for introduction of time supplements up to 3 minutes was always good for instances (A) to (F).

Three-dimensional trade-offs. We now face the challenge of evaluating transport plans to several factors simultaneously and to show their mutual trade-offs. We consider

- C – operational cost,
- T – average travel time of passengers, and
- R – robustness measured as total delay for the passengers in our robustness tests. As combined robustness measure we take the average performance with respect to robustness tests RT1-RT3 with the delay parameters set to a medium value (RT-1 with $x_1 = 8$ minutes, RT-2 with $x_2 = 5$ minutes and RT-3 with $x_3 = 15$ minutes).

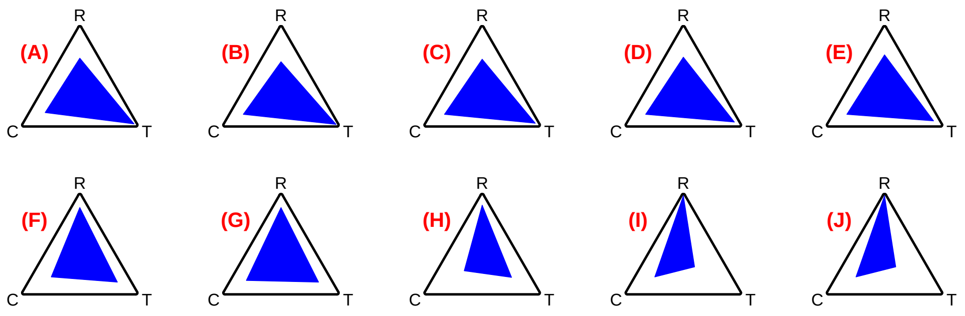


Figure 10 Comparison of operational cost C, average travel-time T, and the combined robustness R with respect to robustness tests RT1-RT3 for instances (A)-(J).

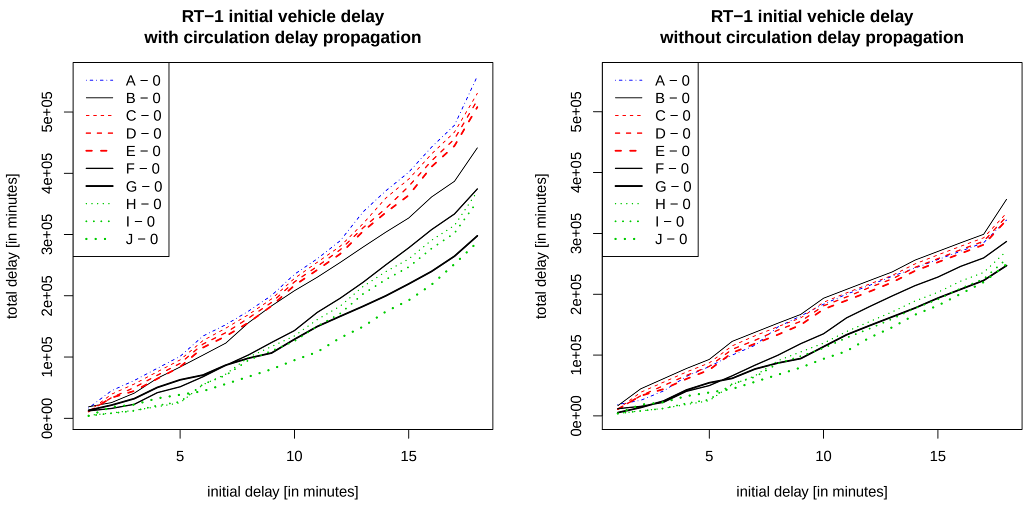


Figure 11 Comparison of robustness test RT-1 with and without vehicle circulations arcs for the grid instances.

In order to visualize the trade-off between different features we use a special form of a radar or spider chart (Figure 10). The center point of one triangle corresponds to the worst solution, while the corners represent the best value of one instance in the respective attribute. In the following comparison we concentrate on only those instances with the same circulation time supplement of 9 minutes. Instances (D), (E), (F) have a large triangle area, which can be interpreted as being good solutions over all criteria. Instance (I), however, seems to be the worst instance.

Impact of vehicle circulations. In another experiment, we studied how important it is to consider vehicle circulations. Therefore, we conducted tests with activated and deactivated vehicle circulation arcs in the EAN. As one example, we show in Figure 11 results of robustness test RT-1 with the grid instances (using zero minutes of circulation time supplement). The measured total delay is clearly significantly larger for the EAN with circulation arcs enabled than for without. It can be larger by up to a factor of 1.5. Moreover, the difference between the instance classes (A)-(J) increases with the initial delay parameter of this test. We conclude that vehicle circulation arcs should be considered.

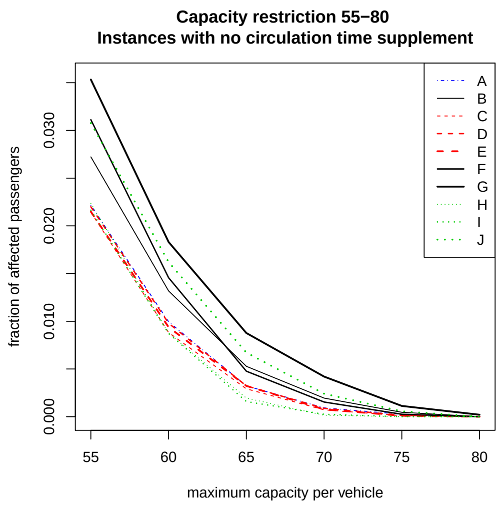


Figure 12 Experiments with vehicle capacities in the range 55-80.

Impact of vehicle capacities. One important aspect covered by our model is to consider the impact of vehicle capacities in public transport. As mentioned earlier we assume hard capacities of at most 65 passengers per vehicle in our simulations. This number was used in the creation of the instances using LinTim. In the following experiment we study the dependence of experienced delays subject to different capacities. Would smaller vehicles or more passengers have an effect on our results? To answer this question we simulated undisturbed days of traffic with constant demand, and varied the maximum capacity of all vehicles. We consider the range of vehicle capacities between 55 and 80.

Figure 12 displays the fraction of passengers which for a given vehicle capacity are affected by congestion in a way that they have to adapt their planned route as they cannot board a full vehicle. We observe that for all considered timetables our assumed vehicle capacity of 65 passengers per vehicle leads to less than 1% of all passengers being affected by congestion. Limiting the capacity further or increasing the number of passengers would decrease the robustness of the instance due to vehicle capacity constraints.

4 Summary

We have presented a general and flexible framework for performing and evaluating robustness tests with varying parameters. Extending earlier work in [12], our refined model now includes circulation arcs, vehicle capacities, a generic cost function for choosing passenger routes, and disposition strategies. Robustness tests RT1-RT3 provide public transport planners with a tool for comparing timetables without empirical delay data. By varying the delay parameters of these tests, it is possible to study the dependence of the robustness on the severeness of the delay scenario. When empirical delay data is available, the robustness test RT-4 provides realistic expected average delays. Exploring a set of instances applying different strategies for distributing time supplements, we have been able to analyze strengths and weaknesses of all instances. An interesting use case of our framework is to optimize waiting time strategies.

In future work we would like to further improve our simulations by including more information about the network infrastructure and applying sophisticated models for passenger behaviour in case of disruptions. Another extension may consider soft vehicle capacity constraints where passenger satisfaction is degraded when vehicles become too crowded.

References

1 Rodrigo Acuna-Agost, Philippe Michelon, Dominique Feillet, and Serigne Gueye. A MIP-based local search method for the railway rescheduling problem. *Networks*, 57(1):69–86, 2011.

2 Bastian Amberg, Boris Amberg, and Natalia Kliewer. Increasing delay-tolerance of vehicle and crew schedules in public transport by sequential, partial-integrated and integrated approaches. *Procedia - Social and Behavioral Sciences*, 20:292–301, 2011.

3 Bastian Amberg, Boris Amberg, and Natalia Kliewer. Robust efficiency in urban public transportation: Minimizing delay propagation in cost-efficient bus and driver schedules. *Transportation Science*, 2018. doi:10.1287/trsc.2017.0757.

4 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016. doi:10.1007/978-3-319-49487-6_2.

5 Nikola Bešinović, Rob M.P. Goverde, Egidio Quaglietta, and Roberto Roberti. An integrated micro-macro approach to robust railway timetabling. *Transportation Research Part B: Methodological*, 87:14–32, 2016.

6 Stefan Bunte and Natalia Kliewer. An overview on vehicle scheduling models. *Public Transport*, 1(4):299–317, 2009.

7 Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Recoverable robust timetabling for single delay: Complexity and polynomial algorithms for special cases. *Journal of Combinatorial Optimization*, 18(3):229, Aug 2009.

8 Twan Dollevoet and Dennis Huisman. Fast heuristics for delay management with passenger rerouting. *Public Transport*, 6(1-2):67–84, 2014.

9 Twan Dollevoet, Dennis Huisman, Marie Schmidt, and Anita Schöbel. Delay management with rerouting of passengers. *Transportation Science*, 46(1):74–89, 2012.

10 Twan Dollevoet, Dennis Huisman, Marie Schmidt, and Anita Schöbel. Delay propagation and delay management in transportation networks. In Ralf Borndörfer, Torsten Klug, Leonardo Lamorgese, Carlo Mannino, Markus Reuther, and Thomas Schlechte, editors, *Handbook of Optimization in the Railway Industry*, pages 285–317. Springer International Publishing, 2018.

11 Markus Friedrich, Maximilian Hartl, Alexander Schiewe, and Anita Schöbel. Angebotsplanung im öffentlichen Verkehr - planerische und algorithmische Lösungen. In *Heureka'17*, 2017.

12 Markus Friedrich, Matthias Müller-Hannemann, Ralf Rückert, Alexander Schiewe, and Anita Schöbel. Robustness Tests for Public Transport Planning. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 6:1–6:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

13 Marc Goerigk, Michael Schachtebeck, and Anita Schöbel. Evaluating line concepts using travel times and robustness: Simulations with the LinTim toolbox. *Public Transport*, 5:267–284, 2013.

14 Marc Goerigk and Anita Schöbel. An Empirical Analysis of Robustness Concepts for Timetabling. In Thomas Erlebach and Marco Lübbecke, editors, *10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10)*, volume 14 of *OpenAccess Series in Informatics (OASIcs)*, pages 100–113, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2010.100.

15 Géraldine Heilporn, Luigi De Giovanni, and Martine Labbé. Optimization models for the single delay management problem in public transportation. *European Journal of Operational Research*, 189(3):762–774, 2008.

16 Sai Prashanth Josyula and Johanna Törnquist Krasemann. Passenger-oriented railway traffic re-scheduling: A review of alternative strategies utilizing passenger flow data. In *7th International Conference on Railway Operations Modelling and Analysis, Lille*, 2017.

17 Leo Kroon, Gábor Maróti, Mathijn Retel Helmrich, Michiel Vromans, and Rommert Dekker. Stochastic improvement of cyclic railway timetables. *Transportation Research Part B: Methodological*, 42(6):553–570, 2008.

18 Richard M. Lusby, Jesper Larsen, and Simon Bull. A survey on robustness in railway planning. *European Journal of Operational Research*, 266(1):1–15, 2018.

19 Matthias Müller-Hannemann and Mathias Schnee. Efficient timetable information in the presence of delays. In R. Ahuja, R.-H. Möhring, and C. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 249–272. Springer, 2009.

20 Jens Parbo, Otto Anker Nielsen, and Carlo Giacomo Prato. Passenger perspectives in railway timetabling: A literature review. *Transport Reviews*, 36:500–526, 2016.

21 Julius Pätzold and Anita Schöbel. A Matching Approach for Periodic Timetabling. In Marc Goerigk and Renato Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 1–15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

22 Ralf Rückert, Martin Lemnian, Christoph Blendinger, Steffen Rechner, and Matthias Müller-Hannemann. PANDA: a software tool for improved train dispatching with focus on passenger flows. *Public Transport*, 9(1):307–324, 2017.

23 Alexander Schiewe, Sebastian Albert, Julius Pätzold, Philine Schiewe, Anita Schöbel, and Jochen Schulz. Lintim: An integrated environment for mathematical public transport optimization. documentation. Technical Report 2018-08, Preprint-Reihe, Institut für Numerische und Angewandte Mathematik, Georg-August Universität Göttingen, 2018. homepage: http://lintim.math.uni-goettingen.de/.

24 Anita Schöbel. A model for the delay management problem based on mixed-integer programming. *Electronic Notes in Theoretical Computer Science*, 50(1), 2001.

25 Anita Schöbel. Line planning in public transportation: models and methods. *OR Spectrum*, 34(3):491–510, Jul 2012.

26 Anita Schöbel. An eigenmodel for iterative line planning, timetabling and vehicle scheduling in public transportation. *Transportation Research Part C: Emerging Technologies*, 74:348–365, 2017.

27 Anita Schöbel and Silvia Schwarze. Finding delay-resistant line concepts using a game-theoretic approach. *Netnomics*, 14:95–117, 2013.

28 Peter Sels, Thijs Dewilde, Dirk Cattrysse, and Pieter Vansteenwegen. Reducing the passenger travel time in practice by the automated construction of a robust railway timetable. *Transportation Research Part B: Methodological*, 84:124–156, 2016.

Fast Robust Shortest Path Computations

Christoph Hansknecht
Institute for Mathematical Optimization, Technical University Braunschweig, Germany
c.hansknecht@tu-braunschweig.de

Alexander Richter
Institute for Mathematical Optimization, Technical University Braunschweig, Germany
a.richter@tu-braunschweig.de

Sebastian Stiller
Institute for Mathematical Optimization, Technical University Braunschweig, Germany
sebastian.stiller@tu-braunschweig.de

Abstract

We develop a fast method to compute an optimal robust shortest path in large networks like road networks, a fundamental problem in traffic and logistics under uncertainty.

In the robust shortest path problem we are given an s-t-graph $D(V, A)$ and for each arc a nominal length $c(a)$ and a maximal increase $d(a)$ of its length. We consider all scenarios in which for the increased lengths $c(a) + \bar{d}(a)$ we have $\bar{d}(a) \leq d(a)$ and $\sum_{a \in A} \frac{\bar{d}(a)}{d(a)} \leq \Gamma$. Each path is measured by the length in its worst-case scenario. A classic result [6] minimizes this path length by solving $(|A|+1)$-many shortest path problems. Easily, $(|A|+1)$ can be replaced by $|\Theta|$, where Θ is the set of all different values $d(a)$ and 0. Still, the approach remains impractical for large graphs.

Using the monotonicity of a part of the objective we devise a Divide and Conquer method to evaluate significantly fewer values of Θ. This methods generalizes to binary linear robust problems. Specifically for shortest paths we derive a lower bound to speed-up the Divide and Conquer of Θ. The bound is based on carefully using previous shortest path computations. We combine the approach with non-preprocessing based acceleration techniques for Dijkstra adapted to the robust case.

In a computational study we document the value of different accelerations tried in the algorithm engineering process. We also give an approximation scheme for the robust shortest path problem which computes a $(1+\epsilon)$-approximate solution requiring $\mathcal{O}(\log(\hat{d}/(1+\epsilon)))$ computations of the nominal problem where $\hat{d} := \max d(A) / \min(d(A) \setminus \{0\})$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Graph Algorithms, Shortest Paths, Robust Optimization

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.5

1 Introduction

We develop an algorithm for the cost-robust shortest path problem that significantly reduces the time needed to compute such paths on road networks in practice.

Finding a shortest path from a source s to a sink t in a graph with arc lengths $c(a)$ is a basic algorithmic problem with numerous applications, prominently involving navigation in road networks. Dijkstra's algorithm is the backbone of most navigation applications, but it requires modern acceleration techniques to find within fractions of seconds a route in a network with several hundred thousands or millions of arcs, e.g., in the European road network.

Unfortunately, input data in real-world applications is usually subject to changes, uncertainty or error. For travel times on roads, i.e., arc lengths in shortest path calculations, the change of data is often caused by varying traffic. Several approaches have been proposed to address this problem, including prediction of traffic, leading to time dependent travel times, as well as stochastic models. In this paper we study the classical cost-robust shortest path problem introduced by Bertsimas and Sim. Cost-robust optimization is an alternative approach to handle varying and uncertain data. It minimizes the cost a solution attains in its specific worst-case scenario out of a given set of scenarios. The advantage of the robust approach is that – within the limits of the scenario set – the objective is a deterministic, guaranteed upper bound on the actual travel time.

The scenario set for cost-robustness introduced by Bertsimas and Sim allows each cost coefficient $c(a)$ of a linear cost function to deviate up to a – individual for each variable x_a – maximal deviation $d(a)$. In addition, the number of deviations in a scenario is limited by an input parameter Γ. This is equivalent to limiting by Γ the sum of the fractions of maximal deviations occurring in a scenario. Formally, for a given set of binary variables $\{x_a, a \in A\}$ and vectors c and d in $\mathbb{N}^{|A|}$ the scenario set for the cost-functions is:

$$\left\{ c + \bar{d} : 0 \leq \bar{d}(a) \leq d(a), \forall a \in A \wedge \sum_{a \in A} \frac{\bar{d}(a)}{d(a)} \leq \Gamma \right\}. \tag{1}$$

For this scenario set the cost-robust counterpart of any binary linear program can be solved by solving at most $(|A| + 1)$–many identical binary linear programs with different linear cost functions. More precisely, let Θ contain 0 and all $d(a)$. Then one has to solve the problem for each $\theta \in \Theta$ and the cost function $\Gamma\theta + \sum_A x_a(c(a) + \max(d(a) - \theta, 0))$. Intuitively, the θ enumerates over the smallest deviation $d(a)$ occurring in the scenario. This highly cited result by Bertsimas and Sim applies to cost-robust shortest path, which can thus be found by solving one standard shortest path problem for each arc in the graph.

For a road network with several hundred thousand or millions of arcs this is impractical even when using fast shortest path algorithms. Therefore, we devise a method to significantly reduce the computational effort.

Starting from the Bertsimas and Sim result we use three ways towards practically useful cost-robust shortest path methods. First we reduce the number of θ-values to be examined. Second, we use fast shortest path methods. Third, we reuse previous computations for bounds and goal-directed search, further accelerating the shortest path computations.

It has been proposed [21] that a cost-robust binary problem can be solved by Γ-many copies of the nominal problem. Unfortunately, this result contains a subtle error. We give a counter-example in the appendix which hints to our conviction that essentially $|\Theta|$-many shortest path computations are needed in general.

Accelerated shortest path methods differ on whether they use preprocessing of the graph or not. In this paper, we restrict ourselves to not preprocess the graph. We instead use goal-directed and bidirectional search and adapt both to the cost-robust setting. The high deviations in the arc length in the robust case inhibit the use of traditional preprocessing techniques used for deterministic shortest paths.

1.1 Our contribution

- We give an approximation scheme for general robust combinatorial optimization problems which can be used to compute a $(1 + \epsilon)$-approximate solution using $\mathcal{O}(\log(\hat{d}/(1 + \epsilon)))$ computations of the original problem.

- We introduce a Divide and Conquer approach together with lower bounds for general robust combinatorial optimization problems which can be used to reduce the number of computations of the original problem. The reduction of computations is achieved by carefully reducing the number of θ-values to be considered.
- When applying this to the robust shortest path problem we additionally accelerate the computations of individual shortest paths using pruning and a goal-directed search tailored to the robust shortest path problem.
- We give an efficient method to obtain lower bounds for the length of shortest paths with respect to c_θ. We use these bounds to speed up the Divide and Conquer approach.
- We conduct a computational study showing the effectiveness of our techniques.

1.2 Organization of this paper

We begin by formally introducing the robust shortest path problem in Section 2. We restate the main theorem by Bertsimas and Sim and devise an approximation scheme for the robust shortest path problem. In Section 3 we propose a general framework designed to reduce the number of computations of shortest path computations required to solve a robust shortest path problem. The framework relies on Theorem 3 which is based on the fact that the costs of arcs are non-increasing with respect to θ. We augment this framework by applying shortest path acceleration techniques to the robust shortest path problem. These techniques are search pruning (see Section 4) and goal-direction (see Section 5). The Divide and Conquer framework relies on lower bounds in order to remove dominated values. In Section 6 we devise a method to derive lower bounds of high quality based on information obtained from previous shortest path computations. We include these lower bounds into our Divide and Conquer approach. In order to show the effectiveness of our approach we conduct a computational experiment in Section 7.

1.3 Related work

Robust optimization evolved as a vivid research field during the past decade and shows a broad range of applications, for recent surveys we refer to [5] and [13]. The popularity of robust optimization is in part due to a large area of applications such as network design and routing problems. Network design problem in particular suffer from uncertainty with respect to demands and construction costs. These uncertainties can be treated by adding robustness to the underlying model [3, 20]. Robustness against demand uncertainty is also an important topic in problems such as vehicle routing [11] and lot sizing [22].

An important question with respect to robust optimization is whether or not tractability is preserved for the robust counterparts of polynomially solvable problems. Whether or not this is the case depends on properties of the nominal problem as well as on the employed robust model. For some choices of models, such as minmax regret models, nominally tractable problems become NP-hard (see for example [12]). In contrast, in [6] Bertsimas and Sim introduced a very general robust model which can be applied to many combinatorial optimization problems while preserving tractability.

The model of Bertsimas and Sim has gained wide acceptance and formed a basis for the study of robust combinatorial optimization problems, in particular regarding problems related to the robustness of shortest paths. Büsing considered the problem of robustness and robust recoverability in [8, 7]. In this setting, after a robust scenario has been realized it is still possible to perform some modifications of the previously chosen path in order to recover from the incurring robust costs. The authors of [19] considered the robust shortest path problem with respect to robust costs corresponding to a product of two factors attained according to the model of Bertsimas and Sim. In [21], Poss considered combinatorial problems which can be solved with a dynamic programming approach. The author claimed that the robust counterparts of such problems can be solved with a dynamic program with a size increased by at most Γ. Unfortunately the proof contains a subtle error and the result does not hold. We give a counter-example in the appendix.

Since the ordinary shortest path problem has many real-world applications, considerable effort was put into an accelerated computation. Over the years, different preprocessing techniques such as arc flags [18] and contraction hierarchies [14] were introduced (see [4] for a summary). Preprocessing techniques require an initial offline phase which is used to augment the underlying problem in order to speed up queries in a subsequent online phase. The techniques perform very well in practice, decreasing query times by several orders of magnitude. It was shown in [1] that the query time with respect to preprocessing techniques decreases asymptotically for graphs with low *highway dimension*, a requirement generally satisfied for road networks. A related area of research considers large-scale networks which occur for example in social graphs. Such networks can comprise more than a billion vertices some of which having extremely large degrees. Conventional preprocessing techniques can't be applied in this case. The authors of [9, 16] introduced an inexact preprocessing based on landmarks which is comparable to the approach in [15] for road networks. In contrast the authors of [2] considered a preprocessing technique which either answers the query correctly (in more than 99 % of the queries conducted in their experiments) and fails otherwise.

2 The robust shortest path problem

The robust shortest path problem is defined on a directed graph $D = (V, A)$ with n vertices and m arcs. Each arc $a \in A$ has costs $c(a) \in \mathbb{N}$ and deviations $d(a) \in \mathbb{N}$. A parameter $\Gamma \in \mathbb{N}$ governs the conservatism in accordance with the model of Bertsimas and Sim. Specifically, consider a path P given as a sequence of arcs. A worst-case scenario in the scenario set defined by (1) can be assumed to increase the costs on Γ of the arcs belonging to P to the upper bound d, yielding a total cost of

$$\sum_{a \in P} c(a) + \max_{\substack{S \subseteq P \\ |S| \leq \Gamma}} \sum_{a \in S} d(a). \tag{2}$$

The following theorem shows that the robust shortest path problem can be solved in polynomial time. This theorem and its proof will form the basis of this paper.

▶ **Theorem 1** (Bertsimas and Sim in [6]). *The robust shortest path problem can be solved using at most $m + 1$ computations of nominal shortest paths.*

Proof. We are attempting to find a path minimizing the cost given by (2). We first consider a fixed path P and rewrite the inner optimization problem in terms of variables denoting membership in the set S:

$$\begin{aligned} \max \; & \sum_{a \in P} x(a) \cdot d(a) \\ \text{s.t.} \; & \sum_{a \in P} x(a) \leq \Gamma \\ & 0 \leq x(a) \leq 1 \quad \forall a \in P \end{aligned} \tag{3}$$

This program has the following dual:

$$\begin{aligned} &\min \Gamma\theta + \sum_{a \in P} y(a) \\ &\text{s.t. } y(a) + \theta \geq d(a) \quad \forall a \in P \\ &\qquad \theta, y(a) \geq 0 \qquad \forall a \in P \end{aligned} \tag{4}$$

It is easy to see that $y(a)$ can be fixed to $\max(d(a) - \theta, 0)$. As a result, minimizing (2) is equivalent to finding a path P minimizing

$$\min_{\theta \in \mathbb{R}_{\geq 0}} \Gamma\theta + \sum_{a \in P} c(a) + \max(d(a) - \theta, 0) \tag{5}$$

The function $\theta \mapsto \max(d(a) - \theta, 0)$ is piecewise linear with a break point at $d(a)$. Therefore the function

$$\theta \mapsto \min_{P \in \mathcal{P}} \Gamma\theta + \sum_{a \in P} c(a) + \max(d(a) - \theta, 0) \tag{6}$$

has break points at $d(a)$ for each $a \in A$. It will therefore attain its minimum either at 0 or at some $d(a)$. Thus, a robust shortest path can be found with at most $m + 1$ nominal shortest path computations according to the costs defined by the corresponding values of θ. ◀

Even though the shortest path problem is easily solvable in practice, the overhead of solving $m + 1$ variants renders the robust counterpart intractable in practice. Observe that the number of shortest path computations required in total does not actually depend on the number of arcs but rather on the cardinality of the set

$$\Theta := \{0\} \cup \{d(a) \mid a \in A\}. \tag{7}$$

This suggests an approximation scheme based on solving an instance with a lower number of deviations:

▶ **Theorem 2.** *Let $\hat{d} := \max d(A)/\min(d(A) \setminus \{0\})$, $\epsilon > 0$. A $(1 + \epsilon)$-approximate solution of the robust shortest path problem can be computed with $\mathcal{O}(\log(\hat{d}/(1 + \epsilon)))$ computations of the nominal shortest path problem.*

Proof. Let $\bar{d} : M \mapsto \mathbb{R}_{\geq 0}$ be the values of d rounded up to the next power of $(1 + \epsilon)$:

$$\bar{d}(a) := (1 + \epsilon)^{\lceil \log_{1+\epsilon}(d(a)) \rceil} \quad \forall\, a \in A. \tag{8}$$

There are only $\mathcal{O}(\log(\hat{d}/(1 + \epsilon)))$ different values for θ with respect to $\bar{d}$, which implies that we have to solve only that many instances of the original problem in order to obtain a robust optimum with respect to $\bar{d}$. Let P be a solution of the robust problem with respect to the deviations d. Let $S \subseteq P$ be the set of at most Γ entries causing the robust cost contribution to P with respect to d. In the worst case, every $d(a)$ increases by a factor of less than $(1 + \epsilon)$ from d to $\bar{d}$. Thus, the robust cost contribution with respect to $\bar{d}$ is again caused by the entries in S, increasing the cost of P by less than $(1 + \epsilon)$. ◀

▶ Remark.

1. The approximation guarantee is tight: Consider an instance of the robust shortest path problem given by a digraph consisting of two parallel arcs with pairs of costs and deviations of $(\epsilon/2, (1 + \epsilon)^k + \epsilon/2)$ and $(0, (1 + \epsilon)^{k+1})$, a parameter of $k \in \mathbb{N}_{>0}$ and $\Gamma = 1$. The robust shortest path has a cost of $\epsilon + (1 + \epsilon)^k$, whereas a robust shortest path for the rounded instance costs $(1 + \epsilon)^{k+1}$ in the original instance. As $k \to \infty$ a ratio of $1 + \epsilon$ is achieved.

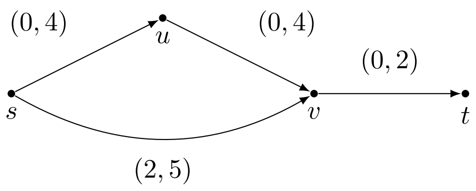


Figure 1 An example for robust shortest paths not forming a tree. Pairs of numbers on arcs represent costs and deviations.

2. Bertsimas and Sim show that robust minimum cost network flow problems can be approximated to a factor of $(1+\epsilon)$ in $\mathcal{O}(\log(m\bar{\theta}/\epsilon))$, where $\bar{\theta} := \max_{a \in A} u_a d_a$ for capacities u. However, robust network flows are not generally integral for integral capacities. Specifically, a robust network flow of one unit no longer corresponds to a path.
3. Recall that the shortest paths problem exhibits an optimal substructure: All shortest paths leaving a common source vertex s can be chosen to form a tree in the underlying graph. This does no longer hold for robust shortest paths, as shown in Figure 1: For $\Gamma = 2$ the unique robust shortest path from s to t leads past vertex u, causing a cost of 8. The robust shortest (s, v) path consists solely of the lower arc.

3 Divide and Conquer

In this section we will describe the main idea used to reduce the number of θ-values which have to be considered to compute a robust shortest path based on Theorem 1. We define $c_\theta(a) := c(a) + \max(d(a) - \theta, 0)$ and observe that this term is non-increasing in θ. The same holds for the cost of a path P defined as $c_\theta(P) := \sum_{a \in P} c_\theta(a)$. For a fixed θ we let

$$c^{\text{opt}}(\theta) := \min_{P \in \mathcal{P}(s,t)} c_\theta(P). \tag{9}$$

Since $c^{\text{opt}}(\theta)$ is the minimum of non-increasing functions, it is non-increasing as well. In order to find a robust shortest path we will minimize the function

$$C^\Gamma(\theta) := \Gamma\theta + c^{\text{opt}}(\theta). \tag{10}$$

If $C^\Gamma(\theta)$ were a convex function in θ, we could use binary search or similar techniques in order to reduce the number of required shortest path computations. Unfortunately $C^\Gamma(\theta)$ is not generally convex. We can however derive the following theorem from the fact that $c^{\text{opt}}(\theta)$ is non-increasing:

▶ **Theorem 3.** *Let $\theta_{\min} < \theta_{\max}$ be in Θ and $\theta \in \Theta \cap (\theta_{\min}, \theta_{\max})$.*
1. *If $c^{\text{opt}}(\theta_{\min}) = c^{\text{opt}}(\theta_{\max})$, then it holds that $C^\Gamma(\theta) \geq C^\Gamma(\theta_{\max})$.*
2. *Let θ^* be in Θ. If $\Gamma\theta + c^{\text{opt}}(\theta_{\max}) \geq C^\Gamma_{\theta^*}$, then the minimum over C^Γ is not attained in $[\theta, \theta_{\max})$.*

Proof. For the first part note that since c^{opt} is non-increasing we have that $c^{\text{opt}}(\theta) = c^{\text{opt}}(\theta_{\min}) = c^{\text{opt}}(\theta_{\max})$. The result then follows from the definition of C^Γ. Turning to the second part, we let $\theta' \in [\theta, \theta_{\max})$. We know that $C^\Gamma(\theta') \geq \Gamma\theta + c^{\text{opt}}(\theta_{\max}) \geq C^\Gamma(\theta^*)$ and therefore $C^\Gamma(\theta)$ is at least $C^\Gamma(\theta^*)$. ◀

Both cases of Theorem 3 enable us to discard an interval of possible values for θ. We therefore use a Divide and Conquer approach as a general framework to speed up computations. The approach works as follows: We maintain a set of intervals of values in Θ together with

Algorithm 1: A Divide and Conquer algorithm for the robust shortest path problem.

```
Algorithm DivideAndConquer
    Input: Digraph $D$, costs $c$, deviations $d$, parameter $\Gamma$, vertices $s, t$
    Output: A robust shortest $(s, t)$-path
    $S \leftarrow \{\Theta\}$
    $\theta^* \leftarrow$ The value of $\min(\Theta)$, $\max(\Theta)$ with lower $C^\Gamma$
    while $S \neq \emptyset$ do
        $I_{\min} \leftarrow$ The interval $I$ from $S$ with the lowest $\min(C^\Gamma(\min(I)), C^\Gamma(\max(I)))$
        if $I_{\min}$ can be discarded then
            continue
        $I_{\min} \leftarrow$ Remove dominated values from $I_{\min}$
        $(I_{\text{low}}, I_{\text{high}}) \leftarrow$ Intervals such that $I_{\text{low}} \cup I_{\text{high}} = I_{\min}$, $|I_{\text{low}} \cap I_{\text{high}}| = 1$, and
          $||I_{\text{low}}| - |I_{\text{high}}|| \leq 1$
        $\theta_{\text{median}} \leftarrow$ The median value, single element in $I_{\text{low}} \cap I_{\text{high}}$
        $\theta^* \leftarrow$ The value of $\theta^*$, $\theta_{\text{median}}$ with lower $C^\Gamma$
        $S \leftarrow S \cup \{I_{\text{low}}, I_{\text{high}}\}$
    return The path corresponding to $\theta^*$
```

the currently best (w.r.t. C^Γ) known value θ^*. We also ensure that the shortest paths with respect to the minimum / maximum of each interval are computed before the interval is considered. At each step of the algorithm we select the interval which has the lowest value of C^Γ at an endpoint. We first use Theorem 3 to try to discard the interval. If the interval can't be discarded we proceed to remove any dominated values. We split the resulting interval into two halves which share exactly one value in Θ, compute the shortest path with respect to that value and decide whether or not to replace θ^*. We then add the two intervals to the set and continue. The details are outlined in Algorithm 1.

Note also that Theorems 3 and 2 (and therefore also Algorithm 1) work for arbitrary robust combinatorial optimization problems.

4 Search pruning

Dijkstra's algorithm explores a graph by *labeling* and *settling* vertices. A vertex is labeled when it is first explored. As soon as a shortest path connecting the vertex is known, the vertex is declared to be settled. Since we compute shortest (s, t)-paths for multiple cost functions c_θ, we reuse information we have gathered from previous computations in order to decrease the number of vertices which have to be labeled / settled in subsequent iterations of Dijkstra's algorithm. The following theorem gives a sufficient condition for excluding vertices during searches:

▶ **Theorem 4.** *Let v be a vertex and $\theta < \theta'$ where $\theta, \theta' \in \Theta$. Let $P_\theta, P_{\theta'}$ be (s, v)-paths that are optimal with respect to c_θ respectively $c_{\theta'}$. Let*

$$\Gamma\theta + c_\theta(P_\theta) > \Gamma\theta' + c_{\theta'}(P_{\theta'}). \tag{11}$$

Then a robust shortest (s, t)-path is either attained for a value $\neq \theta$ or it does not contain v.

Proof. The proof may be found in Appendix A. ◀

We can make the most of this theorem when we evaluate the values of θ in a decreasing fashion. During these computations we maintain a map $\bar{C} : V \to \mathbb{R}_{\geq 0}$. Think of $\bar{C}(v)$ as a known upper bound on the cost of a robust (s, v)-path which we initialize to $\bar{C} \equiv \infty$. When we settle a vertex $u \neq t$ during the search for a shortest path with respect to c_θ we investigate each outgoing arc $(u, v) \in A$. The path leading to u together with (u, v) forms a path P leading to v yielding a value $\Gamma\theta + c_\theta(P)$. If $\Gamma\theta + c_\theta(P) > \bar{C}(v)$ we do not have to label v. Otherwise we label v and decrease $\bar{C}(v)$ to $\Gamma\theta + c_\theta(P)$.

5 Goal-direction

A common extension of Dijkstra's algorithm is known as goal-directed search, introduced in [17]. It is based on a potential $\pi : V \to \mathbb{R}_{\geq 0}$ such that the corresponding reduced costs $c^\pi(u, v) := c(u, v) - \pi(u) + \pi(v)$ are non-negative for each $(u, v) \in A$. It is possible to derive a potential while searching for a shortest path. Consider a search from t in the direction of s. The resulting (partial) shortest-path tree $T = (V(T), A(T))$ is rooted at t and contains all settled vertices. For each $v \in V(T)$ we obtain a path $P(v, t)$ leading from v to the t. Let $c_{\max}(T)$ be the maximum value of $c(P(v, t))$ for $v \in V(T)$. It is then easy to see that the following function is a potential:

$$\pi(v) := \begin{cases} c(P(v, t)) & \text{for } v \in V(T) \\ c_{\max}(T) & \text{otherwise.} \end{cases} \tag{12}$$

In the robust setting, a potential with respect to c_θ is also a potential for $c_{\theta'}$ with $\theta' < \theta$ (since $c_{\theta'} \geq c_\theta$). We use this observation in the following way: We first compute the potential (12) with respect to $\theta_{\max}$ while finding the corresponding path using a backward search. In subsequent forward searches with respect to smaller values in Θ we use this potential. If the costs with respect to θ and $\theta_{\max}$ coincide, the arcs in the backward tree will have zero reduced cost. If all other arcs have nonzero reduced cost, then only the arcs in the shortest paths will have to be settled, greatly decreasing computation time. Intuitively, if θ and $\theta_{\max}$ are close, then the potential computed from $\theta_{\max}$ is an excellent choice for the search with respect to θ.

6 Divide and Conquer for robust shortest paths

We refine Algorithm 1 by exploiting structural properties of the robust shortest path problem. We present our results for a unidirectional search here. In the appendix we show an extension to goal-directed and bidirectional searches in a more general setting.

Consider some interval $I := [\theta_{\min}, \theta_{\max}]$ which appears in the course of Algorithm 1. As an invariant we have completed the Dijkstra search for $\theta_{\min}$. We want to reuse labeling information of this search to derive lower bounds on $C^\Gamma_{\theta_0}$ for some $\theta_0 \in I$. If such a lower bound exceeds the best known upper bound for C^*, we disregard θ_0. In order to accelerate the computation of a robust shortest path, the computation of the lower bound for $C^\Gamma_{\theta_0}$ must be significantly faster than a computation of the path for c_{θ_0}.

We argue about a hypothetical (s, t)-path P and its cost $c_\theta(P)$. The cost is non-increasing and piecewise linear as a function in θ. It has breakpoints whenever θ increases beyond $d(a)$ for some $a \in P$. From this point on the cost $c_a(\theta)$ stays constant at $c(a)$. We know the values $c^{\text{opt}}(\theta_{\min})$ and $c^{\text{opt}}(\theta')$ for some values $\theta' \geq \theta_{\max}$. Whatever the value of $c_{\theta_0}(P)$, the cost of P cannot decrease below these amounts when evaluated at the respective values (see Figure 2).

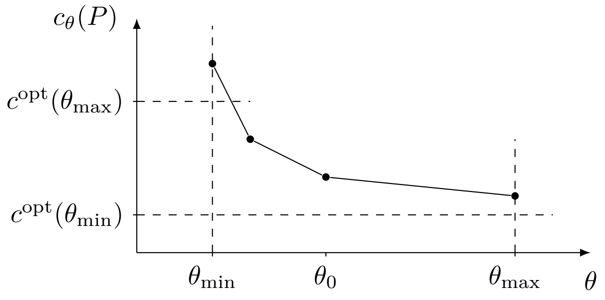


Figure 2 The cost $c_\theta(P)$ of some P. The cost at θ_0 has to be consistent with $c^{\text{opt}}(\theta_{\min})$, $c^{\text{opt}}(\theta_{\max})$.

We go on to formulate a mixed integer program (shown in (13)) to choose an arc set minimizing c_{θ_0}. To make the formulation as strong as possible we choose the smallest possible set of arcs to include into this program: Let $M \subset A$ be the set of scanned arcs, i.e. arcs having a tail which has been settled throughout the search for the shortest (s,t)-path for $\theta_{\min}$. Furthermore, let $M_{\theta_{\min}} \subseteq M$ be the restriction of M to *active* arcs i.e. arcs with $d(a) > \theta_{\min}$. It turns out to be sufficient to consider the arcs in $M_{\theta_{\min}}$ to obtain a lower bound on c_{θ_0}.

We introduce a binary variable x_a for each $a \in A$ denoting whether or not a is contained in P. The variable y models a lower bound on the cost $c_{\theta_{\min}}(P)$ of P yielding (13b). The negative slope of $c_\theta(P)$ at the point $\theta_{\min}$ corresponds to the number of active arcs in P. In the worst case we have $c_\theta(P) = y - \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a),\theta) - \theta_{\min})$ by subtracting from y the contribution of the active arcs. In this case the objective (13a) equals $c_{\theta'}(P)$. However, not all active arcs from $M_{\theta_{\min}}$ can occur in P because for such a path P the value of $c_{\theta'}(P)$ might violate our observations of shortest path lengths for $c^{\text{opt}}(\theta')$. Thus we must raise the variable y to have $c_{\theta'}(P) \geq c^{\text{opt}}(\theta')$. Using the expression for $c_\theta(P)$ from above we obtain (13c) and altogether the following theorem:

▶ **Theorem 5.** *Given an arc set M of scanned arcs during a completed unidirectional search for $c_{\theta_{\min}}$, then a lower bound $O_{\theta_0} \leq c^{opt}(\theta_0)$ is given by*

$$O_{\theta_0} = \min \; y - \sum_{a \in M_{\theta_{\min}}} x_a \cdot (\min(d(a),\theta_0) - \theta_{\min}) \tag{13a}$$

$$\text{s.t.} \; y \geq c^{opt}(\theta_{\min}) \tag{13b}$$

$$y - \sum_{a \in M_{\theta_{\min}}} x_a \cdot (\min(d(a),\theta') - \theta_{\min}) \geq c^{opt}(\theta') \quad \forall\, \theta' > \theta_0 \text{ with known } c^{opt}(\theta') \tag{13c}$$

$$y \geq 0,\; x \in \{0,1\}^{M_{\theta_{\min}}} \tag{13d}$$

The theorem can in fact be further generalized to the bidirectional, goal-directed case. The generalized Theorem 9 and its proof may be found in Appendix A.

The following theorem states that bounds O_θ obtained for multiple θ by Theorem 5 are nonincreasing in θ. This observation can reduce the number of necessary bound computations throughout our algorithm. A proof follows from the more general Theorem 10 in Appendix A.

▶ **Theorem 6.** *For each $\theta_{\min} < \theta_0 < \theta_1$ we have $c^{opt}(\theta_{\min}) \geq O_{\theta_0} \geq O_{\theta_1} \geq c^{opt}(\theta')$ for all θ' that were considered in* (13c) *for both O_{θ_0} and O_{θ_1}.*

It is too time-consuming to solve (13) in order to compute a single bound. We therefore consider a relaxation of the program which can be solved a lot faster while still providing sufficient bounds. Observe that the program has the structure of a multi-dimensional knapsack problem once we fix some value of y. We first relax the integrality of x towards $x \in [0,1]^{M_{\theta_{\min}}}$ and consider a single value $\theta' = \theta_{\max}$ for (13c). What remains is a fractional one-dimensional knapsack problem where arcs correspond to knapsack items:

$$\begin{aligned}
\max \quad & \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta_0) - \theta_{\min}) - y \\
\text{s.t.} \quad & c^{\text{opt}}(\theta_{\min}) \leq y \\
& \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta_{\max}) - \theta_{\min}) \leq y - c^{\text{opt}}(\theta_{\max}) \\
& x \in [0,1]^{M_{\theta_{\min}}}
\end{aligned} \tag{14}$$

Suppose we fix $y = c^{\text{opt}}(\theta_{\min})$. The optimum of the relaxation can be obtained by selecting items greedily w.r.t. their gain, i.e. $\text{gain}(a) := (\min(d(a), \theta_0) - \theta_{\min})/(\min(d(a), \theta_{\max}) - \theta_{\min})$. This leaves exactly one split item a with fractional value for x_a. We argue that increasing y further is not beneficial: An increase of y by ϵ will increase the capacity of the knapsack by ϵ and thereby lead to increased x_a in a greedy optimum. The objective function changes by $\epsilon(\text{gain}(a) - 1)$ which is nonpositive because $\text{gain}(a) \leq 1$ for all arcs. It is therefore never advisable to increase y and we only have to sort the arcs in $M_{\theta_{\min}}$ w.r.t. their gain in order solve problem (14) and obtain a bound O_{θ_0}. Observe that

$$\text{gain}(a) = \begin{cases} (\theta_0 - \theta_{\min})/(\theta_{\max} - \theta_{\min}) & \text{if } d(a) \geq \theta_{\max} \\ (\theta_0 - \theta_{\min})/(d(a) - \theta_{\min}) & \text{if } d(a) < \theta_{\max} \text{ and } d(a) \geq \theta_0 \\ (d(a) - \theta_{\min})/(d(a) - \theta_{\min}) = 1 & \text{if } d(a) < \theta_{\max} \text{ and } d(a) < \theta_0 \end{cases} \tag{15}$$

Thus the value $\text{gain}(a)$ decreases as $d(a)$ increases and it is sufficient to sort the arcs in $M_{\theta_{\min}}$ once according to $d(a)$ in order to compute O_{θ_0} for each $\theta_0 \in \Theta \cap (\theta_{\min}, \theta_{\max})$. We incorporate this *relaxed knapsack bound (RKB)* into the Divide and Conquer approach and apply the generalization of Theorem 5 to goal-directed and bidirectional search.

▶ Remark (Preprocessing). As mentioned above, preprocessing techniques for the ordinary shortest path problem have been extensively studied in the past. Specifically, successful attempts have been made [10] to adapt preprocessing techniques to problems with time-dependent cost functions. Therefore it seems obvious to investigate these techniques with respect to applicability to the robust shortest path problem.

Existing preprocessing techniques operating on problems with changing cost functions generally rely on the ability to provide reasonable bounds on the values attained by the cost functions in order to prune the search space efficiently. Unfortunately, the costs of arcs vary widely between c and $c + d$ in the robust shortest path problem, making it impossible to derive meaningful bounds. As a result we were not able to find preprocessing techniques leading to a significant decrease in query time.

7 Computational experiments

7.1 Experimental network

Due to the long history of experimental evaluations of shortest path algorithms, instances of road networks are ready at hand. However, these networks generally lack data necessary to determine deviation values. Furthermore, shortest path experimentation is conducted on continent-sized networks which are as of yet too large to allow for the computation of robust shortest paths.

We therefore chose to construct a road network ourselves. To this end, we considered a subnetwork of the German road network given by the region of Lower Saxony[1]. We performed the following preprocessing steps in order to obtain a network suitable for routing purposes:

1. We filtered the file to only include ways with `highway` tags, excluding certain highway types such as tracks / service road etc. This process yielded 1.93M nodes and 0.36M ways.
2. We constructed a graph by replacing ways with sequences of arcs, adjusting for one-way restrictions. The resulting graph has 1.93M vertices and 2.17M arcs.
3. We removed directed and undirected chains from the graph. Chains occur frequently as they are used to model the curvature of roads. Therefore the resulting graph shrinks to 0.37M vertices and 0.50M arcs.
4. Since queries for robust paths in an insufficiently connected graph skew computational results we extracted the largest (in terms of the number of vertices) strongly connected component which has 0.15M vertices and 0.23M arcs.

We defined the values of c and d on the network as follows: The nominal length c is defined as the time needed to traverse a segment in accordance with the legal speed limit. To define d we assumed that a certain number of segments is affected by situations such as traffic accidents or road works. If a segment a is affected, the traveling speed drops from the legal speed limit to a value of at most 10 km/h. The value d is chosen such that $c + d$ corresponds to the travel time according to a speed of at most 10 km/h (where $d(a) = 0$ if the speed limit of a is already at most 10 km/h). To avoid numerical problems we rounded both c and d to the nearest second, resulting in $|\Theta| = 1,043$ different deviation values[2]. We further assumed that at most $\Gamma = 5$ road segments suffer from additional congestion.

7.2 Experimental methodology

In order to judge the performance of a shortest path algorithm, the query time of the algorithm is compared to that of Dijkstra's algorithm without any preprocessing applied. This approach raises the following issue: The time to answer a query for a shortest (s,t)-path using Dijkstra's algorithm is highly dependent on the choice of the vertices s and t: If the distance of s and t w.r.t. c is small compared to the diameter of D, then the search explores only a small part of D and finishes quickly. If on the other hand s and t are far apart, then almost the entire graph is explored before a path is found.

This issue can be addressed with the notion of a Dijkstra rank: A search from a fixed source s using Dijkstra's algorithm will settle the vertices in D in the order[3] $s = v_1, v_2, \ldots, v_k$. We define the *Dijkstra rank* of v_j with respect to s as the value j. Note that the distance from s to v_j is non-decreasing and the query time using Dijkstra's algorithm is increasing in the Dijkstra rank. For a pair (s,t) of vertices we define the Dijkstra rank of (s,t) by the Dijkstra rank of t with respect to s.

In order to evaluate the performance of different robust shortest path algorithms we recorded the query time for randomly chosen pairs of vertices with similar Dijkstra ranks. More specifically, we selected pairs of vertices with ranks in $[l \cdot n, u \cdot n)$ where l and u form intervals of size of 10 % of $|V|$.

[1] The initial data was obtained from the OpenStreetMap project, see `https://www.openstreetmap.org`.
[2] The accompanying data may be found at 10.6084/m9.figshare.c.4193588.
[3] We assume that ties are broken consistently.

For each interval we measured the average query time for a sample of 500 random pairs of vertices in order to reduce measurement errors. All query times were obtained using an implementation in the `C++` programming language compiled using the GNU C++ compiler with the optimizing option "-O2". All measurements were taken on an Intel Core i7-965 processor clocked at 3.2 GHz. We implemented Dijkstra's algorithm using binary heaps. Depending on the Dijkstra rank of a pair of vertices, the running time of a shortest path query ranges up to ≈ 35 ms.

7.3 Results regarding search accelerations

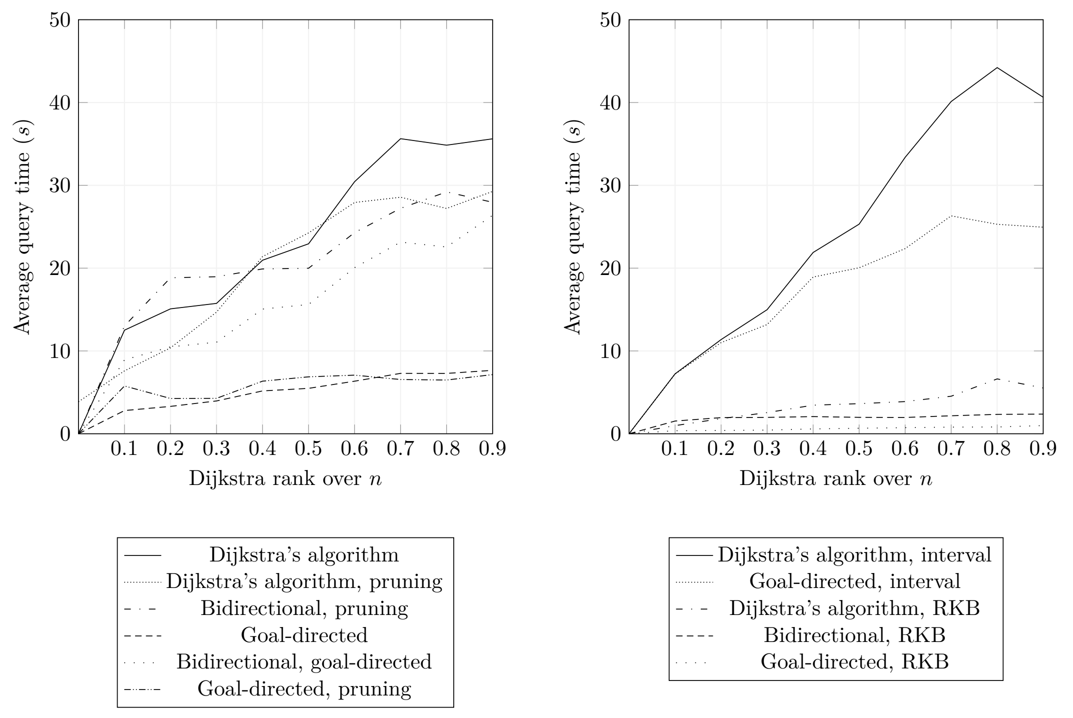
As a first step we evaluated the previously introduced approaches to accelerate individual searches without using the Divide and Conquer approach. The results are depicted in Figure 3a. We remark the following:

1. In order to achieve the best results regarding the goal-directed search we occasionally recompute the potential from scratch. Specifically, we keep track of how many vertices are settled during the recomputation of the potential as well as how many vertices are settled during each subsequent goal-directed search. If the latter amount is within a fraction of α of the former, we reuse the potential in the search to be carried out in the next iteration. Otherwise, we mark the potential to be recomputed during the next query. We found experimentally that a factor of $\alpha = 0.15$ yields the best results.
2. Regarding the bidirectional goal-directed search: We found that the best choice for the combined potential is the average of the two potentials computed during the forward and backward search respectively. Additionally, we found that in order to obtain more accurate potentials it is worth the effort to compute the entire search tree from s to t in the forward search and vice versa in the backward search.
3. Both improvements over Dijkstra's algorithm, the pruning and the goal-directed search, can be combined to speed up the computation even more.

Our findings show that while all approaches lead to reduced computation time, the goal-directed approaches works best, beating a plain evaluation using Dijkstra's algorithm by almost an order of magnitude.

7.4 Results regarding the Divide and Conquer approach

We proceed to consider the impact of the Divide and Conquer approach on the query time (results are shown in Figure 3b). Combining Dijkstra's algorithm with the generic Divide and Conquer approach (Algorithm 1) seems to have little effect on its own. Using the relaxed knapsack bound introduced in Subsection 6 however shows significant improvements. The combination of relaxed knapsack bounds and goal-direction yields the best results with a speedup factor ranging from 34 to 45 with an average of 38. A major contribution to this speed up is due to the fact that the Divide and Conquer approach cuts down on the required number of shortest path computations (see Figure 4): Dijkstra's algorithm alone requires $|\Theta|$-many shortest path computations regardless of the distance between source and target. The value is more than halved using the Divide and Conquer approach, it is cut down to less than ten percent if the relaxed knapsack bound is incorporated.



(a) Average query time for different search accelerations.

(b) Average query time for selected combinations of search accelerations together with the Divide and Conquer approach.

Figure 3 Average query time for different robust shortest path algorithms.

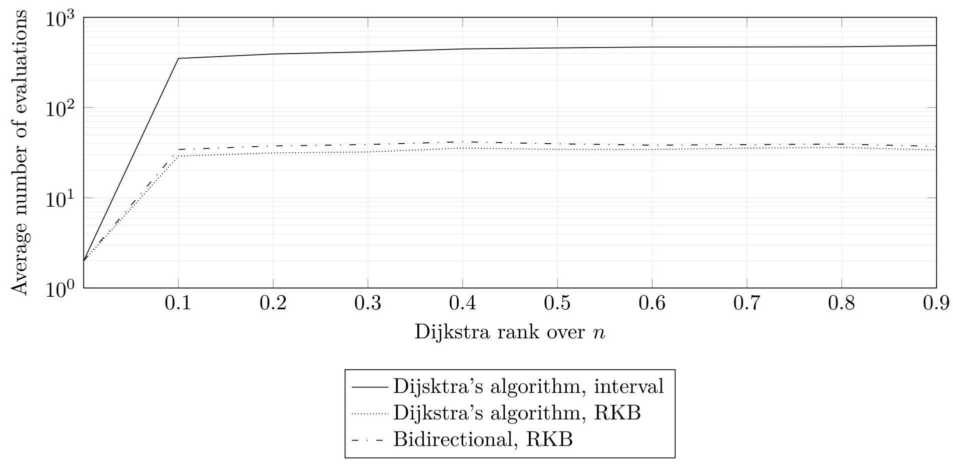


Figure 4 Average number of shortest path computations for different variants of the Divide and Conquer approach. The naive algorithm consistently requires $|\Theta| = 1,043$ evaluations.

8 Conclusion

We have presented an approximation scheme and a Divide and Conquer approach for general robust combinatorial optimization problems. The approximation scheme can be used to trade solution quality and running time. We introduced multiple techniques to accelerate the computation of robust shortest paths without abandoning solution quality ranging from the acceleration of individual queries to augmenting the Divide and Conquer approach by adding efficiently computable lower bounds of high quality. We evaluated our approaches by performing computational experiments on a digraph corresponding to a reasonable large road network. We found that a combination of the acceleration techniques decreased computation time by a factor of up to 45.

As the result for only Γ many shortest path computations does not hold and similar results seem unattainable in light of the counter-example, we give a currently best possible practical approach to solve the fundamental problem of shortest path in the classic Bertsimas Sim model for robustness.

References

1 Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 782–793. Society for Industrial and Applied Mathematics, 2010.

2 Rachit Agarwal, Matthew Caesar, Brighten Godfrey, and Ben Y. Zhao. Shortest paths in less than a millisecond. *CoRR*, abs/1206.1134, 2012. doi:10.1145/2342549.2342559.

3 Alper Atamtürk and Muhong Zhang. Two-stage robust network flow and design under demand uncertainty. *Oper. Res.*, 55(4):662–673, 2007. doi:10.1287/opre.1070.0428.

4 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. *Route Planning in Transportation Networks*, pages 19–80. Springer International Publishing, 2016. doi:10.1007/978-3-319-49487-6_2.

5 Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011. doi:10.1137/080734510.

6 Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98(1):49–71, 2003. doi:10.1007/s10107-003-0396-4.

7 Christina Büsing. The exact subgraph recoverable robust shortest path problem. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos Zaroliagis, editors, *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, pages 231–248. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-05465-5_9.

8 Christina Büsing. Recoverable robust shortest path problems. *Networks*, 59(1):181–189, 2012. doi:10.1002/net.20487.

9 Atish Das Sarma, Sreenivas Gollapudi, Marc Najork, and Rina Panigrahy. A sketch-based distance oracle for web-scale graphs. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 401–410. ACM, 2010. doi:10.1145/1718487.1718537.

10 Daniel Delling and Dorothea Wagner. Time-dependent route planning. *Robust and online large-scale optimization*, 5868(1):207–230, 2009. doi:10.1007/978-3-319-17885-1_101392.

11 Maged M Dessouky, Fernando Ordonez, and Ilgaz Sungur. A robust optimization approach for the capacitated vehicle routing problem with demand uncertainty. *IIE Transactions*, 40(5):509–523, 2008. doi:10.1080/07408170701745378.

12 Maciej Drwal. Complexity of scheduling on parallel identical machines to minimize total flow time with interval data and minmax regret criterion. *CoRR*, abs/1412.4273, 2014. URL: `http://arxiv.org/abs/1412.4273`.

13 Virginie Gabrel, Cécile Murat, and Aurélie Thiele. Recent advances in robust optimization: An overview. *European Journal of Operational Research*, 235(3):471–483, 2014. `doi:10.1016/j.ejor.2013.09.036`.

14 Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In Catherine C. McGeoch, editor, *Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings*, pages 319–333. Springer Berlin Heidelberg, 2008. `doi:10.1007/978-3-540-68552-4_24`.

15 Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

16 Andrey Gubichev, Srikanta Bedathur, Stephan Seufert, and Gerhard Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 499–508. ACM, 2010. `doi:10.1145/1871437.1871503`.

17 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968. `doi:10.1109/tssc.1968.300136`.

18 Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Fast point-to-point shortest path computations with arc-flags. In *The Shortest Path Problem, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, November 13-14, 2006*, pages 41–72, 2006. `doi:10.1090/dimacs/074/03`.

19 Changhyun Kwon, Taehan Lee, and Paul Berglund. Robust shortest path problems with two uncertain multiplicative cost coefficients. *Naval Research Logistics (NRL)*, 60(5):375–394, 2013. `doi:10.1002/nav.21540`.

20 Supakom Mudchanatongsuk, Fernando Ordóñez, and Jie Liu. Robust solutions for network design under transportation cost and demand uncertainty. *Journal of the Operational Research Society*, 59(5):652–662, 2008. `doi:10.1057/palgrave.jors.2602362`.

21 Michael Poss. Robust combinatorial optimization with variable cost uncertainty. *European Journal of Operational Research*, 237:836–845, 2014. `doi:10.1016/j.ejor.2014.02.060`.

22 Muhong Zhang. Two-stage minimax regret robust uncapacitated lot-sizing problems with demand uncertainty. *Operations Research Letters*, 39(5):342–345, 2011. `doi:10.1016/j.orl.2011.06.013`.

A Proofs

We begin by giving the proof of Theorem 4, which was stated as follows:

▶ **Theorem 4.** *Let v be a vertex and $\theta < \theta'$ where $\theta, \theta' \in \Theta$. Let $P_\theta, P_{\theta'}$ be (s, v)-paths that are optimal with respect to c_θ respectively $c_{\theta'}$. Let*

$$\Gamma\theta + c_\theta(P_\theta) > \Gamma\theta' + c_{\theta'}(P_{\theta'}). \tag{11}$$

Then a robust shortest (s, t)-path is either attained for a value $\neq \theta$ or it does not contain v.

Proof. Assume for a contradiction a shortest robust (s,t)-path P is attained for θ and P contains v. P consists of two subpaths, i.e. P_θ and a path P_v leading from v to t. We let P' be the (s,t)-path which consists of $P_{\theta'}$ and P_v. We have:

$$\begin{aligned} C^\Gamma(\theta) &= \Gamma\theta + c_\theta(P_\theta) + c_\theta(P_v) \\ &> \Gamma\theta' + c_{\theta'}(P_{\theta'}) + c_\theta(P_v) \\ &\geq \Gamma\theta' + c_{\theta'}(P_{\theta'}) + c_{\theta'}(P_v) \\ &\geq \Gamma\theta' + c_{\theta'}(P') \end{aligned} \tag{16}$$

We have used here that $c_\theta \geq c_{\theta'}$ which follows from $\theta < \theta'$. This inequality implies that P' is a robust (s,t)-path which is shorter than P which is clearly a contradiction. ◀

We go on to present a more general variant of Theorem 5: We assume that we used a version of Dijkstra's algorithm with respect to reduced costs $c^\pi_{\theta_{\min}}$ obtained from a potential π computed while conducting a search for $c^{\text{opt}}(\theta_{\min})$. During the execution of the search we settled vertices and obtained information regarding the shortest paths for the part of the graph we have explored: In a most general situation, this information is accessible via a fixed arc set $M \subseteq A$ and various subsets $B \subseteq M$ together with bounds $\lambda(B)$ fulfilling

$$\lambda(B) \leq c^\pi_{\theta_{\min}}(P \cap M) \, \forall \, P \in \mathcal{P}(s,t) \text{ with } B \subseteq P. \tag{17}$$

We give some examples for this abstract setting, but first observe that M should contain the arc set corresponding to some $s-t$ cut to yield a bound $\lambda(\emptyset) > 0$. Otherwise the right hand side of the inequality (17) is equal to 0 for some path P with $P \cap M = \emptyset$. In case that a shortest path search completes, it determines $c^{\text{opt}}_\pi(t)$ as the length of a shortest (s,t)-path for $c^\pi_{\theta_{\min}}$, which leads to $c^{\text{opt}}(\theta_{\min}) = c^{\text{opt}}_\pi(t) + \pi(s) - \pi(t)$. This allows us to infer $\lambda(\emptyset) = c^{\text{opt}}(\theta_{\min}) - \pi(s) + \pi(t)$ for the set M containing all scanned arcs. As before we let $M_{\theta_{\min}} \subseteq M$ be the restriction to arcs a with $d(a) > \theta_{\min}$.

► **Example 7.** If we stop unidirectional search prematurely we can use for M the set of arcs, that have a head with settled label and $\lambda(\emptyset)$ can be chosen as the last settled distance label from the search. This situation applies to Theorem 5. Additionally, for some arc $a = (u,v) \in M_{\theta_{\min}}$ we can infer $\lambda(\{a\})$ as the label that v received from u via a because it is a lower bound on $c^\pi_{\theta_{\min}}(P \cap M)$ for any (s,t)-path P that contains a.

► **Example 8.** If some bidirectional Dijkstra search has been stopped prematurely, then let M^s be the set of arcs that have their head settled by the search from s, and let M^t contain the arcs with their tail settled by the search from t. We can use $M = M^s \cup M^t$ and for $\lambda(\emptyset)$ the sum of both lastly settled distance labels in the searches from s and t. For some $B = \{e_s, e_t\}$ $e_s \in M^s$, $e_t \in M^t$, $e_s \neq e_t$ we get for $\lambda(B)$ the sum of the head label from e_s, the tail label from e_t, and both arc costs $c^\pi_{\theta_{\min}}(e_s) + c^\pi_{\theta_{\min}}(e_t)$. Similar bounds for singleton B can be derived as well.

The idea of Theorem 9 is to compute a bound for $c^{\text{opt}}(\theta_0)$ using the abstract bound information. In a suitable program we optimize over the arcs in $M_{\theta_{\min}}$ that an imaginary path P could contain to minimize $c_{\theta_0}(P)$. The program also makes use of values $c^{\text{opt}}(\theta')$ known from previous computations if $\theta' > \theta$.

► **Theorem 9.** *Given a potential π, an arc set M, and a collection $\mathcal{B} \subset 2^{M_{\theta_{\min}}}$, such that bounds $\lambda(B)$ fulfilling* (17) *can be obtained for each B in $\mathcal{B}$, then for each $\theta_0 > \theta_{\min}$, such* *that π is also feasible for c_{θ_0}, we obtain a bound $O_{\theta_0} \leq c^{opt}(\theta_0)$ where O_{θ_0} is an optimum of*

$$\begin{aligned}
&\textit{min} && y - \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta_0) - \theta_{\min}) && && \text{(18a)}\\
&\text{s.t.} && (\lambda(B) + \pi(s) - \pi(t)) \prod_{b \in B} x_b \leq y && \forall B \in \mathcal{B} && \text{(18b)}\\
& && y - \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta') - \theta_{\min}) \geq c^{opt}(\theta') && && \\
& && \forall \theta' : \theta_0 < \theta' \textit{ with } c^{opt}(\theta') \textit{ known} && && \text{(18c)}\\
&\textit{variables:} && y \geq 0,\ x \in \{0,1\}^{M_{\theta_{\min}}} && && \text{(18d)}
\end{aligned}$$

Proof. Let P be any (s,t)-path. We show that $c_{\theta_0}(P) \geq O_{\theta_0}$ holds: Let us consider the arc sets $P', \bar{P} \subseteq P$ given by $P' := P \cap M$ and $\bar{P} := P \setminus P'$.

We claim that setting $x_a := 1$ if and only if $a \in P' \cap M_{\theta_{\min}}$ together with $y := c_{\theta_{\min}}(P') + c_{\theta_0}(\bar{P})$ constitutes a feasible solution to (18) and the cost of this solution is then a lower bound on $c_{\theta_0}(P)$. To get the lower bound we can first write $c_{\theta_0}(P')$ in terms of $c_{\theta_{\min}}(P')$:

$$\begin{aligned}
c_{\theta_0}(P') &= c(P') + \sum_{a \in P' : d(a) > \theta_{\min}} \max\{d(a) - \theta_0, 0\} + c_{\theta_{\min}}(P') - c_{\theta_{\min}}(P')\\
&= c(P') + \sum_{a \in P' : d(a) > \theta_{\min}} \max\{d(a) - \theta_0, 0\} + c_{\theta_{\min}}(P')\\
&\quad - \left(c(P') + \sum_{a \in P' : d(a) > \theta_{\min}} \max\{d(a) - \theta_{\min}, 0\} \right)\\
&= c_{\theta_{\min}}(P') + \sum_{a \in P' : d(a) > \theta_{\min}} (\max\{d(a) - \theta_0, 0\} - \max\{d(a) - \theta_{\min}, 0\})\\
&= c_{\theta_{\min}}(P') - \sum_{a \in P' : d(a) > \theta_{\min}} (\min(d(a), \theta_0) - \theta_{\min})\\
&= c_{\theta_{\min}}(P') - \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta_0) - \theta_{\min})
\end{aligned} \tag{19}$$

Here, the last equality holds, because by its definition P' is fully contained in M and all of its arcs with $d(a) > \theta_{\min}$ are contained in $M_{\theta_{\min}}$. With this expression we obtain

$$\begin{aligned}
c_{\theta_0}(P) &= c_{\theta_0}(\bar{P}) + c_{\theta_0}(P')\\
&= c_{\theta_0}(\bar{P}) + c_{\theta_{\min}}(P') - \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta_0) - \theta_{\min})\\
&= y - \sum_{a \in M_{\theta_{\min}}} x_a(\min(d(a), \theta_0) - \theta_{\min})\\
&\geq O_{\theta_0}
\end{aligned} \tag{20}$$

where the last inequality only holds if x, y is a feasible solution of (18). To prove this feasibility, we first consider (18b) and let $B \in \mathcal{B}$. If $B \nsubseteq P \cap M_{\theta_{\min}}$ then the corresponding Inequality (18b) has its left hand side equal to zero by the definition of x and is feasible. So let $B \subseteq P \cap M_{\theta_{\min}}$ which means that $\prod_{b \in B} x_b = 1$. Feasibility of (18b) in this case then follows from the feasibility of π for c_{θ_0}, we first have:

$$\begin{aligned} y &= c_{\theta_{\min}}(P') + c_{\theta_0}(\bar{P}) \\ &= \sum_{a=(u,v)\in P'} (c^{\pi}_{\theta_{\min}}(a) + \pi(u) - \pi(v)) + \sum_{a=(u,v)\in \bar{P}} (c^{\pi}_{\theta_0}(a) + \pi(u) - \pi(v)) \\ &\geq \sum_{a=(u,v)\in P'} (c^{\pi}_{\theta_{\min}}(a) + \pi(u) - \pi(v)) + \sum_{a=(u,v)\in \bar{P}} (\pi(u) - \pi(v)) \\ &= c^{\pi}_{\theta_{\min}}(P') + \pi(s) - \pi(t) \end{aligned} \tag{21}$$

Here the last equality follows from resolving the telescope sum for the (s,t)-path $P = P' \cup \bar{P}$. Since $B \subseteq P \cap M$ we can use the bound $c^{\pi}_{\theta_{\min}}(P') = c^{\pi}_{\theta_{\min}}(P \cap M) \geq \lambda(B)$ which now implies (18b).

To show that Inequalities (18c) are satisfied, let $\theta' \geq \theta_0$ and $c^{\text{opt}}(\theta')$ be known. We know that $c^{\text{opt}}(\theta') \leq c_{\theta'}(P)$ because P is an (s,t)-path. So we are interested in bounding $c_{\theta'}(P) = c_{\theta'}(\bar{P}) + c_{\theta'}(P')$ against the left hand side of (18c). Because $\theta' > \theta_{\min}$ holds, we can do a similar calculation as before to express $c_{\theta'}(P')$ in terms of $c_{\theta_{\min}}(P')$:

$$c_{\theta'}(P') = c_{\theta_{\min}}(P') - \sum_{a\in M_{\theta_{\min}}} x_a(\min(d(a),\theta') - \theta_{\min})$$

This implies

$$\begin{aligned} c^{\text{opt}}(\theta') &\leq c_{\theta'}(P) \\ &= c_{\theta'}(\bar{P}) + c_{\theta'}(P') \\ &= c_{\theta'}(\bar{P}) + c_{\theta_{\min}}(P') - \sum_{a\in M_{\theta_{\min}}} x_a(\min(d(a),\theta') - \theta_{\min}) \\ &\leq c_{\theta_0}(\bar{P}) + c_{\theta_{\min}}(P') - \sum_{a\in M_{\theta_{\min}}} x_a(\min(d(a),\theta') - \theta_{\min}) \\ &= y - \sum_{a\in M_{\theta_{\min}}} x_a(\min(d(a),\theta') - \theta_{\min}) \end{aligned} \tag{22}$$

where the last inequality holds because $\theta' > \theta_0$ implies $c_{\theta'}(\bar{P}) \leq c_{\theta_0}(\bar{P})$. ◀

▶ **Theorem 10.** *For each $\theta_{\min} < \theta_0 < \theta_1$ such that π is also feasible for c_{θ_0} and c_{θ_1} we have $c^{opt}(\theta_{\min}) \geq O_{\theta_0} \geq O_{\theta_1} \geq c^{opt}(\theta')$ for all θ' that were considered in* (18c) *for both O_{θ_0} and O_{θ_1}.*

Proof. We consider the definitions of (18) for θ_0 and θ_1 respectively. Observe that the sets $M_{\theta_{\min}}$, the bounds $\lambda(B)$ as well as constraints (18b) and (18c) are independent of θ_0 and thus both programs for O_{θ_0} and O_{θ_1} optimize over the same set of feasible solutions. The only difference is the objective function, where for some $a \in M_{\theta_{\min}}$ its coefficient for θ_1 is less or equal than its coefficient for θ_0. This implies $O_{\theta_0} \geq O_{\theta_1}$ but also $c^{\text{opt}}(\theta_{\min}) \geq O_{\theta_0}$: Note that $O_{\theta_{\min}}$ is well-defined and contains only variable y because $M_{\theta_{\min}} = \emptyset$. An optimum is given by $y = c^{\text{opt}}(\theta_{\min})$ and thus $c^{\text{opt}}(\theta_{\min}) \geq O_{\theta_{\min}} \geq O_{\theta_0}$ because $\theta_0 > \theta_{\min}$. Finally, it holds for some θ' which was considered in (18b), that

$$\begin{aligned} O_{\theta_1} &= y - \sum_{a\in M_{\theta_{\min}}} x_a(\min(d(a),\theta_1) - \theta_{\min}) \\ &\geq y - \sum_{a\in M_{\theta_{\min}}} x_a(\min(d(a),\theta') - \theta_{\min}) \\ &\geq c^{\text{opt}}(\theta'). \end{aligned}$$

◀

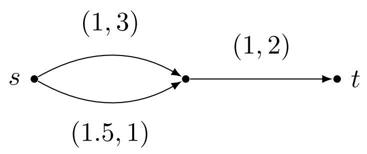


Figure 5 A counter-example to a claim regarding robust combinatorial optimization. Numbers on arcs represent costs and deviations.

B A counter-example to a claim regarding robust combinatorial optimization

We consider a claim made in [21] regarding a type of combinatorial optimization problems solvable by a dynamic programming (DP) algorithm. A combinatorial optimization problem is solvable by a DP algorithm if it can be expressed using a set of functional equations. More specifically, it is assumed that there is a set of states denoted by S with a subset $\mathcal{O}$ of initial states and a final state N. The optimal cost of state $s \in S$ is given by $F(s)$, the set of variables set to 1 in state s is denoted by $q(s)$. The state $p(s,i) \in S$ is set to be the previous state of s where s is obtained from $p(s,i)$ by fixing variable $i \in q(s)$ to 1. The relationship between the states is assumed to be governed by the following set of functional equations:

$$\begin{cases} F(s) = \min_{i \in q(s)}\{F(p(s,i)) + c_i\}, & s \in S \setminus \mathcal{O} \\ F(s) = 0, & s \in \mathcal{O} \end{cases} \tag{23}$$

In order to solve this problem the functional equation is applied to determine the optimal cost of new states until the optimal cost of the final state is determined. The question is whether the robust counterpart of such a problem can be solved in a similar manner using functional equations.

▶ **Theorem 11** (Theorem 6 in the original article). *Consider an instance of a combinatorial optimization problem which can be solved in $\mathcal{O}(\tau)$ for some $\tau : \mathbb{N} \to N$ by using the functional equations* (23)*. Then, its robust version can be solved in $\mathcal{O}(\Gamma\tau)$ using the following functional equations:*

$$\begin{cases} F(s,\alpha) & = \min_{i \in q(s)}\{\max(F(p(s,i),\alpha) + c_i, F(p(s,i),\alpha - 1) + c_i + d_i)\}, \\ & \qquad\qquad\qquad\qquad\qquad\qquad s \in S \setminus \mathcal{O}, 1 \le \alpha \le \Gamma \\ F(s,0) & = \min_{i \in q(s)}\{F(p(s,i),0) + c_i\}, \qquad\qquad s \in S \setminus \mathcal{O} \\ F(s,\alpha) & = 0, \qquad\qquad\qquad\qquad\qquad 0 \le \alpha \le \Gamma, s \in \mathcal{O} \end{cases} \tag{24}$$

As an example of such a problem the authors consider the shortest path in a directed graph with conservative arc costs. It is well known that in this case the Bellman-Ford algorithm finds a shortest path by solving a dynamic program. As a counter-example to the claim stated above, we consider the graph in Figure 5 together with a parameter of $\Gamma = 1$. It should be apparent, that the robust shortest path in this case is the lower path with a total cost of 4.5. In order to compute the shortest path we start evaluating the functional equations for $\alpha = 0$. In this the coefficients coincide with those of the original problem. The graph corresponding to these functional equations is shown in Figure 6. Unfortunately, the path resulting from applying the functional equations is the upper path which has total costs of 5. The failure is due to the fact that the equations do not take into account that the first arc on the upper path has a high value of d.

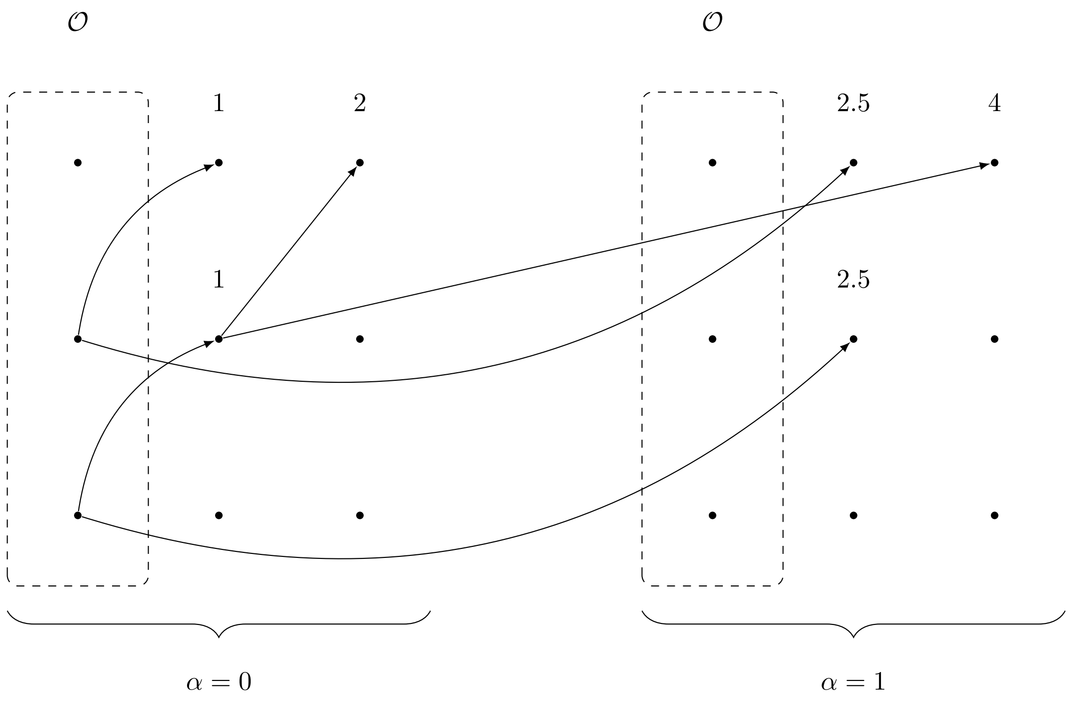


Figure 6 A depiction of the functional equations applied to the robust shortest path problem in Figure 5.

C Figures and tables

The following table contains the average query time plotted in Figures 3a and 3b. Regarding the distribution of the values: As is usually the case when it comes to the evaluation of running times, there is a certain variance in the recorded data. Figure 7 shows the distribution of running times for vertices with large Dijkstra ranks. Note that while the minimum / maximum query times are spread far apart, many of the individual values fall into much smaller intervals around the average. This behavior is consistent throughout the data and justifies the comparison based on the average query time.

Table 1 Average query time in seconds for various algorithms with respect to different ranks

	Dijkstra rank over n									
Algorithm	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Dijkstra's algorithm	0.00	12.51	15.08	15.74	20.95	22.94	30.43	35.63	34.85	35.62
Simple pruning	3.87	7.59	10.37	14.69	21.40	24.25	27.93	28.56	27.21	29.27
Bidirectional, pruning	0.00	13.02	18.84	18.96	19.90	19.99	24.29	27.23	29.20	27.95
Goal-directed	0.00	2.80	3.30	3.96	5.17	5.48	6.35	7.29	7.29	7.66
Bidirectional, goal-directed	0.01	8.93	10.50	11.06	15.06	15.61	20.06	23.14	22.55	26.38
Goal-directed, pruning	0.00	5.77	4.26	4.28	6.35	6.87	7.08	6.55	6.49	7.14
Dijkstra's algorithm, interval	0.00	7.23	11.37	14.98	21.89	25.31	33.38	40.12	44.22	40.64
Goal-directed, interval	0.00	7.21	11.01	13.19	18.92	20.04	22.36	26.31	25.29	24.94
Dijkstra's algorithm, RKB	0.01	0.98	1.83	2.56	3.45	3.64	3.88	4.53	6.62	5.53
Bidirectional, RKB	0.00	1.54	1.95	1.98	2.07	1.98	1.97	2.17	2.34	2.37
Goal-directed, RKB	0.02	0.37	0.40	0.44	0.58	0.67	0.74	0.80	0.82	0.99

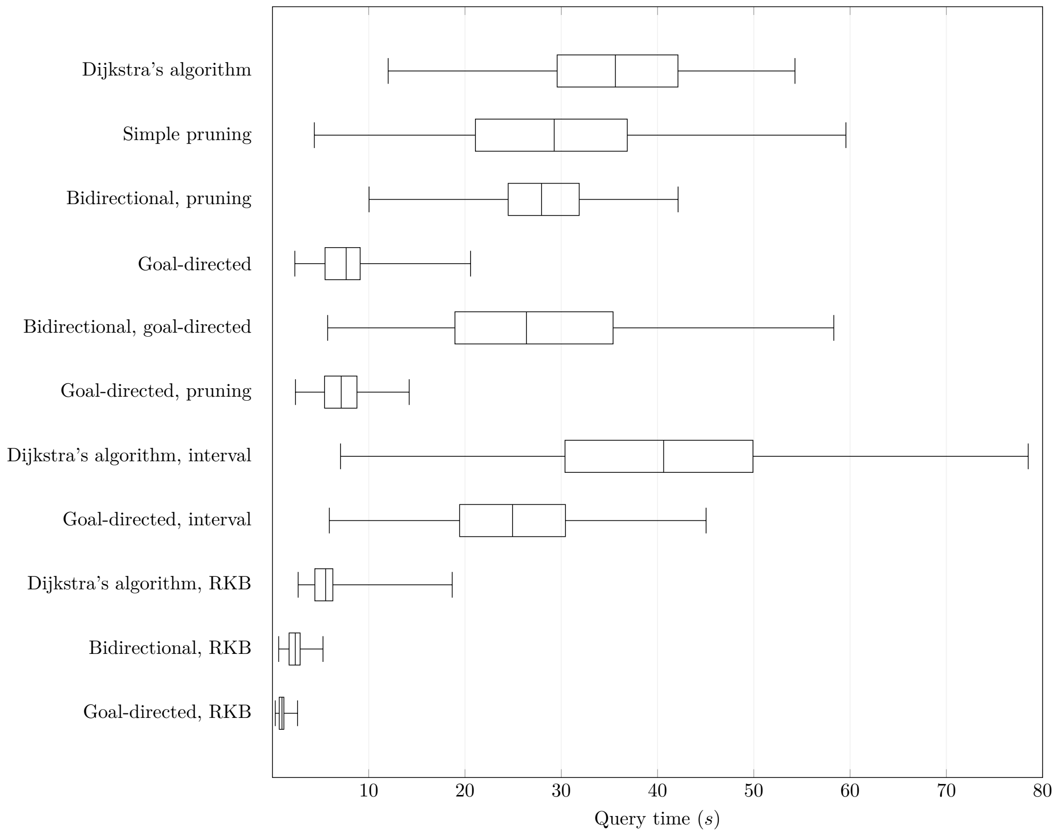


Figure 7 Distribution of the recorded running times. The boxes show minimum, first quartile, average, third quartile, and maximum for a rank of $0.9 \cdot n$.

Tree Decomposition Methods for the Periodic Event Scheduling Problem

Irving van Heuven van Staereling
Centrum Wiskunde & Informatica
Science Park 123, Amsterdam, Netherlands
heuven@cwi.nl / i.i.van.heuvenvanstaereling@vu.nl

Abstract

This paper proposes an algorithm that decomposes the Periodic Event Scheduling Problem (PESP) into trees that can efficiently be solved. By identifying at an early stage which partial solutions can lead to a feasible solution, the decomposed components can be integrated back while maintaining feasibility if possible. If not, the modifications required to regain feasibility can be found efficiently. These techniques integrate dynamic programming into standard search methods.

The performance of these heuristics are very satisfying, as the problem using publicly available benchmarks can be solved within a reasonable amount of time, in an alternative way than the currently accepted leading-edge techniques. Furthermore, these heuristics do not necessarily rely on linearity of the objective function, which facilitates the research of timetabling under nonlinear circumstances.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Dynamic Programming, Trees, Periodic Event Scheduling Problem

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.6

1 Introduction

In many countries with an advanced transport network, the planning process of the transport provider is an extremely complicated and time-consuming procedure. Due to the applications of the algorithms proposed in this paper, we focus mainly on the train timetabling process, although the algorithms presented in this paper are not restricted to this setting. From a high-level point of view, the planning process for train networks, can be divided into the following tasks [1]:

1. Network planning: constructing the infrastructure of the railway network.
2. Line planning: determining the routes (and frequencies) of trains within the railway network.
3. Train timetable generation: determining the arrival and departure times of trains, including their routes through the infrastructure/stations.
4. Rolling stock and personnel planning: assigning the available rolling stock and personnel to the trips.
5. Real time traffic: ensuring the realization of the planning by solving irregularities (e.g., delays) on an operational level.

This paper focuses on a part of the third step within this hierarchy, the design of train timetables (excluding routing through the infrastructure). Due to the numerous constraints that are involved in a timetable, it is practically undesirable or even impossible to construct a feasible timetable manually, which motivates the research for automated timetable generation.

A considerable part of this research is based on the Periodic Event Scheduling Problem (PESP), as initially proposed in [16]. One of the earlier and more influential solution methods in a railway timetabling context is found in [15], which briefly will be described further. Moreover, an overview of the operations research of railway timetabling can be found in [3], while an overview for the PESP in particular (including extensions) can be found in [5].

Overview

As opposed to the modern solution methods that are based on mathematical programming, this research combines dynamic programming based methods with heuristics to find feasible and optimal solutions within the PESP framework. In Section 2, the PESP model will be discussed, alongside its complexity and differences between the model within this paper and the models in the literature.

Section 4 considers a special case of the PESP which can be solved efficiently using dynamic programming, even when a (possibly non-linear) optimization function is used (the standard PESP is a feasibility problem). This dynamic provides the required insights to understand several heuristics that will be proposed in Section 5, whose performance is described in the experimental results in Section 6 and the method is concluded in Section 7. Sections 1 to 3 contains work that for a large part already has been discussed and/or noted in the current literature, while Sections 4 until 7 concern own work.

2 Problem description

2.1 The Periodic Event Scheduling Problem

The Periodic Event Scheduling Problem (PESP) aims to schedule a number of events within a cyclic framework of length T, i.e., all events occur exactly once every cycle. In a railway timetabling context, examples of such events can be the departure, pass-through or arrival of a train at a station.

Define V as the set of events that need to be scheduled, and decision variable $v_i \in [0, T)$ as the time at which event i takes place for all $i \in V$. Within the standard PESP model with constraint set A, every constraint $a \in A$ may only induce a lower and upper bound, respectively L_a and U_a, on the scheduled time difference of two events i and j. Therefore, constraints can be formulated as:

$$(v_j - v_i) \bmod T \in [L_a, U_a] \tag{1}$$

for every $(i, j) \in A$. Thus, every constraint can be specified by two events and two constants. For example, if i and j represent the departure of two different trains from the track, safety regulations could require the trains to depart at least 3 minutes after each other. In this case, $L_a = 3$ and $U_a = 57$ to prevent trains (from possibly different cycles) to coincide, assuming $T = 60$.

A PESP instance can be transformed and visualized in a directed graph $D = (V, A)$, where $n = |V|$ is the number of vertices/variables, and $m = |A|$ is the number of arcs/constraints. For every constraint a, an arc $i \to j$ is introduced and labeled with $[L_a, U_a]$. For convenience, vertices and variables are used as synonyms throughout this paper. The same holds for constraints and arcs. See Figure 1 for a simple example with only three constraints.

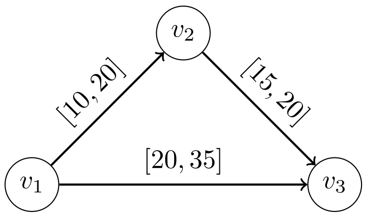


Figure 1 Example of a PESP instance visualized in a graph ($T = 60$).

As a notational remark: x mod T is abbreviated to $(x)_T$, where x can be a number, but also an interval (that will be scaled within the interval $[0, T)$. Since the graph formulation is slightly preferred in the literature, this paper adopts the same notation, which allows the problem to be formally defined as follows.

Periodic Event Scheduling Problem (PESP)	
Given:	A directed graph $D = (V, A)$, a feasible interval $[L_a, U_a]$ for every $a = (i, j) \in A$ and a cycle time T.
Goal:	Find a $v \in [0, T)^n$ such that $v_j - v_i \in [L_a, U_a]$ for every $a = (i, j) \in A$, or state infeasibility.

Trivially, it is assumed that $L_a \leq U_a$ as the instance is infeasible otherwise, and that $U_a - L_a < T$, since the constraint would be redundant otherwise. Moreover, all L_a and U_a are assumed to be integer, which is practically justified because timetables are usually published in minutes (integers). Using this assumption, [10] proved that every feasible PESP-instance then has an integer solution.

Note that by the cyclicity of PESP, the orientation of the arcs can be reversed by "mirroring" the corresponding interval with $T/2$ as the center, i.e., constraints of the type in Equation 1 is equivalent to

$$(v_j - v_i)_T \in [T - U_a, T - L_a]\,. \tag{2}$$

2.2 Complexity of PESP

For $T = 2$, PESP can be solved in polynomial time, for which an algorithm is given in [13]. However, the PESP is strongly NP-complete for $T \geq 3$. At least three proofs are currently known, being reductions from the Linear Ordering Problem [6], the Hamiltonian Cycle Problem [8] and the k-Vertex Colorability Problem [10]. Hence, no (pseudo)polynomial time algorithm can be found to solve the PESP, unless P = NP.

2.3 Handling the modulo operator

Even though the modulo operator follows naturally from the cyclicity of the model, most standard mathematical optimization techniques (such as Branch and Bound) are unable to handle this operator. For this reason, constraints of the type as in Equation 1 are alternatively in the literature formulated as:

$$L_a \leq v_j - v_i + T \cdot p_{ij} \leq U_a \tag{3}$$

at the cost of one extra integer variable p_{ij} per constraint (in similar other models, p_{ij} can also be a binary variable). Here, $p_{ij} \in \mathbb{Z}$ indicates the cycle difference between i and j. In these constraints, p_{ij} is also referred to as the modulo parameter of the constraint. The model now has become suitable for Mixed Integer Linear Programming (MIP) methods.

Using this integer variable, one can implicitly define non-convex intervals, even though the interval $[L_a, U_a]$ for every constraint a is convex. This follows from the possibility in the model to allow multiple constraints between a pair of events, and because L_a and U_a do not necessarily need to be in $[0, T)$. For instance, the two constraints:

$$(v_j - v_i)_T \in [0, 45] \text{ and } (v_j - v_i)_T \in [30, 72]$$

result in a feasible difference interval between v_i and v_j of $[0, 12] \cup [30, 45]$, by the cyclicity of the model. Even though this model is used widely in the literature, this is not the model to be used in this paper, but will be referred to further in this paper for comparison.

2.4 Cost optimization

Although PESP is originally formulated as a feasibility problem, an objective function can be added without complications. One of the easiest, but also practically most useful, objective functions can be deduced from the constraints. In many cases, the lower bound L_a of the constraint is an optimal value to obtain from an efficiency perspective.

For example, if arc $a = (i, j)$ corresponds to the constraint that the changeover time between two trains (that correspond to variables i and j) should lie in $[L_a, U_a]$, the waiting time is minimized if $v_j - v_i = L_a$. If w_a denotes the cost of every time unit that all travellers need to wait longer at the changeover corresponding to the constraint, one could add the term:

$$z_a(v) = ((v_j - v_i)_T - L_a) \tag{4}$$

to an objective function. The objective function, referred to as the weighted slack function, can then be expressed as $z(v) = \sum_{a \in A} w_a \cdot z_a(v)$.

We focus in this paper on this weighted slack function. Other objective functions are discussed in [13] and [7], such as minimization of passenger travel time, required rolling stock, or the number of violated constraints (in case of an infeasible instance), while maximization functions include the profit or robustness.

2.5 Related work

This paper focuses for a large part on heuristics, but will use efficient combinatorial optimization algorithms to solve subproblems if possible. The PESP was originally formulated in [16], where directly several algorithms were proposed. These are primarily searching methods where the modulo parameters are solved first. To this aim, a minimum spanning tree is initially constructed, where the interval cardinalities are used as weights on the arc. The idea is that a solution is found that satisfied the $n - 1$ of the tightest (and therefore expected to be the hardest to fulfill) constraints beforehand, but similar techniques might lead to a brute-force algorithm in an early stage.

Exact methods

A large part of the methods in the current literature focus on the PESP as a feasibility problem, rather than an optimization problem. One of the first solution methods in a railway timetabling context has been implemented by [15] by solving the Mixed Integer Linear Program (MILP) using constraints of the type as in Equation 3. With the aid of the commercial optimization software package CPLEX, solutions for practical railway timetabling instances can be found with the aid of searching algorithms and adjustable parameters within the software package. Other papers that focus on solving the MILP can be found in [11], [12], [7] and [13], using cutting planes and similar other mathematical optimization techniques. A relevant approach, but different perspective is presented in [14], where feasible railway timetables can be found with minimal deviations from the original constraints in case no feasible timetable exists.

Heuristics

A few heuristics already exist that output only very few violated constraints for real-world instances, for example in [4], where cuts and/or local improvements are used to improve the original heuristic from [16]. Although the performance may be relatively good in practice, many of the currently known heuristics struggle with the task of restoring an infeasible solution, without using brute-force early.

The work presented in this paper is similar to the modulo simplex algorithm, firstly presented in [9], and improved by [2], by exploiting advanced methods in the modulo simplex tableau and larger classes of cuts to escape from local optima. This method currently performs best on many benchmarks that are also used for this paper. Still, more ways to backtrack a solution and escape local optima are searched for in the current literature. This paper aspires to contribute to this concept from a difference perspective.

3 State- and search space reduction techniques

From a practical point of view, it may be computationally very beneficial to reduce the state- and search space without excluding feasible solutions. This usually can be achieved fairly simple indeed, especially within a railway timetabling context. In the following paragraphs, several state- and search space reduction techniques are discussed, of which most are also (partially) noted in [7]. Even though most of these methods are straightforward, it is useful to mention these methods (informally) to provide an intuition for the complexity of the reduced problem.

3.1 Intersecting feasible intervals

As also noted in Subsection 2.3, multiple constraints between a pair of variables i and j can be constructed to implicitly define a constraint with a non-convex feasible interval. When using MILP methods, it is essential that a single constraint induces a convex interval. However, the heuristics explained in this paper are not MILP methods, and are not affected by whether these intervals are convex or not. This allows to combine all constraints between a specific pair of variables, into one constraint. To elaborate the possibilities, the following simple definition is introduced for notational convenience.

▶ **Definition 1.** The **feasible interval** Δ_{ij} between variables i and j are the values $v_j - v_i$ such that all constraints $a \in A$ with $i \in a$ and $j \in a$ are satisfied.

Initializing Δ_{ij} can simply be done as follows. For every constraint, scale the feasible interval $[L_a, U_a]$ within the cycle $[0, T)$ and call this new interval Δ_a. For example, $[30, 75]$ will be scaled to $[0, 15] \cup [30, 59]$. Then, let $\Delta_{ij} = \cap_{(i,j) \in A} \Delta_a$. Note that Δ_{ij} instead of Δ_a now may be used as notation, since there exists only one constraint including both variables i and j.

For this reason, constraints are referred to either (i, j, Δ_{ij}) or (i, j, Δ_a). In Subsection 2.1 was argued that the orientation of arcs can simply be redirected, which implies that at most $\frac{1}{2}n(n-1)$ constraints have to be considered.

3.2 Eliminating variables

Variables can be eliminated in two ways.

- For every constraint (i, j, Δ_{ij}) where $|\Delta_{ij}| = 1$, either variable v_i or v_j does not have to be considered for optimization, as its value completely depends on the other variable. Let δ_{ij} be the only value in Δ_{ij}. Assuming v_j will be deleted, all constraints of the type (j, k, Δ_{jk}) can be replaced by $(i, k, (\Delta_{jk} + \delta_{ij}) \mod T)$. A similar shift can be done for constraints of the type (k, j, Δ_{jk}). After solving the model without x_j, its value can easily be determined by $v_j = (v_i + \delta_{ij}) \mod T$.
- If a variable v_i is contained in only one constraint (i, j, Δ_{ij}), the constraint always can be satisfied. After all, consider the problem without v_i. Once v_j is determined, one can afterwards choose $|\Delta_{ij}|$ different values for v_i such that the constraint is satisfied.

3.3 Propagating constraints

Constraint propagation refers to the method of tightening the feasible interval between variable i and j, Δ_{ij}, by combining a series of $\Delta_{ik}, \ldots, \Delta_{k'j}$, where $i \to k \to \ldots \to k' \to j$ is a path from i to j in the PESP graph.

Reconsider the example in Figure 1. There is one direct constraint which initializes Δ_{13} to $[20, 35]$. However, using constraints $(1, 2, [10, 20])$ and $(2, 3, [15, 20])$, it is easy to see this sequence induces a constraint between variable 1 and 3 with feasible interval:

$$[10, 20] \oplus [15, 20] = [25, 40]$$

Hence, Δ_{13} can be reduced to $[20, 35] \cap [25, 40] = [25, 35]$. To describe the method informally, let $P \subseteq A$ be a path from i to j. To reduce the feasible interval Δ_{ij}, consider all possible paths P between i and j and verify whether $\oplus_{a \in P} \Delta_a$ reduces the feasible interval Δ_{ij}. Indeed, the number of possible paths between i and j may be exponential, but a precise description on how to propagate constraints efficiently can be found in [7].

4 The Restricted Periodic Event Scheduling Problem

This section defines and analyzes a special case of the PESP, the so-called Restricted Periodic Event Scheduling Problem (RPESP), which provides the basis for heuristic methods for the PESP in this paper. Even though these heuristics will be explained in detail in the next section, it is helpful to provide a motivation for the upcoming heuristics in a later section, in order to understand the intuition behind the problem considered in this section.

4.1 Motivation

The heuristics in this paper are based on the concept of decomposing a PESP instance into components that each contain a subset of the variables (and therefore also a subset of the constraints), which separately will be solved. Trees are large components, for which will be shown that these can be efficiently solved, and even optimized. To clarify the concept, a few definitions will be introduced first.

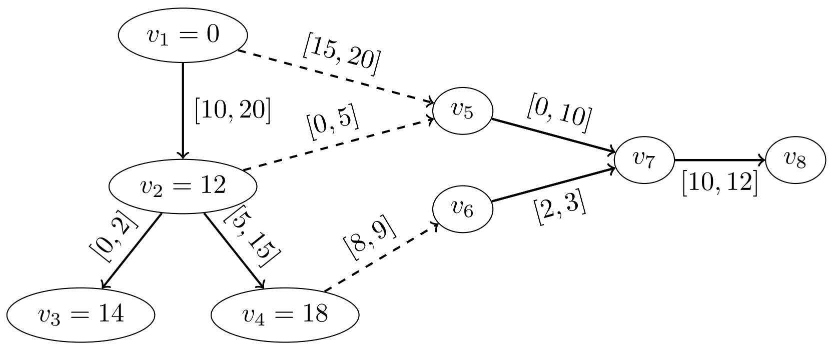


Figure 2 Example of restrictions while integrating components.

▶ **Definition 2.** A PESP instance $C_x = (V_x, A_x)$ is a component of PESP instance $D = (V, A)$ if $V_x \subset V$ and $A_x = \{(i, j) \in A : i, j \in V_x\}$.

It is important to see that whenever a problem $D = (V, A)$ is decomposed into k disjoint subproblems $C_1, \ldots, C_k$ with $\cup_{x=1}^{k} V_x = V$, that A is not necessarily equal to $\cup_{x=1}^{k} A_x$. After all, constraints/arcs that connect two components in the original instance D are not included in $A_1, \ldots, A_k$.

▶ **Definition 3.** The bridging constraints B_{xy} between two components $C_x = (V_x, A_x)$ and $C_y = (V_y, A_y)$ with respect to $D = (V, A)$ are all constraints $(i, j) \in A$ for which $i \in V_x$ and $j \in V_y$.

With this definition, note that $A = \left(\cup_{x=1}^{k} A_x\right) \cup \left(\cup_{x=1}^{k} \cup_{y=x+1}^{k} B_{xy}\right)$. In particular, given two components (or subproblems) C_x and C_y w.r.t. D, the combined subproblem is denoted by $C_{xy} = (V_x \cup V_y, A_x \cup A_y \cup B_{xy})$.

When two components are solved separately, it is likely that the combined solution does not correspond to a feasible solution with respect to D, because the bridging constraints cannot be satisfied. If so, one prefers to make as few adjustments as possible to the components, such that two solutions can be integrated. This idea provides the basis for the heuristics in this paper, and also motivates the consideration of trees because of the following concept.

Suppose that the solution values of the variables in a component C_x are fixed, and one wants to integrate this component, with another component, a tree $C_y = (V_y, A_y)$. The solution within C_x might induce several constraints on the values in C_y (the bridging constraints). Basically, these bridging constraints induce restrictions on the exact values of the variables in C_y, alongside the constraints that already were in C_y. See Figure 2 for an example.

The graph contains 8 variables and 9 constraints. An already solved component C_x is the subgraph containing variables v_1 to v_4. The dashed lines correspond to the bridging constraints, which are not considered when the components are solved individually.

Based on these values, an algorithm needs to determine whether the fixed solution $(v_1, \ldots, v_4)$ w.r.t. C_x can be extended to a feasible solution $(v_1, \ldots, v_8)$ w.r.t. D. To do so, the algorithm needs to solve C_y based on the values $v_1, \ldots, v_4$ and the bridging constraints. In this case, one can easily see that at least $v_5 \in [15, 20] \cap [12, 17] = [15, 17]$ and $v_6 \in [26, 27]$. These constraints need to be taken as a starting point for solving C_y, in order to determine whether a solution for the entire problem can be found with the starting solution for C_x. Such constraints are referred to as exact variable restrictions X_i for variable v_i. This concept motivates the subproblem defined in the following subsection.

4.2 Problem description

▶ **Lemma 4.** *A PESP instance for which the underlying graph $D = (V, A)$ is a tree can be solved in linear time.*

To see the correctness of this lemma, take an arbitrary vertex $i \in V$ and fix v_i with any value (e.g., $v_i = 0$). The possible values from the adjacent variables can be determined directly from the constraints corresponding to the arc. This procedure can be repeated for unfixed variables adjacent to fixed variables, until all variable values are fixed.

As argued in the motivation, so-called variable restrictions will be added to the problem, meaning that every variable v_i might be bound to a specific set of values X_i. This notation allows the RPESP to become formulated as follows.

Restricted Periodic Event Scheduling Problem (RPESP)

Given: A directed, cycle-free graph $D = (V, A)$, a cycle time T, a feasible interval $\Delta_{ij} \subseteq \{0, \ldots, T-1\}$ for all $(i, j) \in A$ and variable restrictions $X_i \subseteq \{0, \ldots, T-1\}$ for all $i \in V$.

Goal: Find a $v \in [0, T)^n$ such $v_j - v_i \in \Delta_{ij}$ for all $(i, j) \in A$ and $v_i \in X_i$ for all $i \in V$, or state infeasibility.

Note that due to the addition of variable restrictions, the problem has become non-trivial and a different algorithm is required.

4.3 Optimizing RPESP

▶ **Theorem 5.** *RPESP can be optimized in $\mathcal{O}(nT^2)$ time.*

Theorem 5 is fundamental for the heuristic in this paper, and will be proven using dynamic programming. To this aim, label a vertex of choice as the root r of the tree, and define $d(i)$ as the minimum number of arcs required from vertex i to reach the root r. A vertex j is a child of i if $d(j) - d(i) = 1$ and there exists an arc between i and j. Similarly, i is the parent of j, which is denoted by $i \downarrow j$.

The dynamic program starts with the vertices at the bottom of the tree (i.e., the vertices without children), and proceeds in a bottom-up fashion by considering in every iteration a vertex of which all children have been considered earlier. Because the graph contains no cycles, such a vertex always exists.

At vertex i, the dynamic program enumerates all feasible solution values for $v_i \in X_i$ and determines for which of these values a feasible solution exists, considering *only* the constraints and variables in the subtree rooted at i (i.e., a subproblem is considered). Using the mentioned model and definitions, the dynamic program will use the following function:

$$f(i, x) = \begin{array}{l}\text{minimum cost of a feasible solution of the subproblem rooted at} \\ \text{vertex } i \text{, while } x \in X_i \text{ and } v_i = x\end{array}$$

with initialization for the leaves as:

$$f(i, x) = \begin{cases} 0 & \text{if } x \in X_i \\ \infty & \text{otherwise} \end{cases}$$

In other words, the subproblem rooted at vertex i using $x_i = t$ is infeasible if and only if $f(i, x) = \infty$. The recursive identity that solves the dynamic program is:

$$f(i, x) = \sum_{(j: i \downarrow j)} \min_{v_j \in \{0, \ldots, T-1\}} \left(f(j, v_j) + z_{ij}(v_i, v_j) \right)$$

for $x_i \in X$. This correctness of the recursion of the dynamic program can be inductively argued as follows. One wants to know the optimal solution value of the subproblem rooted at i, when v_i is fixed at x. Prior to this stage, the dynamic program has determined for all children j of i determined what the optimal value $f(j, v_j)$, for every possible value $v_j = 0, \ldots, T-1$ of the individual subproblems rooted at its children j. Whenever vertex i is added to the subproblem, more terms in the objective function need to be considered. However, by the assumption at the beginning of this section, only terms to the objective function are added between i and its children (i.e., the terms $z_{ij}(v_i, v_j)$ for all j). Since a fixed $v_i = x$ is considered for evaluating $f(i, x)$ and the subproblems rooted at the children of i can be optimized independently of each other, one can simply iterate in linear time what the optimal value for v_j is, including also the terms in $z_{ij}(v_i, v_j)$

The running time of this dynamic program is as follows. Let c_i be the number of children of vertex i. Note that $\sum_{i \in V} c_i = n - 1$, because every vertex, apart from the root, is a child of exactly one other vertex. Computing one value for $f(i, x)$ takes $\mathcal{O}(c_i W)$ time, because for every child $j = 1, \ldots, n_i$ of i, for exactly $|\Delta_{ij}| = \mathcal{O}(W)$ values need to be verified whether there exists a v_j such that $(v_j - x) \bmod T \in \Delta_{ij}$. Since $f(i, x)$ needs to be calculated for at most W values for every vertex $i \in V$, the running time concludes to $\mathcal{O}\left(W \cdot \sum_{i \in V} c_i W\right) = \mathcal{O}(\left(\sum_{i \in V} c_i\right) W^2) = \mathcal{O}(nW^2)$. This proves Theorem 5.

Finally note that the dynamic program can be terminated earlier if it detects for a vertex i that there exists no $f(i, x) < \infty$, as this implies there is no solution for the subproblem rooted at i (and therefore the RPESP instance).

5 Tree decomposition heuristics

Decomposing the PESP into trees is the key technique for heuristics used in this paper to solve PESP instances. The intuition behind this method has been explained in Subsection 4.1: the problem is decomposed in subproblems which are solved independently, and integrated afterwards. If integration is not possible, it is desirable to make a few changes as possible to enable integration. This is elaborated in the next subsections.

5.1 Decomposing a PESP graph into trees

An important part of the algorithm concerns the decomposing of the original graph D into trees. Clearly, this can be done in numerous ways for realistic instances. For this research, a simple greedy heuristic has been applied based on the feasible intervals Δ_{ij}. To describe the method informally, a component C will be initialized by adding the two vertices i and j that correspond to the arc with minimal $|\Delta_{ij}|$. Subsequently, a vertex is added to C if its addition will not lead to a cycle within the component.

The resulting tree graphs, which by definition are components, are denoted as $C_1, \ldots, C_k$. As mentioned earlier, the original graph D is not equal to $\cap_{i=1}^{k} C_i$, since the bridging constraints are not considered. Indeed, when all trees are optimized individually, the bottleneck lies in satisfying the bridging constraints.

5.2 Requirements for partial solutions

▶ Remark. Given two components C_x and C_y w.r.t. D, a given solution v^x can be **extended** to a feasible solution for the (merged) component $C_{xy} = (V_x \cup V_y, A_x \cup A_y \cup B_{xy})$ if and only if there exists a solution to the RPESP instance C_y with variable restrictions $X_j = \cap_{(i,j) \in A: i \in V_x} \left((v_i \oplus \Delta_{ij}) \bmod T\right)$, for all $v_j \in V_y$.

To emphasize the difference, v^x is a **partial solution** to D, but a complete solution to C_x. It is of interest whether v^x can be extended to a feasible solution for the merged subproblem C_{xy}, including the bridging constraints.

To see the correctness of Remark 2, note that by definition, all constraints in A_x are satisfied by definition of v^x. Moreover, by construction of X_i, the bridging constraints B_{xy} are fulfilled if the variable restrictions are satisfied. Hence, the remaining constraints A_y are fulfilled if there exists a solution to the RPESP instance using these variable restrictions.

Note that the dynamic program explained in Subsection 4.3 can answer the question whether a partial solution v^x can be extended to a feasible solution for C_{xy}. Moreover, optimization of an objective can be taken into account to retrieve the best solution for C_{xy} given v^x. This justifies more formally the consideration of the RPESP. Indeed, the next step is to integrate a feasible solution for C_{xy} to a solution for a larger component.

Using this concept, one needs to find partial solutions $v^1, \dots, v^k$ such that $v^x \cup v^y$ is a feasible solution for C_{xy} for all $x = 1, \dots, k$ and $y = x+1, \dots, k$.

Clearly, a prerequisite for every partial solution v^x w.r.t. D is that it can be extended to a solution for the merged subproblem C_{xy} for all $y = 1, \dots, k$. If not, then v^x clearly cannot be extended to a solution for the original problem $D = (V, A)$. One can verify in $\mathcal{O}(knT^2)$ time whether a solution can be extended to a solution for merged subproblems, using the dynamic program.

5.3 Identifying non-extendable partial solutions

The idea will firstly be illustrated informally by reconsidering the example in Figure 2. Given the solution $v^1 = (0, 12, 14, 18)$ for C_1, the bridging constraints impose variable restrictions $X_5 = \{15, 16, 17\}$ and $X_6 = \{26, 27\}$. It turns out that, given the solution v^1 for C_1, that C_2 in fact has become infeasible. After all, the constraint a_{57} demands that $v_7 \in \{15, \dots, 27\}$, while a_{67} demands that $v_7 \in \{28, 29, 30\}$, making the feasible region for v_7 equal to $\{15, \dots, 27\} \cap \{28, 30\} = \emptyset$.

Even though the full PESP-instance is feasible, e.g., $v = (0, 10, 10, 15, 15, 23, 25, 35)$, no feasible solution v^2 for C_2 can be found given the variable restrictions imposed by solution v^1. This clearly means that a different solution for C_1 needs to be found. While attempting to solve C_2, the dynamic program will note this as well, since $f(7, x)$ will be FALSE for all x. Informally, the dynamic program needs to send feedback to C_1 on how to find a feasible solution (that can be extended to a feasible solution for C_2), by imposing additional constraints on finding a solution for v^1 for C_1.

In this specific example, note that a change has to be made in the subset (v_1, v_2, v_4); a feasible value for v_3 can instantly be found due to the tree structure. Thus, one needs to analyze the possible values for (v_1, v_2, v_4) and identify which combinations of values can never lead to a feasible solution for C_2. This procedure will be formalized in the next section.

5.4 Fixing non-extendable partial solutions

▶ **Definition 6.** A **subset ban** $(Y_i, \dots, Y_k)$, with $Y_j \subseteq \{0, \dots, T-1\}$ for $j = i, \dots, k$, is a set of variable values for which any combination $(v_i, \dots, v_k) \in Y_i, \dots, Y_k$ can never extend to feasible solution.

Subset bans basically form an administration of combinations of variables from which the dynamic program already concluded that this leads to guaranteed infeasibility. In this way, an earlier found partial solution for a component C_x can never be considered again, if it has

Table 1 Results of the tree decomposition for the PESP using the PESLlib datasets.

Dataset	Variables	Constraints	Trees	Sol. value	Best value	% Difference
R1L1	3664	6385	5	36.1	31.1	+16.0%
R1L2	3668	6543	4	38.3	31.7	+20.8%
R1L3	4184	7031	5	35.0	30.5	+14.8%
R1L4	4760	8528	4	31.9	27.9	+14.3%
R2L1	4156	7361	4	48.8	42.5	+14.8%
R2L2	4204	7563	5	50.1	43.1	+16.2%
R2L3	5048	8286	4	42.9	39.9	+7.5%
R2L4	7660	13173	4	40.1	33.0	+21.5%
R3L1	4516	9145	5	55.4	45.4	+22.0%
R3L2	4452	9251	5	54.7	46.2	+18.4%
R3L3	5724	11169	5	56.5	43.0	+31.4%
R3L4	8180	15657	5	N/A	35.5	N/A
R4L1	4932	10262	5	61.2	51.7	+18.3%
R4L2	5048	10735	5	64.6	52.0	+24.4%
R4L3	6368	13238	6	N/A	45.8	N/A
R4L4	8384	17754	4	N/A	38.8	N/A

been proven to be non-extendable to another component. When finding a feasible solution from the dynamic program described in 5, one can easily determine a value that fulfills these bans by picking a value x for a variable i for which $f(i,x) < \infty$ and $v_i \notin X_i$.

To complete the heuristic, suppose v^x can be extended to a solution $v^x \cup v^y$ for C_{xy}, and v^x can also be extended to a solution $v^x \cup v^z$ for $C_{x,z}$, where v^y and v^z can be deduced from the dynamic programs. Having found these solutions, this does not necessarily mean that $v^y \cup v^z$ is a solution for C_{yz} (the constraints in B_{yz} have not been considered). This directly implies that $v^x \cup v^y \cup v^z$ is not necessarily a solution to C_{xyz}. This is indeed where exponentiality theoretically can occur. Once multiple trees are integrated in a component C, but are not able to be integrated with another tree C_x, there may be subset bans in C spanning multiple trees. Note that this problem occurs more if the trees are connected to each other, which occurs less in a railway timetabling framework due the railway network (variables/trains in a specific part of the country are less related to variables/trains at the far other end of the country).

6 Experimental results

For this research, the 16 railway timetabling instances from publicly available PESP benchmark library PESPlib[1] have been used. The upper bound for the running time has been set to 1 hour, though if a possible solution can be found, it is usually done within minutes. The remainder of the running time is spent on optimizing the objective function. The results are summarized in Table 1.

All experiments were conducted on a PC with an AMD Ryzen 5 1600 Six-Core Processor (3.20 GHz) with 16 GB of RAM. The source code was written in Java. To clarify Table 1:

[1] `http://num.math.uni-goettingen.de/ m.goerigk/pesplib/`

- **Trees** is the number of trees are the minimum number of trees to which the variables can be decomposed for the tree decomposition heuristic.
- **Sol. value** is the solution value when using the tree decomposition heuristic presented in this paper in millions. If no feasible solution could be found within the time bound, N/A is given.
- **Best value** is the currently best found solution value so far (also in millions), generally by Goerigk & Liebchen.
- **% difference** is the percentual difference between sol. value and best value.

Although the tree decomposition heuristic does not give the hoped results, the performance on these datasets can still be satisfying and at least offer perspective for improvements. Particularly the short duration of the tree decomposition method, for an entire timetable with constraints of an entire country, is one of they key contributions of this paper. To the best of the knowledge presented in this paper, there exists no method that can solve large instances (after data reduction) within such a short amount of time.

Unfortunately, three of the datasets could not be solved by the tree decomposition heuristic. This may be due to the higher number or constraints, or possibly a structure within the constraints where the heuristic cannot deal properly with. Nevertheless, the other 13 datasets could be solved, although the performance is about 20% worse on average than the currently best found solutions. Still, since this method is a heuristic from a new perspective, there is room for improvements and perhaps potential to improve the currently known approaches.

7 Conclusions and future work

The PESP is a difficult problem for which the current literature is seeking more practical methods to escape local optima, without applying brute force in an early stage. This paper has proposed techniques for heuristics that decompose a PESP problems into trees. These techniques are primarily based on dynamic programming, which allows the usage of a smart objective function that heuristically maximizes the possibility that a solution for a component can be extended to a solution for all other components. Experiments are performed using online benchmarks, and the even though the heuristic performs on average about 20% worse in terms of objective function, feasible solutions can still be found quickly.

Future research will be done in improving this method to find feasible and better solutions in a faster way. Other future work concerns the incorporation of heuristics for the PESP into parallel problems; current research includes the routing of trains through stations in parallel to the optimization of the PESP. Due to the highly complex structure of both problems, heuristics are likely to be more suitable than standard optimization techniques.

References

1 M. R. Bussieck, T. Winter, and U.Y. Zimmerman. Discrete optimization in public rail transport. *Mathematical Programming B 79*, pages 415–444, 1997.

2 M. Goerigk and A. Schobel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers & Operations Research 40*, page 1363–1370, 2013.

3 L. G. Kroon, D. Huisman, and G. Maróti. Railway timetabling from an operations research perspective. *Econometric Institute Report EI2007-22*, 2007.

4 C. Liebchen. A cut-based heuristic to produce almost feasible periodic railway timetables. *Lecture Notes in Computer Science*, 2005:354–366, 2005.

5 C. Liebchen and R. H. Mohring. The modeling power of the periodic event scheduling problem: Railway timetabes - and beyond. *Algorithmic Methods for Railway Optimization*, pages 3–40, 2007.

6 C. Liebchen and L. Peeters. Some practical aspects of periodic timetabling. In *Operations Research Proceedings 2001*, pages 25–32, 2001.

7 T. Lindner. *Train Schedule Optimization in Public Rail Transport*. PhD thesis, Braunschweig University of Technology, 2000.

8 K. Nachtigall. Cutting planes for a polyhedron associated with a periodic network. Technical report, DLR Interner Bericht, 1996.

9 Karl Nachtigall and Jens Opitz. Solving Periodic Timetable Optimisation Problems by Modulo Simplex Calculations. In Matteo Fischetti and Peter Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*, volume 9 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/OASIcs.ATMOS.2008.1588`.

10 M. A. Odijk. Construction of periodic timetables, part i: A cutting plane algorithm. Technical report, Delft University of Technology, 1994.

11 M. A. Odijk. Construction of periodic timetables, part ii: An application. Technical report, Delft University of Technology, 1994.

12 M. A. Odijk. A constraint generation algorithm for the construction of periodic railway timetables. In *Transportation Research Part B: Methodological*, volume 30, pages 455–464, 1996.

13 L. W. Peeters. *Cyclic Railway Timetable Optimization*. PhD thesis, Erasmus University Rotterdam, 2003.

14 G. J. Polinder. *Resolving infeasibilities in the PESP model of the Dutch railway timetabling problem*. PhD thesis, Erasmus University Rotterdam, 2015.

15 A. Schrijver and A. Steenbeek. Spoorwegdienstregelingontwikkeling. Technical report, Centrum voor Wiskunde en Informatica, 1993.

16 P. Serafini and W. Ukovich. A mathematical model for periodic event scheduling problem. *SIAM Journal of Discrete Mathematics*, 2:550–581, 1989.

A Bilevel Approach to Frequency Optimization in Public Transportation Systems

Agustín Arizti
Universidad de la República
J. Herrera y Reissig 565, Montevideo, Uruguay
agustin.arizti@fing.edu.uy

Antonio Mauttone
Universidad de la República
J. Herrera y Reissig 565, Montevideo, Uruguay
mauttone@fing.edu.uy

María E. Urquhart
Universidad de la República
J. Herrera y Reissig 565, Montevideo, Uruguay
urquhart@fing.edu.uy

Abstract

We consider the problem of frequency optimization in transit systems, whose objective is to determine the time interval between subsequent buses for a set of public transportation lines. We extend an existing single level model by adding a constraint on bus capacities, while maintaining user choice on routes by means of an assignment sub-model. The resulting formulation is bilevel, and is transformed into a mixed integer linear programming formulation (MILP) that can be solved to optimality for small-sized problem instances, using standard MILP techniques. We study different variants of the same formulation to better understand the bilevel nature of the model and its application to real settings.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases transportation, public transport capacity, transit frequency optimization, mixed integer linear programming, bilevel programming

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.7

Acknowledgements The authors would like to thank to four reviewers, whose comments have improved the manuscript. This work is partially funded by PEDECIBA.

1 Introduction

There are different stages for the design of a public transportation system based on buses. The literature identifies fives stages [4] that are usually performed sequentially in real systems: route network design, frequency setting, timetable design, fleet assignment and crew assignment. The decisions taken at each stage influence the decisions that can be taken at later stages, and they are taken considering different planning horizons, depending on whether the context of the planning is strategic (long term), tactical (medium term) or operational (short term). The frequency setting decisions are usually part of a tactical planning [10], although at least an initial frequency setting is necessary to evaluate the decisions taken during route network design, which happens on a strategic basis.

18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018).
Editors: Ralf Borndörfer and Sabine Storandt; Article No. 7; pp. 7:1–7:13
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The design of a public transportation system needs to consider monetary costs, that range from fixed costs due to the construction of the infrastructure, to variable costs due to the operation of the services. It must also consider the interest of the users, e.g., in providing reasonable travel times, waiting times, and number of transfers. The frequency setting affects directly both concerns, impacting the level of service provided to the users (waiting time, capacity of the lines) and the costs that planners need to incur to run the system (the fleet size is determined by the required frequency of the lines).

The user of a public transportation system usually behaves in an egoistic way, that is, in such a manner as to minimize its individual total travel time (on-board time plus waiting time). Therefore, in order to measure the performance of a transportation system from the viewpoint of the users, models should take into consideration how the users behave when faced with the choice of a specific line from a set of candidate bus lines that can take them to their destinations. Such is the responsibility of an *assignment sub-model*, that by applying a set of hypotheses on how the users behave selects the appropriate lines in order to satisfy travel demands. The assignment model is in itself an optimization problem, usually having a complex formulation and solution method, specially when the influence of the bus capacity is considered in the modelling of the user behavior. Therefore, the complexity of the overall frequency optimization model is strongly determined by the complexity of the underlying assignment sub-model.

In several real settings, public transportation systems run over capacity, meaning that the nominal frequencies of the transit lines are not respected due to lack of capacity. In this context, the capacity is determined by the capacity of the vehicles and the frequencies of the lines. To model these situations, capacity constraints should be taken into consideration when representing the passenger behavior. Even though the problem of transit assignment considering capacities has been properly addressed by the existing literature [15], the problem of transit frequency optimization considering capacities has been more scarcely studied.

The consideration of the bus capacity constraint alongside an assignment sub-model changes the nature of frequency optimization, turning a single level (uncapacitated) formulation into a bilevel one [2]. In bilevel problems there is a constraint that establishes that one or several decision variables must be part of the optimal solution of yet another optimization problem, known as the lower level problem [2] [6]. Exactly two decision makers exist, and the objectives of them do not necessarily coincide. Furthermore, the individual decision each one can take influences the decisions of the other.

The bilevel nature of the frequency optimization problem stems from the fact that the direct addition of bus capacities to the model, involving variables that affect both the planner and the users of the system, would disrupt the underlying assignment sub-model by forcing users to take sub-optimal paths to reach their destination.

The remainder of the article is organized as follows. In section 2 we present a review on related literature and the contributions of this work. The mathematical model and proposed formulation is described in detail in section 3, while in section 4 we present computational experiments using a simple test case on alternative formulations. We conclude the work and refer to future research directions in section 5.

2 Related literature and statement of contribution

In this section we review related relevant literature on frequency optimization in public transportation systems, with a special focus in works that have incorporated either the behavior of the users in an explicit manner (i.e., by means of an assignment sub-model) or bus capacities.

In [8] a nonlinear bilevel formulation for frequency optimization is proposed. It incorporates an explicit assignment model [24] in the lower level, while the upper level problem represents the interest and constraints of the planner, who wants to provide a minimal overall travel time for the users of the system while at the same time diminish the monetary costs by constraining the fleet size. The authors propose a resolution method based on a gradient descent, exploiting specific properties of the problem. The model is applied to several case studies of small to medium sizes.

A bilevel model is proposed in [23], where the upper level seeks to improve an overall cost function and the lower-level consists of the capacity constrained assignment problem formulated in [9]. Tabu Search [17] is used as the heuristic search.

In [20] a MILP formulation is proposed that models user behavior by means of the optimal strategies [24] assignment model. The objective is to minimize the overall travel time of users (on-board travel time plus waiting time) while the operational cost is constrained with an upper limit on the allowed fleet size. The model is solved exactly by using a commercial solver on small instances; for larger instances, a metaheuristic based on Tabu Search is used. The metaheuristic approach is tested using real case studies.

More recently, [18] propose two different integer programming formulations for the problem of designing lines in a public transport system. As part of the line design, frequencies are considered as decision variables to incorporate bus capacities into the model, however, the waiting time of the users is not modeled. Exact solution methods are proposed, and a genetic algorithm is used in order to solve large-scale instances.

The contributions of our work are:

- We consider the passenger behavior as well as the bus capacity and the waiting time of the users, into a single and explicit mathematical programming formulation for the transit frequency optimization problem.
- We propose a bilevel formulation that is converted to a mixed integer linear programming (MILP) formulation suitable of being solved exactly by using commercial MILP solvers for small-sized instances of the problem.
- By applying the exact approach developed to small-sized cases, we are able to study the sensitivity of the solutions with respect to certain aspects of the problem, and therefore, to achieve a better understanding of its nature.

3 Mathematical model

We base our formulation on the one proposed in [20]. In order to model user behavior, it incorporates an explicit assignment model [24].

We propose an extension of the model by adding the bus capacity constraint. This leads us to consider a bilevel formulation that is able to capture the impact that constraints such as the bus capacity, have on the nature of the problem.

3.1 Basic concepts and notation

Before presenting the proposed mathematical programming formulation, we need to provide some concepts as well as a detailed explanation of the used representation.

We make use of a network represented as a directed graph $G = (N, A)$ where nodes acting as bus stops N^P and street endpoints N^S are included in the set N. The movement of the buses along the street is represented by travel arcs (A^T) that connect nodes of N^S. A fixed nonnegative travel time c_a is associated with each travel arc. Boarding (A^B) and alighting (A^L) arcs are also contained in the set A, connecting nodes from N^P to N^S and from N^S to N^P, respectively.

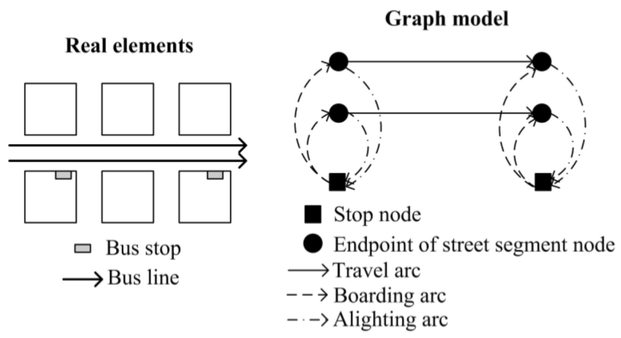


Figure 1 Graph model (extracted from [20]).

We assume that the demand is generated at the bus stops. The demand is represented using an origin-destination matrix, where the set of OD pairs K is such that for a given pair $k \in K$, there are $O_k, D_k \in N^P$ origin and destination nodes, respectively, and a nonnegative value δ_k that represents the amount of people (per time unit in a given time horizon) that have a travel requirement on the pair k.

Lines are defined over the set of travel arcs A^T. Each line $l \in L$ is composed of a sequence of adjacent travel arcs. The round-trip time for a given line is defined as $\sum_{a \in l} c_a$. Lines are either circular, or composed by the concatenation of forward and backward travel arc sequences. Figure 1 illustrates the graph model.

3.2 Assignment model

An assignment model determines user behavior, that is, the way in which users satisfy their travel needs using the existing public transportation lines. Users of the system must choose a line from a set of possible candidate lines that can bring them to their intended destination. Since in order to measure the performance of the system, user satisfaction is of great importance, the assignment model is a critical component of any model of frequency optimization.

The factors that a user considers to make such a choice (i.e., minimize travel time, number of transfers) and the amount of detail and information they have at their disposal (i.e., if the infrastructure provides real time information) determines whether an assignment model is appropriate for the real scenario under study. The way the users behave have a direct influence on the calculation of measures such as the waiting time and occupancy of the buses that end users experience.

The assignment model used in this work is the one proposed in [24], called *optimal strategies*. A strategy is a set of rules that when applied, allow users to reach their destinations. In particular, the model assumes that a given user selects the strategy that minimizes his or her total travel time, including the waiting time at the bus stops. In order to achieve this, it is assumed that users have knowledge of the on-board travel times and frequencies of all the lines of the system. That information is then used to refine a set of attractive lines that can be used to reach the desired destination from the origin. At the bus stop, a given user will take the first bus belonging to the attractive set of lines that passes by that stop. Since the model is probabilistic, an optimal strategy is defined as a strategy that minimizes the total expected travel time.

The probabilistic nature of the model is evident when considering how the waiting time of a passenger waiting on a stop is calculated, for a set of lines $R = \{r_1, \ldots, r_m\}$ with corresponding frequencies $F = \{f_1, \ldots, f_m\}$. As commonly accepted in the literature [10], the waiting time can then be modeled by a random variable of mean value $E(tw) = \beta / \sum_{r_i \in R} f_i$, where β is a parameter which depends on assumptions concerning service regularity. Since the model assumes that passengers take the first bus that arrives at the stop, the probability of using the route r_i, known as the *frequency share rule*, is $P_i = f_i / \sum_{r_j \in R} f_j$.

For a single OD pair, the assignment model can be formulated as follows:

$$\min_{v,w} \quad \sum_{a \in A} c_a v_a + \sum_{n \in N^P} w_n \tag{1}$$

$$\text{s.t.} \quad \sum_{a \in A_n^+} v_a - \sum_{a \in A_n^-} v_a = b_n \qquad \forall\, n \in N, \tag{2}$$

$$v_a \leq f_a w_n \qquad \forall\, n \in N^P, a \in A_n^+, \tag{3}$$

$$v_a \geq 0 \qquad \forall\, a \in A \tag{4}$$

where w_n is the waiting time multiplied by the amount of demand at node $n \in N^P$, A_n^- are incoming arcs to node n, v_a is the amount of demand flowing through arc $a \in A$, f_a is the frequency of the line corresponding to the boarding arc a, and b_n is a value equal to the demand requirement at that node, that is, δ_k if $n = O_k$, $-\delta_k$ if $n = D_k$, and 0 otherwise.

The objective function (1) states the intention of the users of the system, that is, to minimize their total travel time (sum of on-board travel time and the waiting time at the stops). The flow conservation constraint (2) guarantees that all users are able to reach their destinations. Constraint (3) splits the demand among the different lines that belong to the attractive set, and prohibits flow passing through arc a if the arc is not part of the optimal strategy. If $v_a > 0$ the arc must belong to some optimal strategy and the constraint verifies with equality, restoring the frequency share rule expression.

This is a linear formulation that closely resembles a shortest path problem. The particularities of the formulation consist of a new term in the objective function, representing the waiting time at nodes, and constraint (3) that represents what is known as the *split rule*, where demand is split among the attractive lines leading to the destination and passing by the given stop. Due to the latter constraint, the solution of the assignment problem consists of a *hyperpath* [22] representing different trajectories from origin to destination, instead of a single path on the graph as it is the case when solving the shortest path problem.

The model presented above can be easily extended to consider demand generated (both produced and attracted) in places other than the bus stop. This can be done by considering centroid nodes (representing zones of the study region) which are connected to stop nodes through walking arcs.

3.3 Frequency optimization model

The frequency optimization model proposed in [20] is based on the one proposed in [8], which has a nonlinear bilevel formulation. Formulation (5 - 12) is a linear transformation of that original model, where authors introduce a discretization of the domain of frequencies $\Theta = \{\theta_1 \ldots \theta_m\}$ where each element θ_i is a nonnegative value representing a possible value for the frequency of any line.

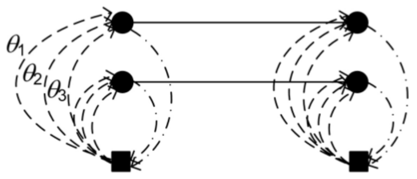


Figure 2 Discretized domain of frequencies (extracted from [20]).

$$
\begin{aligned}
\min_{y,v,w} \quad & \sum_{k\in K}\Big(\sum_{a\in A} c_a v_{ak} + \sum_{n\in N^P} w_{nk}\Big) && && (5)\\
\text{s.t.} \quad & \sum_{l\in L}\sum_{f\in\Theta} \theta_f y_{lf} \sum_{a\in l} c_a \leq B, && && (6)\\
& \sum_{f\in\Theta} y_{lf} = 1 && \forall\, l \in L, && (7)\\
& \sum_{a\in A_n^+} v_{ak} - \sum_{a\in A_n^-} v_{ak} = b_{nk} && \forall\, n\in N, k\in K, && (8)\\
& v_{ak} \leq \theta_{f(a)} w_{nk} && \forall\, a\in A_n^+, n\in N^P, k\in K, && (9)\\
& v_{ak} \geq 0 && \forall\, a\in A, k\in K, && (10)\\
& v_{ak} \leq \delta_k y_{l(a)f(a)} && \forall\, a\in A^B, k\in K, && (11)\\
& y_{lf} \in \{0,1\} && \forall\, l\in L, f\in\Theta. && (12)
\end{aligned}
$$

In doing this, the authors define a new structure of the graph G, where for each line passing by a given bus stop node, there exists as many boarding arcs to that node as possible values of Θ. Figure 2 illustrates the changes introduced in the graph model by using a discretized domain of frequencies.

The model is mixed integer, due to the introduction of the binary variable y_{lf}, which takes value 1 if frequency θ_f is associated with the line l. To keep the planner costs bounded, the parameter B is introduced, which represents an upper limit on the fleet size. To indicate the line frequencies some notation is introduced: $f(a)$ specifies the index in Θ of the frequency associated with the arc a, while $l(a)$ specifies the line that corresponds to that arc. Index k is used to indicate OD pairs.

In formulation (5 - 12) the objective function is that of the users, which intend to minimize their total travel times, while taking into account the interest of the planners that seek to minimize operational costs (6). The assignment model is included in constraints (8 - 10), now expanded to consider each demand pair k. Constraint (7) enforces the fact that each line must have exactly one frequency associated, while constraint (11) prohibits flow on nodes v_{ak} when the frequency associated with that boarding arc is not active ($y_{l(a)f(a)} = 0$) and is redundant otherwise.

This results in a mixed integer linear formulation, where the main source of complexity is the existence of binary variables, and the fact that the discretization of the domain of frequencies increases the size of the underlying graph model due of the addition of new boarding arcs, one per possible frequency value.

3.4 Adding the bus capacity constraint

The assignment sub-model embedded in formulation (5 - 12) assumes that there is sufficient capacity to carry all the passengers that desire to use any line. Furthermore, there is no additional constraint in the formulation that considers the capacity of the lines, which is unrealistic in systems that exhibit high affluence of passengers. Upon introducing a new parameter ω that represents the capacity of a bus, and considering that line capacity (measured in passengers per time unit) is defined as the product of its frequency by the capacity of the bus, we can impose feasible line flows by adding the following constraint:

$$\sum_{k \in K} v_{ak} \leq \sum_{f \in 1..m} y_{l(a)f} \theta_f \omega \quad \forall a \in A^T \tag{13}$$

However, this could result in solutions where the flow of a given OD pair is distributed among:

- A shortest hyperpath comprising lines whose capacity is saturated, i.e., constraint (13) is active for their corresponding travel arcs. This represents the optimal strategy.
- Other alternative hyperpaths, whose cost according to expression (1) is higher than the cost of the shortest one. This represents (sub-optimal) strategies that the users choose a priori, knowing the existence of a shortest hyperpath which is saturated.

This leads us to the concepts of *line planning with route assignment* (LPRA) and *line planning with route choice* (LPRC), first defined in [18]. LPRA models are widespread in the literature, and assume that passengers can be steered by the public transportation planner, an assumption that usually results in simpler but unrealistic models. The utilization of assignment models such as the one used in this work imply a LPRC approach, where each user chooses the route that best fits his or her expectations. Adding constraint (13) directly into the formulation would violate the LPRC approach, as users would need to consider a priori lines that must conform with the new constraint (planners concern) rather than choose the lines in an egoistic way. In a general sense, the addition of any constraint that may impact the variables that model user behavior, and that are not required by the hypothesis of the considered assignment model, would defeat the purpose of the model, since users would behave in a way such as to pursue the optimization of some global optimum that benefits the formulation in place but not necessarily their own interests.

There are at least two ways of modeling the capacity of the buses in the frequency optimization problem while honouring the expected user behavior:

- Assuming that the planner ensures sufficient capacity on the lines that the users want to use. This is done by setting appropriate values of frequencies on the corresponding lines.
- Modeling a congested system, through an assignment sub-model which represents the user behavior under a situation of lack of line capacity. In this case, it is assumed that some users are forced to wait for the next bus of the line, with available capacity, or wait for a different line.

The second one entails to consider an equilibrium assignment sub-model [5] [9] embedded into the frequency optimization model, which is considerably more complex than the first approach [13]. Furthermore, to the best of our knowledge, there is not a formal criterion to decide between both approaches from the modeling point of view. In practice, constraints related to capacity of infrastructure, budget and policy come into play to determine whether it is possible to operate a not congested system. In this work we follow the first approach.

3.4.1 Bilevel mathematical programming formulation

If constraint (13) is added to formulation (5 - 12) we would be considering decisions taken by different actors in the same model. Variables y represent planner decisions in assigning frequencies to lines, while variables v and w represent decisions of the users, that select which lines to use to reach their destinations. Bilevel mathematical programs [2] [6] [12] are used to model scenarios with similar characteristics.

In order to incorporate the bus capacity constraint in our model, we propose the following bilevel formulation:

$$\min_{y,v,w} \quad \sum_{k\in K}(\sum_{a\in A} c_a v_{ak} + \sum_{n\in N^P} w_{nk}) \tag{14}$$

$$\text{s.t.} \quad \sum_{l\in L}\sum_{f\in\Theta}\theta_f y_{lf}\sum_{a\in l} c_a \le B, \tag{15}$$

$$\sum_{f\in\Theta} y_{lf} = 1 \qquad \forall\, l\in L, \tag{16}$$

$$\sum_{k\in K} v_{ak} \le \sum_{f\in\Theta} y_{l(a)f}\theta_f\omega \qquad \forall\, a\in A^T, \tag{17}$$

$$y_{lf}\in\{0,1\} \qquad \forall\, l\in L, f\in\Theta, \tag{18}$$

$$\min_{v,w} \sum_{k\in K}(\sum_{a\in A} c_a v_{ak} + \sum_{n\in N^P} w_{nk}) \tag{19}$$

$$\text{s.t.} \sum_{a\in A_n^+} v_{ak} - \sum_{a\in A_n^-} v_{ak} = b_{nk} \qquad \forall\, n\in N, k\in K, \tag{20}$$

$$v_{ak} \le \theta_{f(a)} w_{nk} \qquad \forall\, a\in A_n^+, n\in N^P, k\in K, \tag{21}$$

$$v_{ak} \le \delta_k y_{l(a)f(a)} \qquad \forall\, a\in A^B, k\in K, \tag{22}$$

$$v_{ak} \ge 0 \qquad \forall\, a\in A, k\in K. \tag{23}$$

where the upper level (14)-(18) represents decisions of the planners while the lower level (19 - 23) represents decisions of the users, that is, the assignment sub-model with the input of fixed frequencies $\theta_{f(a)}$. The objective function of both levels is the same, considering only the objective of the users, which is to minimize the overall travel time. Arguably, the fleet size constraint (15) could be modeled as another objective to minimize at the upper level, which would lead us to consider a multi-objective bilevel formulation, probably increasing the complexity of the formulation [16].

The planners can ensure sufficient capacity on the lines that the users want to use by adjusting the frequencies according to constraint (17). In that manner, users are assumed to perceive unlimited capacities on the lines they might take.

Formulation (14 - 23) is classified as Discrete Continuous Linear Bilevel (DCLB) [2] since the upper level is linear with discrete variables while the lower level is linear with continuous variables. Therefore, it can be reformulated into a MILP problem and in theory it could be solved to optimality. Some commonly used reformulation strategies for doing this are:

- Using the Karush-Kuhn-Tucker (KKT) conditions to substitute the lower level problem and therefore removing the distinction among the different levels. Due to the complementarity term, that is not linear, the resulting reformulation would be a standard single level nonlinear mathematical program that is suitable to be solved by some of the existing nonlinear algorithms. Usually, the reformulation is combined with a linearization of the complementary slackness term using the *big-M* method [12]. This approach has been described and used in [2] [12].

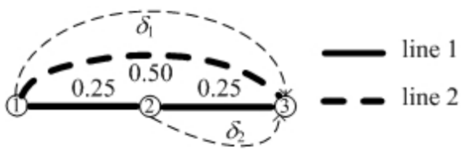


Figure 3 Illustrative example.

- Primal-Dual reformulation. In this case the lower level problem is replaced by using its dual constraints, primal (original) constraints, and the strong duality theorem equality (equality between the lower and upper level objective functions), since the KKT conditions are equivalent to the later conditions when the lower level problem is linear. This approach has been used in [1] [3] [14].

In the present work formulation (14 - 23) was transformed into a single level formulation using the first approach, that is, by replacing the lower level problem by the optimality conditions given by its constraints, the constraints of its dual and the complementary slackness constraints, which were linearized using the *big-M* method. In that way, by replacing the lower level with its optimality conditions, variables which represent decisions of the users (v and w) are restricted to take values which solve problem (19 - 23). Therefore, the whole model will adjust the frequency values (variable y) so as to respect the constraints which are directly included in the upper level (among them, bus capacity) as well as the optimality conditions which represent the (uncapacitated) lower level problem.

After applying the KKT conditions, the resulting MILP model, equivalent to (14 - 23), is (24 - 51), where (33 - 36) correspond to the constraints of the dual of problem (19 - 23), π_{nk}, ν_{ak}, and μ_{ak} are the dual variables corresponding to constraints (20), (21), and (22), respectively, s^1_{ak} and s^2_{ak} are slack variables associated with inequality constraints (21) and (22), respectively, and t^1_{ak}, t^2_{ak} and t^3_{nk} are slack variables associated with the inequality constraints (33), (34) and (35), respectively. The complementary slackness conditions are linearized by applying the big-M method (37 - 46), obtaining in this manner a MILP single level formulation.

4 Experiments for a small-sized example

In order to illustrate the application of the bilevel model explained in section 3, we show in Figure 3 the small-sized case considered.

The numbers close to the arcs indicate their corresponding travel times. There are two OD pairs, such that $O_1 = 1$, $O_2 = 2$, $D_1 = D_2 = 3$ and $\delta_1 = \delta_2 = 5$. We consider values of fleet size $B = 10$, bus capacity $\omega = 1.0$ and the set of possible frequencies $\Theta = \{1.0, 2.5, 5.0, 7.0, 9.0\}$. The lines defined for this case are $l_1 = \{(1,2),(2,3)\}$ and $l_2 = \{(1,3)\}$, both having symmetrical forward and backward itineraries.

Table 1 shows the results of applying three different variants of formulation (24 - 51) to the example of Figure 3, where τ (calculated in (24)) is the total travel time of the optimal solution and β (calculated in (25)) its corresponding fleet size; it also shows the line capacity (as defined in expression (13)) and the critical flow of each line (defined as the flow of the arc v_a with maximum flow on the line). Even though the model has a large number of variables, due to the small size of the instance, the execution time is negligible.

Table 1 Impact of adding the bus capacity constraint.

Model	cap. l_1	critical flow l_1	cap. l_2	critical flow l_2	τ	β
uncapacitated	9.0	$9/10\delta_1 + \delta_2 = 9.5$	1.0	$1/10\delta_1 = 0.5$	4.8	≤ 10
cap. single-level	9.0	$8/10\delta_1 + \delta_2 = 9.0$	1.0	$2/10\delta_1 = 1.0$	5.3	≤ 10
cap. bilevel	9.0	$9/11.5\delta_1 + \delta_2 = 8.9$	2.5	$2.5/11.5\delta_2 = 1.1$	≤ 4.8	11.5

$$
\begin{aligned}
\min_{y,v,w} \quad & \sum_{k\in K}\Big(\sum_{a\in A} c_a v_{ak} + \sum_{n\in N^P} w_{nk}\Big) && && (24)\\
\text{s.t.} \quad & \sum_{l\in L}\sum_{f\in\Theta} \theta_f y_{lf} \sum_{a\in l} c_a \leq B, && && (25)\\
& \sum_{f\in\Theta} y_{lf} = 1 && \forall\, l \in L, && (26)\\
& \sum_{k\in K} v_{ak} \leq \sum_{f\in\Theta} y_{l(a)f}\theta_f\omega && \forall\, a \in A^T, && (27)\\
& \sum_{a\in A_n^+} v_{ak} - \sum_{a\in A_n^-} v_{ak} = b_{nk} && \forall\, n \in N, k \in K, && (28)\\
& v_{ak} \leq \theta_{f(a)} w_{nk} && \forall\, a \in A_n^+, n \in N^P, k \in K, && (29)\\
& v_{ak} \geq 0 && \forall\, a \in A, k \in K, && (30)\\
& v_{ak} \leq \delta_k y_{l(a)f(a)} && \forall\, a \in A^B, k \in K, && (31)\\
& y_{lf} \in \{0,1\} && \forall\, l \in L, f \in \Theta, && (32)\\
& \pi_{ik} - \pi_{jk} \leq c_a && \forall\, a = (i,j) \in A - A^B, k \in K, && (33)\\
& \pi_{ik} - \pi_{jk} - \mu_{ak} - \nu_{ak} \leq c_a && \forall\, a = (i,j) \in A^B, k \in K, && (34)\\
& \sum_{a\in A_n^{B+}} \theta_{f(a)}\nu_{ak} \leq 1 && \forall\, n \in N, k \in K, && (35)\\
& \mu_{ak}, \nu_{ak} \geq 0 && \forall\, a \in A^B, k \in K, && (36)\\
& \theta_{f(a)} w_{ik} - v_{ak} \leq s^1_{ak} M && \forall\, a = (i,j) \in A^B, k \in K, && (37)\\
& \nu_{ak} \leq (1 - s^1_{ak})M && \forall\, a \in A^B, k \in K, && (38)\\
& \delta_k y_{l(a)f(a)} - v_{ak} \leq s^2_{ak} M && \forall\, a \in A^B, k \in K, && (39)\\
& \mu_{ak} \leq (1 - s^2_{ak})M && \forall\, a \in A^B, k \in K, && (40)\\
& c_a - \pi_{ik} + \pi_{jk} \leq t^1_{ak} M && \forall\, a = (i,j) \in A - A^B, k \in K, && (41)\\
& v_{ak} \leq (1 - t^1_{ak})M && \forall\, a \in A - A^B, k \in K, && (42)\\
& c_a - \pi_{ik} + \pi_{jk} + \mu_{ak} + \nu_{ak} \leq t^2_{ak} M && \forall\, a = (i,j) \in A^B, k \in K, && (43)\\
& v_{ak} \leq (1 - t^2_{ak})M && \forall\, a \in A^B, k \in K, && (44)\\
& 1 - \sum_{a\in A_n^{B+}} \theta_{f(a)}\nu_{ak} \leq t^3_{nk} M && \forall\, n \in N, k \in K, && (45)\\
& w_{nk} \leq (1 - t^3_{nk})M && \forall\, n \in N, k \in K, && (46)\\
& s^1_{ak} \in \{0,1\} && \forall\, a \in A, k \in K, && (47)\\
& s^2_{ak} \in \{0,1\} && \forall\, a \in A^B, k \in K, && (48)\\
& t^1_{ak} \in \{0,1\} && \forall\, a \in A - A^B, k \in K, && (49)\\
& t^2_{ak} \in \{0,1\} && \forall\, a \in A^B, k \in K, && (50)\\
& t^3_{nk} \in \{0,1\} && \forall\, n \in N, k \in K && (51)
\end{aligned}
$$

4.1 Experiment 1: comparison of uncapacitated and single level capacitated models

The first line of Table 1 shows the results of applying the uncapacitated model (5 - 12). When capacities are not considered, the entire flow of OD pair 2 uses l_1, while the flow of OD pair 1 is distributed between both lines (4.5 uses l_1 and 0.5 uses l_2) due to the flow splitting constraint (9).

When we consider bus capacities in the original uncapacitated model (second line of the table), adding the constraint directly, we obtain the same setting of frequencies but with a different assignment of flows. In this case, 1.0 units of the demand corresponding to OD pair 1 uses l_2. This is because l_1 has capacity to accommodate only up to 9.0 units of flow. The 0.5 units of flow corresponding to OD pair 1, which were moved from l_1 to l_2 represent a set of users who are forced to use a sub-optimal hyperpath, knowing the existence of a better one, that is, they behave in an unrealistic way. Moreover, we note that the model is not able to represent this situation consistently, since it can not represent different waiting times for passengers corresponding to the same OD pair at the same stop (variables w_{nk}).

The example shows through a numerical application, the consequences of solving the capacitated problem in a straightforward (not realistic) way. When we apply the bilevel model (24 - 51) to the same case, we obtain no feasible solution. This is due to the fleet size constraint, that does not allow for an increase of frequencies in order to accommodate the demand on the lines that the users want to use; moreover, the model is not able to change the frequencies in such a way as to redistribute the flows in order to respect the line capacities. That difficulty was already noted in [7]. In order to overcome this difficulty, we identify two approaches in the literature:

- Soften the bus capacity constraint, by moving it as a term of the objective function [7].
- Allow the model to increase the fleet size, by including its respective constraint in the objective function [19].

By adopting the first approach, the solutions obtained may violate the bus capacity constraint; the higher the violation, the less valid is the corresponding assignment of flows, which is done assuming sufficient capacity. On the other hand, the second approach assumes that the fleet size can be increased. This may be a reasonable assumption in the context of strategic planning, where the model can be used to estimate the investment required to offer a given level of service. In this case, by adding a new objective function the resulting model becomes multi-objective, which requires a special treatment depending on how this nature is represented: for example, by setting appropriate weights or calculating non-dominated solutions [21].

4.2 Experiment 2: calculation of required fleet size

Considering the discussion above, another possible application of the bilevel model to the capacitated case would be to state the fleet size minimization as upper level objective, subject to a constraint of maximum travel time; that is, swapping objective function (14) and constraint (15).

The results of applying this model to the small instance can be found in the third line of Table 1, where we state a maximum travel time equal to 4.8 (the optimal value of the uncapacitated model). The optimal value in this case (which corresponds to the fleet size), is equal to 11.5. The interpretation of the result is that in order to obtain a setting of frequencies which respects the bus capacity constraint while at the same time producing a total travel time which is no worse than the one corresponding to the uncapacitated case, the fleet size should be increased in 15%.

5 Conclusions and further research

In the present work we propose a new bilevel formulation for transit frequency optimization, based on the model presented in [20]. The proposed model considers individual passenger route choice, using an assignment model [24], as well as considering the waiting time of the users and the bus capacity when measuring the performance of the system. We derived a mixed integer linear programming (MILP) formulation which is equivalent to the bilevel one, that is susceptible of being solved by common solvers using standard MILP techniques, for small-sized problem instances.

We have also explored the bilevel nature of the problem by applying different formulations to the same example instance. The results obtained suggest that a true bilevel approach should be considered whenever bus capacities are contemplated, and that uncapacitated models are able to produce solutions that are not appropriate in contexts where the transit system is operating over its capacity.

We note that all variants of the bilevel model discussed here maintain the DCLB structure. This enables to apply exact solution methods. However, the existing (general purpose) solution methods for this kind of bilevel problems [2] [3] [12] do not necessarily handle models with many variables and constraints, as it is the case of frequency optimization problems. Therefore, further research is needed in order to devise tailored solution methods for the specific problem. An example of such an approach can be found in [18]. Metaheuristic techniques may also aid in finding good solutions to solve the transit frequency optimization problem. The *Tabu Search* [17] based metaheuristic presented in [20] to solve a single level instance of the problem might also be extended to cope with a bilevel program. There are a growing number of metaheuristic approaches that deal with bilevel problems. A good survey can be found in [11].

It is also desirable to apply the proposed formulation to instances corresponding to real cities of medium size, in order to study the scalability of the method and the improvements obtained when compared to the current solutions of real transportation systems. The addition of other constraints, such as enforcing a maximum waiting time for users of the transit system may further help achieving solutions of good performance in real world contexts.

Regarding capacitated models, a formal criterion for switching between uncongested and congested frequency optimization models would be desirable to establish.

References

1 José M. Arroyo. Bilevel programming applied to power system vulnerability analysis under multiple contingencies. *IET Generation, Transmission and Distribution*, 4(2):178–190, 2010. doi:10.1049/iet-gtd.2009.0098.

2 Jonathan F. Bard. *Practical bilevel optimization.* Kluwer, 1998.

3 Luis Baringo and Antonio J. Conejo. Transmission and Wind Power Investment. *IEEE Transactions on Power Systems*, 27(2):885–893, 2012. doi:10.1109/TPWRS.2011.2170441.

4 Avishai Ceder and Nigel H. M. Wilson. Bus network design. *Transportation Research B*, 20(4):331–344, 1986.

5 Manuel Cepeda, Roberto Cominetti, and Michael Florian. A frequency-based assignment model for congested transit networks with strict capacity constraints: characterization and computation of equilibria. *Transportation Research B*, 40(6):437–459, 2006.

6 Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, 2007.

7 Isabelle Constantin. *L'optimisation des fréquences d'un réseau de transport en commun.* Doctorate thesis on informatics, Université de Montréal, 1992.

8 Isabelle Constantin and Michael Florian. Optimizing frequencies in a transit network: a nonlinear bi-level programming approach. *International Transactions in Operational Research*, 2(2):149–164, 1995.

9 Joaquín de Cea and Enrique Fernández. Transit assignment for congested public transport systems: An equilibrium model. *Transportation Science*, 27(2):133–147, 1993.

10 Guy Desaulniers and Mark D. Hickman. Public transit. *Transportation*, 14:69–127, 2007. `doi:10.1016/S0927-0507(06)14002-5`.

11 El-Ghazali Talbi. Metaheuristics for Bi-level Optimization. *Studies in Computational Intelligence*, 482, 2013.

12 José Fortuny-Amat and Bruce McCarl. A representation and economic interpretation of a two-level programming problem. *Journal of the Operational Research Society*, 32:783–792, 1981.

13 Ziyou Gao, Huijun Sun, and Lian Long Shan. A continuous equilibrium network design model and algorithm for transit systems. *Transportation Research B*, 38(3):235–250, 2004.

14 Lina P. Garcés, Antonio J. Conejo, Raquel García-Bertrand, and Rubén Romero. A bilevel approach to transmission expansion planning within a market environment. *IEEE Transactions on Power Systems*, 24(3):1513–1522, 2009. `doi:10.1109/TPWRS.2009.2021230`.

15 Guido Gentile, Michael Florian, Younes Hamdouch, Oded Cats, and Agostino Nuzzolo. *The Theory of Transit Assignment: Basic Modelling Frameworks*, pages 287–386. Springer International Publishing, Cham, 2016. `doi:10.1007/978-3-319-25082-3_6`.

16 Ricardo Giesen, Héctor Martínez, Antonio Mauttone, and María E. Urquhart. A method for solving the multi-objective transit frequency optimization problem. *Journal of Advanced Transportation*, 50, 2017. `doi:10.1002/atr.1461`.

17 Fred Glover. Tabu search part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

18 Marc Goerigk and Marie Schmidt. Line planning with user-optimal route choice. *European Journal of Operational Research*, 259:424–436, 2017.

19 Carola Leiva, Juan Carlos Muñoz, Ricardo Giesen, and Homero Larrain. Design of limited-stop services for an urban bus corridor with capacity constraints. *Transportation Research B*, 44(10):1186–1201, 2010.

20 Héctor Martínez, Antonio Mauttone, and María E. Urquhart. Frequency optimization in public transportation systems: formulations and metaheuristic approach. *European Journal of Operational Research*, 236(1):27–36, 2014. `doi:10.1016/j.ejor.2013.11.007`.

21 Antonio Mauttone and María E. Urquhart. A multi-objective metaheuristic approach for the transit network design problem. *Public Transport*, 1(4):253–273, 2009.

22 Sang Nguyen and Stefano Pallottino. Equilibrium traffic assignment for large scale transit networks. *European Journal of Operational Research*, 37(2):176–186, 1988.

23 Francisco Ruisánchez, Luigi dell'Olio, and Angel Ibeas. Design of a tabu search algorithm for assigning optimal bus sizes and frequencies in urban transport services. *Journal of Advanced Transportation*, 46(4):366–377, 2012.

24 Heinz Spiess and Michael Florian. Optimal strategies: a new assignment model for transit networks. *Transportation Research B*, 23(2):83–102, 1989.

Cost-Minimal Public Transport Planning

Julius Pätzold
University of Goettingen
Lotzestr. 16-18, 37083 Göttingen, Germany
j.paetzold@math.uni-goettingen.de

Alexander Schiewe
University of Goettingen
Lotzestr. 16-18, 37083 Göttingen, Germany
a.schiewe@math.uni-goettingen.de

Anita Schöbel
University of Goettingen
Lotzestr. 16-18, 37083 Göttingen, Germany
schoebel@math.uni-goettingen.de

Abstract

In this paper we discuss what a cost-optimal public transport plan looks like, i.e., we determine a line plan, a timetable and a vehicle schedule which can be operated with minimal costs while, at the same time, allowing all passengers to travel between their origins and destinations. We are hereby interested in an exact solution of the *integrated* problem. In contrast to a passenger-optimal transport plan, in which there is a direct connection for every origin-destination pair, the structure or model for determining a cost-optimal transport plan is not obvious and has not been researched so far.

We present three models which differ with respect to the structures we are looking for. If lines are directed and may contain circles, we prove that a cost-optimal schedule can (under weak assumptions) already be obtained by first distributing the passengers in a cost-optimal way. We are able to streamline the resulting integer program such that it can be applied to real-world instances. The model gives bounds for the general case. In the second model we look for lines operated in both directions, but allow only simplified vehicle schedules. This model then yields stronger bounds than the first one. Our most realistic model looks for lines operated in both directions, and allows all structures for the vehicle schedules. This model, however, is only computable for small instances. Finally, the results of the three models and their respective bounds are compared experimentally.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics, Applied computing → Transportation

Keywords and phrases Public Transport Planning, Integer Optimization, Line Planning, Vehicle Scheduling

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.8

Funding This work was partially supported by DFG under SCHO 1140/8-1 and by the Simulation Science Center Clausthal/Göttingen.

1 Introduction

Public transport planning is a challenging task since it consists of several stages: network design, line planning, timetabling, vehicle- and crew scheduling. In this paper we look for a line plan in combination with a timetable and a vehicle schedule, i.e., a *public transport plan*. Apart from the different subproblems that need to be solved in an integrated way, there are also different objectives to be considered. A public transport plan should be passenger-friendly (mostly reflected by a short traveling time for the passengers) but also have low operating costs. For individual planning stages such as line planning or vehicle scheduling there exist models and algorithms but finding an integrated solution to this multi-stage problem is more challenging. Surprisingly, only few papers even *evaluate* both cost and traveling time for integrated public transport plans. A first approach in which line plans, timetables and vehicle schedules have been evaluated together under different criteria has been given in [16]. More recently, [13] proposes to measure the costs and the traveling time, and evaluates public transport plans under these criteria (cf. Figure 4).

The goal of integrated planning is to find the set of Pareto solutions with respect to costs and traveling time and then to choose a solution from this set that is affordable and good for the passengers. From an academic point of view it is interesting to find theoretical bounds on the two objective function values of the Pareto solutions, i.e. finding the best achievable traveling time for the passengers, and finding the minimal costs (under the condition that all passengers can be transported). The former problem can be solved by a *taxi-solution*, providing a direct and fast connection for each origin-destination pair. Nevertheless, what a cost-optimal transportation plan would look like has not been studied so far and does not seem to be obvious. Given a line pool, [4] determine a line plan such that all origin-destination pairs can travel. The costs for the lines, however, are only approximated and not determined by the vehicle schedule. Furthermore, capacities are neglected. In contrast to this work, we now take an integrated point of view and propose models for finding cost-optimal public transport plans, including lines, timetables, and vehicle schedules.

In this paper we propose models for finding cost-optimal public transport plans. More precisely, we assume that the public transport network with its stops and direct connections is given, and that the passengers' demand is known in form of an origin-destination (OD) matrix. For a homogeneous fleet with a given capacity for each vehicle we then design a line plan, a timetable, and a vehicle schedule under the constraint that all passengers can be transported, i.e., for each passenger there exists a possible (maybe non-optimal) connection from their origin to their destination such that none of the vehicles is overloaded. We aim at solving the integrated system exactly, meaning that we do not provide iterative heuristics as in [7, 34, 37] or a sequential approach as the one in [25]. This becomes possible because we neglect the traveling time and only look at the costs meaning that the computationally hard step of timetabling becomes irrelevant.

For the single planning stages line planning, timetabling, and vehicle scheduling, models and algorithms are well-researched. For line planning, cost-oriented models (e.g. [10, 18, 38]) and passenger-oriented models (e.g. [2, 8, 35]) are known, see [33] for a survey. (Periodic) timetabling focuses on the passengers and is the hardest of the three problems. Exact approaches to this problem can be found in [36, 23, 29, 19] and heuristics in [24, 17, 26] and references therein. Integrating the passengers' routes in timetabling is an ongoing problem, see [3, 32, 15]. For vehicle scheduling we refer to the survey in [6].

2 A cost-optimal LTS-plan

In this section we formally describe what a feasible public transport plan (*LTS-plan*), consisting of a *line plan* (L), a *timetable* (T), and a *vehicle schedule* (S), is and how its quality can be evaluated. We restrict ourselves to periodic LTS-plans (including the vehicle scheduling) in this paper.

▶ **Notation 1.** The following input data is needed:

- a public transport network PTN $= (V, E)$ with a set of stops V and direct connections E between them,
- for every edge $e \in E$:
  - a length (in kilometers) length_e,
  - a lower bound on the traveling time along the edge L_e^{drive},
- a lower bound L^{wait} for the time vehicles have to wait at every stop,
- a minimal turnaround time for vehicles L^{turn}, denoting the minimal time a vehicle has to wait at the end of a line. We assume that $L^{\text{wait}} \leq L^{\text{turn}}$.
- an OD-matrix W with entries W_{uv} for each pair of stops $u, v \in V$, denoting how many passengers want to travel from an origin u to the destination v in a representative time period. A pair of stations $u, v \in V$ with $W_{uv} > 0$ is called an OD-pair.
- a capacity Cap being the maximal number of passengers each vehicle can transport,
- cost parameters
  - c_{time} costs per hour for a vehicle driving,
  - c_{length} costs per kilometer for a vehicle driving.

We assume that the fixed costs (cost of a vehicle, administration, etc.) are included in the costs per hour and the costs per kilometer, as is often done in practice.
With this input data we then look for an LTS-plan, whose objects are described next.

Line plan L

A *line* is a path through the PTN. A *line plan* is a set of lines $\mathcal{L}$, each of them operated once in the planning period (often an hour). A line plan is *feasible* if every passenger can be transported, i.e., if for every OD-pair (u, v) there exist

- a set of directed paths P_{uv} from u to v, $P_{\text{all}} = \bigcup_{u,v \in V} P_{uv}$, and
- weights w_p for each path $p \in P_{uv}$

such that $\sum_{p \in P_{uv}} w_p = W_{uv}$ and such that for every edge e it holds that

$$\sum_{p \in P_{\text{all}}: e \in p} w_p \leq \text{Cap} \cdot |\{l \in \mathcal{L} : e \in l\}|. \tag{1}$$

Note that feasibility does not require the paths P_{uv} to be good paths for the passengers, but only that all passengers can be transported.
We furthermore assume that lines are simple paths and that every line is operated in both directions. We do not forbid identical lines, i.e., there may be multiple lines with the same path. In our setting we allow any path as a possible line (as also done in [2]) in contrast to many papers which require a line pool of limited size.

Timetable T

Given a set of lines $\mathcal{L}$, a timetable assigns a time to every departure and arrival of each line at its stops. Determining a (periodic) timetable is the hardest of the three problems line planning, timetabling, and vehicle scheduling, and even finding a feasible timetable that respects the upper and lower bounds on driving, waiting, transfer and turnaround activities is intractable. Since we neglect the passengers, no upper bounds on transfer activities are needed, and hence a feasible timetable exists for every possible line plan $\mathcal{L}$ (since the timetable for each line can then be determined separately.). Since we are only interested in minimizing the costs we furthermore need not care about optimizing the traveling time of the passengers, meaning that any feasible timetable is sufficient. More precisely, we can neglect the timetabling as a separate planning stage in cost-optimal planning and simply use the arrival and departure times which are determined by the vehicle schedule.

Vehicle schedule S

Given a line plan a *vehicle schedule* determines the number of vehicles and the exact routes of the vehicles for operating the lines. We construct a set of *trips* $\mathcal{L}'$ which contains two directed lines for every (undirected) line $l \in \mathcal{L}$, one in forward and the other in backward direction.
A route of a vehicle is given by the sequence of (directed) lines it passes,

$$r = (l'_1, \dots, l'_k),\ l'_i \in \mathcal{L}'$$

whereby we require that the $l'_i, i = 1, \dots, k$ are pairwise distinct. We assume that after having taken the last trip l'_k in a route, the vehicle starts again with l'_1.

This sequence r is interpreted as follows: A vehicle starts with operating line l'_1 at some point in time, x. At the end of line l'_1 it drives to the start point of line l'_2, operates this line, and so on. At the end of line l'_k the vehicle returns to the start point of l'_1 and starts from the beginning of the next time period. In order to ensure the required periodicity of the schedule, the vehicle needs to start after an integer multiple of the period T, i.e., at a time $y = x + d_r \cdot T$, whereby the integer d_r is the number of periods needed for a complete operation of the route r.

A vehicle schedule thus consists of a set of routes $\mathcal{R}$. It is *feasible* if each directed line in $\mathcal{L}'$ is contained in exactly one route, i.e., if

$$|\{r \in \mathcal{R} : l' \in r\}| = 1 \quad \forall l' \in \mathcal{L}'. \tag{2}$$

With these assumptions in place we can then define what an *LTS-plan* is.

▶ **Definition 2.** An LTS-plan is a tuple $(\mathcal{L}, \mathcal{R})$, such that

- $\mathcal{L}$ is a feasible line plan, i.e., it satisfies (1),
- $\mathcal{R}$ is a feasible vehicle schedule for the directed lines $\mathcal{L}'$, i.e., it satisfies (2).

Costs of an LTS-plan

The costs of an LTS-plan are given by the distance driven by all vehicles and its total duration. Since we compute a periodic schedule, we consider the costs per planning period T.

A vehicle route r consists of (directed) lines $l' \in \mathcal{L}'$. Hence, we first determine time and duration of a line l', namely,

$$\begin{aligned} \text{length}_l &= \sum_{e \in l} \text{length}_e && (3) \\ \text{dur}_l &= (|l|-1)L^{\text{wait}} + \sum_{e \in l} L_e^{\text{drive}}, && (4) \end{aligned}$$

where $|l| := \{e \in E | e \in l\}$ and (4) uses the fact that it is always cheaper to operate a line as fast as possible. For the empty rides between a pair of lines l'_1 and l'_2 we can use the PTN to determine the parameters

$$\begin{aligned} \text{length}_{l'_1,l'_2} &= \text{length when driving from the last station of } l'_1 \text{ to the first station of } l'_2 \\ \text{time}_{l'_1,l'_2} &= \text{time for driving from the last station of line } l'_1 \text{ to the first station of } l'_2 \end{aligned}$$

The minimum turnaround time (usually accounting for a driver's break) has to be added to the duration of an empty ride. This yields

$$\text{dur}_{l'_1,l'_2} = L^{\text{turn}} + \text{time}_{l'_1,l'_2}. \qquad (5)$$

The number of kilometers a given LTS-plan covers is determined by summing up the kilometers of each single route, i.e.,

$$\begin{aligned} \text{length}(\mathcal{L},\mathcal{R}) &= \sum_{l' \in \mathcal{L}'} \text{length}_{l'} + \sum_{r=(l'_1,\dots,l'_{k_r}) \in \mathcal{R}} \sum_{i=1}^{k_r} \text{length}_{l'_i,l'_{i+1}} \\ &= \sum_{l \in \mathcal{L}} 2 \cdot \text{length}_l + \sum_{r=(l'_1,\dots,l'_{k_r}) \in \mathcal{R}} \sum_{i=1}^{k_r} \text{length}_{l'_i,l'_{i+1}} \end{aligned}$$

with $l'_{k_r+1} := l'_1$. The duration of a route $r = (l'_1, \dots, l'_{k_r}) \in \mathcal{R}$ is measured by the number of time periods dur_r needed. This can be formally computed by

$$\text{dur}_r = \left\lceil \sum_{i=1}^{k_r} \text{dur}_{l'_i} + \text{dur}_{l'_i,l'_{i+1}} \right\rceil_T \qquad (6)$$

with $\lceil a \rceil_T := \min\{n \in \mathbb{N} | n \cdot T \geq a\}$ for any $a \in \mathbb{R}$ and $l'_{k_r+1} := l'_1$. The overall duration is hence given as

$$\text{dur}(\mathcal{L},\mathcal{R}) = \sum_{r \in \mathcal{R}} \text{dur}_r. \qquad (7)$$

Finally, the cost function is defined as

$$g(\mathcal{L},\mathcal{R}) := c_{\text{time}} \cdot \text{dur}(\mathcal{L},\mathcal{R}) + c_{\text{length}} \cdot \text{length}(\mathcal{L},\mathcal{R}). \qquad (8)$$

Note that the number of required vehicles is determined by the total duration, i.e., by $\frac{dur(\mathcal{L},\mathcal{R})}{T}$. The fixed costs per vehicle γ can be included by adding $\frac{\gamma}{T}$ to c_{time}. Since this does not change the structure of the cost function we assume the vehicle costs to be already included in c_{time}.
The cost function defined above allows us to define the optimization problem we are concerned with in this paper.

Problem (cost-opt LTS): Given the input data from Notation 1, find a feasible LTS-plan $(\mathcal{L},\mathcal{R})$ with minimal costs $g(\mathcal{L},\mathcal{R})$.

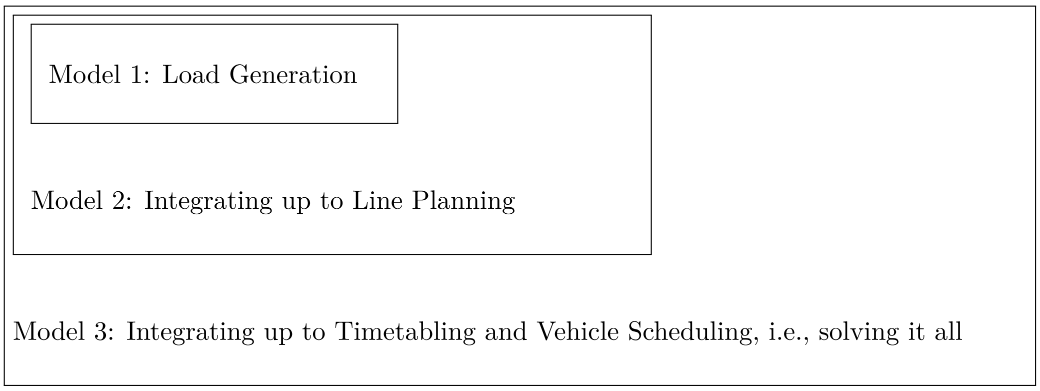

Model 1: Load Generation
Model 2: Integrating up to Line Planning
Model 3: Integrating up to Timetabling and Vehicle Scheduling, i.e., solving it all

Figure 1 Three proposed models for solving (cost-opt LTS).

Traditionally, calculating an LTS-plan consists of solving a series of problems in a sequential order, as can be seen in [9, 11, 21]. A sequential approach, however, is flawed, since the costs are mainly determined by the vehicle schedule, which constitutes the last step of the planning process. Nevertheless, this has been tackled in [25] by a heuristic approach. The aim of our paper, however, is to find the exact cost minimum of the integrated problem. In order to address this issue we present three different models for minimizing the costs of the resulting LTS-plan (see Figure 1).
The first model aims at distributing the OD-pairs in a cost-optimal way (called *load generation*). Although it only concerns this very first step we can show that this determines the minimal costs of an integrated LTS-plan under certain conditions. The second model integrates load generation and line planning, minimizing a cost function that approximates (now in greater detail) the costs of a resulting LTS-plan. Finally, the third model presents an IP formulation for integrating load generation, line planning, timetabling, and vehicle scheduling; it hence provides an exact model for (cost-opt LTS).

3 Model 1: Creating a cost-efficient load

Line planning is often decomposed into two steps. In the first step, all OD-pairs (u, v) are routed through the PTN resulting in paths P_{uv}, $P_{\text{all}} = \bigcup_{u,v \in V} P_{uv}$, and weights w_p for every path $p \in P_{uv}$ (with $\sum_{p \in P_{uv}} w_p = W_{uv}$). This data is then used to define the *loads*

$$f_e^{\min} = \left\lceil \sum_{p \in P_{\text{all}}: e \in p} w_p \cdot \frac{1}{\text{Cap}} \right\rceil$$

specifying how often an edge $e \in E$ in the PTN has at least to be served by some vehicle. In the second step, the line planning problem is solved using these minimal frequencies.
Normally the $f_e^{\min}$ are calculated assuming that all passengers travel on their shortest path in the PTN to their destination. Since we are interested in finding a cost-minimal LTS-plan, we do not want to work with that assumption. In our system we require just enough capacities so that every passenger has some possibility to travel to their destination. We use this insight to find a load that eventually even leads to a cost-minimal LTS-plan.
Of course, in this early planning stage we do not yet have all information to exactly determine the costs of the resulting LTS-plan, since they depend on the line plan and the vehicle schedule. Nevertheless, we can already approximate the costs with the following model.

▶ **Model 1.** Given the input data from Notation 1, calculate a load (i.e., $f_e^{\min}$ for all $e \in E$) that aims at minimizing the cost of an LTS-plan.

$$\min \quad c_{\text{time}} \cdot \text{dur} \cdot T + c_{\text{length}} \sum_{e \in E} 2 \cdot \text{length}_e \cdot f_e^{\min} \tag{9}$$

$$\text{s.t.} \quad \sum_{e \in E} 2 f_e^{\min}(L_e^{\text{drive}} + L^{\text{wait}}) \leq T \cdot \text{dur} \tag{10}$$

$$\sum_{u \in V} f_{(i,j),u} \leq f_e^{\min} \cdot \text{Cap} \quad \forall i,j \in V \text{ with } \{i,j\} \in E \tag{11}$$

$$\sum_{i \in V: \{i,v\} \in E} f_{(i,v),u} = W_{uv} + \sum_{i \in V: \{v,i\} \in E} f_{(v,i),u} \quad \forall u \in V \quad \forall v \in V \backslash \{u\} \tag{12}$$

$$\sum_{i \in V: \{u,i\} \in E} f_{(u,i),u} = \sum_{v \in V} W_{uv} \quad \forall u \in V \tag{13}$$

Variables:

- $f_{(i,j),u}$ – number of passengers starting from stop $u \in V$ traveling on arc (i,j) for some $i,j \in V$ with $\{i,j\} \in E$ (non-negative, continuous)
- $f_e^{\min}$ – how often edge e has to be covered (integer)
- dur – total duration (counted in periods) (integer)

In this model we define from every stop $u \in V$ in the PTN some passenger flow going to all destinations $v \in V$. In order not to mix up passengers starting from different stations we accordingly have to define $|V|$ different flows. The constraints (12) and (13) describe the flow conservation constraints. In order to restrict the number of passengers traveling on a certain edge in the network we defined the capacity constraints (11). Note that the flow variables $f_{(i,j),u}$ for $u \in V$ are defined on directed edges (i,j) whereas the minimal frequencies $f_e^{\min}$ are defined on undirected edges $\{i,j\} = e \in E$. Finally constraint (10) rounds the minimal duration up to the next multiple of a time period T and the objective function gives the costs which are needed in the best case, namely for a vehicle schedule without any empty ride and as few time loss (through the periodicity) as possible.

The following theorem shows that Model 1 is indeed an approximation of (cost-opt LTS), as its optimal solution yields a lower bound.

▶ **Theorem 3.** *The optimal objective value of Model 1 is a lower bound on the optimal objective value of (cost-opt LTS).*

Proof. See Appendix B. ◀

For large problem instances a speed-up of the solution process is possible by adding the following valid inequalities to Model 1.

▶ **Lemma 4.** *Let (X, Y) be some cut, i.e., some disjoint partition of all nodes in the PTN with $E_{cut} = \{\{i,j\} = e \in E | i \in X \text{ and } j \in Y\}$ being all cut edges. Then it holds that*

$$\sum_{u \in X} \sum_{v \in Y} W_{uv} \leq Cap \cdot \sum_{e \in E_{cut}} f_e^{\min}.$$

Proof. See Appendix B. ◀

In the computational experiments (Section 6) we investigated adding these valid inequalities, which resulted in an improvement of the runtime of up to 50%.
Model 1 does not only yield some lower bound, but we can even construct an optimal solution to (cost-opt LTS) if a particular assumption is met.

▶ **Theorem 5.** *Let $L^{\text{wait}} = L^{\text{turn}}$ and let the graph $G = (V, \bar{E})$ with $\bar{E} = \{e \in E : f_e^{\min} > 0\}$ for an optimal solution $f_e^{\min}$ of Model 1 be connected. Then the optimal objective of Model 1 is equal to the optimal objective of (cost-opt LTS).*

Proof. For every solution to Model 1, i.e., for some feasible $f_e^{\min}$ with $e \in E$, we can construct some feasible solution $(\mathcal{L}, \mathcal{R})$ to (cost-opt LTS) as follows: We define the line plan $\mathcal{L}$ that contains for each edge $e \in E$ exactly $f_e^{\min}$ lines containing exactly this one edge e, i.e., $\mathcal{L} := \{e^1, \dots, e^{f_e^{\min}} : e \in E\}$. Since $f_e^{\min} = |\{l \in \mathcal{L} | e \in l\}|$ and $f_e^{\min}$ admits a feasible load, the line plan $\mathcal{L}$ is feasible.
For this line plan we now generate a vehicle schedule $\mathcal{R}$ that consists of only one large route. To this end, we consider the resulting set of directed lines $\mathcal{L}'$

$$\mathcal{L}' = \left\{(i,j)^1, \dots, (i,j)^{f_e^{\min}}, (j,i)^1, \dots, (j,i)^{f_e^{\min}} : e = \{i,j\} \in E\right\}$$

which contains $f_e^{\min}$ copies of both directions of every edge $e \in E$. This is a set of directed edges which creates a directed multigraph $(V, \mathcal{L}')$. Due to the assumption in the theorem, this graph is strongly connected and every node in $(V, \mathcal{L}')$ has the same indegree as outdegree. Hence we can find an Eulerian Cycle on it (see e.g. [12]). This means that we can form a route containing all directed lines $r = (l'_1, \dots, l'_k)$ (with $|r| = |\mathcal{L}'|$) such that $\text{length}_{l'_i, l'_{i+1}} = 0$ and $\text{time}_{l'_i, l'_{i+1}} = 0$. So we set the vehicle schedule $\mathcal{R} = \{r\}$ to contain exactly this route r. We hence have constructed some solution $(\mathcal{L}, \mathcal{R})$ to (cost-opt LTS) with

$$\begin{aligned}\text{length}(\mathcal{L}, \mathcal{R}) &= \sum_{l \in \mathcal{L}'} \text{length}_l + \sum_{r = (l'_1, \dots, l'_{k_r}) \in \mathcal{R}} \sum_{i=1}^{k_r} \underbrace{\text{length}_{l'_i, l'_{i+1}}}_{=0} \\ &= \sum_{l \in \mathcal{L}} 2 \cdot \text{length}_l \underbrace{=}_{f_e^{\min} = \{e \in \mathcal{L} | e \in l\}} \sum_{e \in E} 2\text{length}_e f_e^{\min}\end{aligned}$$

and

$$\begin{aligned}\text{dur}(\mathcal{L}, \mathcal{R}) = \sum_{r \in R} \text{dur}_r &\underbrace{=}_{|\mathcal{R}|=1} \left\lceil \sum_{l \in \mathcal{L}'} (\text{dur}_l + L^{\text{turn}}) \right\rceil_T \\ &\underbrace{=}_{f_e^{\min} = \{e \in \mathcal{L} | e \in l\}} \left\lceil \sum_{e \in E} 2 f_e^{\min} (L_e^{\text{drive}} + L^{\text{turn}}) \right\rceil_T \\ &\underbrace{=}_{L^{\text{turn}} = L^{\text{wait}}} \left\lceil \sum_{e \in E} 2 f_e^{\min} (L_e^{\text{drive}} + L^{\text{wait}}) \right\rceil_T = \text{dur} \cdot T.\end{aligned}$$

Hence, for every solution to Model 1 we can construct a solution $(\mathcal{L}, \mathcal{R})$ to (cost-opt LTS) such that $g(\mathcal{L}, \mathcal{R}) = c_{\text{time}} \text{dur} \cdot T + c_{\text{length}} \sum_{e \in E} 2\text{length}_e \cdot f_e^{\min}$. Together with Theorem 3 $(\mathcal{L}, \mathcal{R})$ is optimal for (cost-opt LTS)and hence Model 1 has the same objective value as (cost-opt LTS). ◀

In case the assumption $L^{\text{wait}} = L^{\text{turn}}$ does not hold, we still get a feasible solution and therefore an upper bound for (cost-opt LTS), when we slightly modify Model 1.

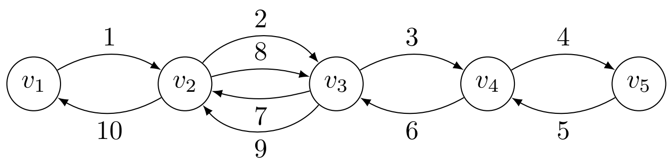


Figure 2 Solution of Model 1 for Example 9.

▶ **Definition 6.** We define an adjusted version of Model 1, where L^{wait} is replaced by L^{turn} in constraint (10), to be Model 1*.

▶ **Corollary 7.** *The solution $(\mathcal{L}, \mathcal{R})$ constructed in the proof of Theorem 5 is an upper bound for (cost-opt LTS) and can be found by solving Model 1*.*

If we allow that lines do not have to be bidirectional and simple paths in the PTN, we can always obtain an optimal solution to (cost-opt LTS) by just solving Model 1. This can be done by converting the Eulerian Cycle constructed the proof of Theorem 5 into one big line.

▶ **Corollary 8.** *Let $L^{\text{wait}} \leq L^{\text{turn}}$. Then the optimal objective value of Model 1 is equal to the optimal objective of (cost-opt LTS) if we allow directed and non-simple lines.*

This, of course, may lead to non-practical lines, as can be seen in the following example.

▶ **Example 9.** We examine the solution provided by Corollary 8 on a small example. Consider the PTN given in Figure 2, with Cap passenger traveling from v_1 to v_5 and 1 passenger traveling from v_2 to v_3. Then the solution provided by Model 1 is given by lower bounds of $[1, 2, 1, 1]$ and the vehicle schedule of Corollary 8 is depicted in Figure 2, where the edges are numbered in the order of their usage. As can be seen here, the resulting line structure is not suitable for a practical public transport system, since it contains a cycle.

4 Model 2: Integrating load generation and line planning

Although we can already find a cost-optimal solution using Model 1, this only works in the special case of $L^{\text{wait}} = L^{\text{turn}}$. We have seen that for $L^{\text{wait}} < L^{\text{turn}}$ the resulting line plan consists of directed lines (without their symmetric counterparts) and the lines may contain circles. We therefore further explore the next steps for obtaining an LTS-plan in which the lines satisfy the usual requirements. To this end, we combine the load generation of Model 1 with line planning to improve the approximation of the cost objective of the overall LTS-plan. This idea is approached by the following model.

▶ **Model 2.** Given the input data from Notation 1, calculate a load $f_e^{\min}$ and a line plan $\mathcal{L}$ that aim at minimizing the costs of an LTS-plan.

$$\min \quad c_{\text{time}} \cdot \text{dur} \cdot T + c_{\text{length}} \sum_{l=1}^{L} \sum_{e\in E} 2x_{e,l}\text{length}_e \tag{14}$$

$$\text{s.t.} \quad (11) - (13)$$

$$\sum_{l=1}^{L} \left(2z_l(L^{\text{turn}} - L^{\text{wait}}) + \sum_{e\in E} 2(L_e^{\text{drive}} + L^{\text{wait}}) \cdot x_{e,l} \right) \leq \text{dur} \cdot T \tag{15}$$

$$\sum_{l=1}^{L} x_{e,l} \geq f_e^{\min} \quad \forall e \in E \tag{16}$$

$$x_{e,l} \leq z_l \quad \forall e \in E\ \forall l \in [L] \tag{17}$$

$$\sum_{e\in E} x_{e,l} \geq z_l \quad \forall l \in [L] \tag{18}$$

$$\sum_{e\in E: s\in e} x_{e,l} \leq 2 \quad \forall s \in V\ \forall l \in [L] \tag{19}$$

$$2x_{e,l} \leq y_{i,l} + y_{j,l} \quad \forall l \in [L]\ \forall (i,j) = e \in E \tag{20}$$

$$\sum_{s\in V} y_{s,l} = \sum_{e\in E} x_{e,l} + z_l \quad \forall l \in [L] \tag{21}$$

$$\sum_{(i,j)=e\in E: i\in C \text{ and } j\in C} x_{e,l} \leq |C| - 1 \qquad \forall \text{ circles } C \subseteq E\ \forall l \in [L] \tag{22}$$

Coefficients:

- L – maximal possible number of lines (integer) and $[L] := \{1, ..., L\}$.

Variables:

- z_l – is 1 iff line l is non-empty. (binary)
- $y_{s,l}$ – is 1 iff stop s is contained in line l. (binary)
- $x_{e,l}$ – is 1 iff edge e is contained in line l. (binary)
- dur – total duration of all lines (counted in periods) (integer)
- $f_e^{\min}$ – as in Model 1, including the variables $f_{e,u}$ and constraints (11) - (13) from Model 1.

This model finds some feasible line plan. First the z_l-variables determine if line number l is a line or empty. Constraint (17) and (18) ensure this. Now we need for every index l that for every stop of some line there are at most two incident edges (constraint (19)). This ensures that the $x_{e,l}$ variables form circles or paths. To ensure that they form only one connected path we could consider them as flow variables. Here, we decided to add y-variables for every visited stop and count the number of stops that a line visits. The y-variables are set to one for the incident nodes of all edges the line visits in (20). We then can ensure that there is some connected path by requiring that there exists exactly one more stop than edges in a line in constraint (21). Finally we need to rule out subtours which is done by constraint (22) (As usual they are added by constraint generation procedures). The variables $f_e^{\min}$ taken from Model 1 help us to determine feasibility of the line plan, which is done by constraint (16). Finally we round the duration up to the next multiple of a time period, which is done by (15).

The objective function is again a lower bound on the exact costs of an LTS-plan. This is shown in the next theorem.

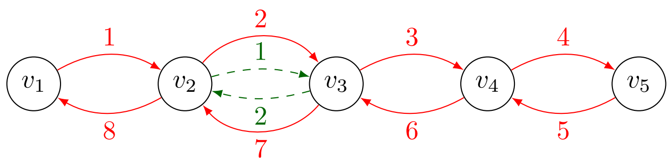


Figure 3 Solution of Model 2.

▶ **Theorem 10.** *The optimal objective value of Model 2 is a lower bound on the optimal objective value of (cost-opt LTS) and an upper bound to the optimal objective value of Model 1.*

Proof. See Appendix B. ◀

We can again construct a feasible solution for (cost-opt LTS) from the solution of Model 2 in the case that we are only interested in line-pure vehicle schedules. In such schedules, every vehicle serves the same line, alternating between its forward and its backward direction. More formally:

▶ **Definition 11.** A solution to (cost-opt LTS) is called line-pure if $\mathcal{R} = \{r_l : l \in \mathcal{L}\}$, with $r_l = (l^+, l^-)$ being the route that contains only the forward and backward direction of line $l \in \mathcal{L}$.

We now show that the following slight modification of Model 2 can find a cost-optimal LTS-plan under the restriction that only line-pure vehicle schedules are allowed.

▶ **Definition 12.** Consider Model 2 and replace constraint (15) by

$$2z_l(L_e^{\text{turn}} - L_e^{\text{wait}}) + \sum_{e \in E} 2(L_e^{\text{drive}} + L_e^{\text{wait}}) \cdot x_{e,l} \leq d_l \cdot T \quad \forall l \in [L] \tag{23}$$

$$\sum_{l=1}^{L} d_l = dur \tag{24}$$

with integer variables $d_l \in \mathbb{N}$. We call this modified version Model 2*.

Restricting ourselves to a special structure of the vehicle schedules, we are still able to obtain the optimal solution to (cost-opt LTS) (under some assumptions) by simply considering loads and the lines. This is the main result of this section.

▶ **Theorem 13.** *An optimal solution to Model 2* solves (cost-opt LTS) under the restriction that only line-pure vehicle schedules are allowed.*

Proof. See Appendix B. ◀

For the general case of (cost-opt LTS), Model 2* still finds a feasible solution and therefore provides an upper bound to (cost-opt LTS).

▶ **Corollary 14.** *The optimal objective value to Model 2* imposes an upper bound on the optimal objective value of (cost-opt LTS).*

▶ **Example 15.** We continue Example 9 and now consider the solution constructed in Theorem 10. These now provide simple lines, resulting in the line-pure vehicle schedule depicted in Figure 3, improving on the line structure of Example 9. The first line is depicted in red, the second is dashed in green. The lines here look much more reasonable for practical implementation than the solution which was obtained by Model 1*.

5 Model 3: Integrating timetabling and vehicle scheduling

In Model 1 and Model 2 we did not consider all subproblems of (cost-opt LTS), especially we did not include a proper vehicle scheduling. With the following model we want to overcome this issue and formulate the whole problem in an integrated way.

To formulate the integrated model, we need a notation for the event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ (see, e.g., [19, 21, 23, 27, 28]). The set of events $\mathcal{E}$ consists of all departures and all arrivals of all lines at all stops and two additional OD-events $((u, \text{dep}), (u, \text{arr}))$ per stop u for passengers to enter and leave the network, denoted as $\mathcal{E}_{OD}$. The set $\mathcal{A}$ connects the events by driving, waiting and transfer activities. The OD-events are connected to each departure event of the corresponding stop using OD-activities ($\mathcal{A}_{OD}$). Using this, we can now formulate the integrated model. Let further denote with $\mathcal{A}_{l'}$ all activities in $\mathcal{A} \setminus \mathcal{A}_{OD}$ that are included in a directed line $l' \in \mathcal{L}'$.

▶ **Model 3.** Given the input data from Notation 1, find a feasible LTS-plan $(\mathcal{L}, \mathcal{R})$ with minimal costs, i.e., minimizing $g(\mathcal{L}, \mathcal{R})$.

$$\min \sum_{v \in V} \text{cost}_v$$

$$\text{s.t.} \quad \text{dur}_r \geq \frac{1}{T} \cdot \sum_{l' \in \mathcal{L}'} x_{l',r} \cdot \text{dur}_l + \sum_{l'_1, l'_2 \in \mathcal{L}'} x_{(l'_1, l'_2), r} \cdot \text{dur}_{l'_1, l'_2} \quad \forall r \in [R] \tag{25}$$

$$\text{length}_r \geq \sum_{l' \in \mathcal{L}'} x_{l',r} \cdot \text{length}_l + \sum_{l'_1, l'_2 \in \mathcal{L}'} x_{(l'_1, l'_2), r} \cdot \text{length}_{l'_1, l'_2} \quad \forall r \in [R] \tag{26}$$

$$\text{cost}_r \geq c_{\text{length}} \cdot \text{length}_r + c_{\text{time}} \cdot \text{dur}_r \quad \forall r \in [R] \tag{27}$$

$$\sum_{l^* \in \mathcal{L}'} x_{(l', l^*), r} = x_{l',r} = \sum_{l^* \in \mathcal{L}'} x_{(l^*, l'), r} \quad \forall l' \in \mathcal{L}', \forall r \in [R] \tag{28}$$

$$\sum_{r \in R} x_{l',r} = \sum_{v \in V} x_{b(l'),r} \quad \forall l' \in \mathcal{L}' \tag{29}$$

$$\text{Cap} \cdot \sum_{r \in R} x_{l',r} \geq \sum_{u,v \in V} f_{a,(u,v)} \quad \forall l' \in \mathcal{L}', \forall a \in \mathcal{A}_{l'} \tag{30}$$

$$\sum_{\substack{i \in \mathcal{E} \\ (i,j) = a \in \mathcal{A}}} f_{a,(u,v)} = \sum_{\substack{i \in \mathcal{E}: \\ (j,i) \in \mathcal{A}_t}} f_{a,(u,v)} \quad \forall p \in P, \forall j \in \mathcal{E} \setminus \mathcal{E}_{OD} \tag{31}$$

$$\sum_{\substack{i \in \mathcal{E}: \\ (i,j) = a \in \mathcal{A}_{\text{OD}}}} f_{a,(u,v)} = W_{uv} \quad \forall u, v \in V, \forall j = (v, \text{arr}) \in \mathcal{E}_{OD} \tag{32}$$

$$\sum_{\substack{i \in \mathcal{E}: \\ (j,i) = a \in \mathcal{A}_{\text{OD}}}} f_{a,(u,v)} = W_{uv} \quad \forall u, v \in V, \forall j = (u, \text{dep}) \in \mathcal{E}_{OD} \tag{33}$$

$$\sum_{(l'_1, l'_2) \in U'} x_{(l'_1, l'_2), r} \leq |U'| - 1 \quad \forall U' \subsetneq \mathcal{L}' \times \mathcal{L}', \forall r \in [R] \tag{34}$$

$$\text{dur}_r \in \mathbb{N} \quad \forall r \in [R] \tag{35}$$

Coefficients:
- R: number of possible vehicle routes, we assume it to be sufficiently large
- $\mathcal{L}'$: the set of all possible directed lines in the network, $b(l')$ denotes the backwards direction for a directed line l', l is the corresponding undirected line.

Table 1 Properties of the examined datasets.

Instance	Nodes	Edges	Passengers
`Linear`	5	4	141
`Toy`	8	8	2622
`Grid`	25	40	2546
`Germany`	250	326	385868

Variables:

- $x_{l',r}$ – is 1 iff the directed line l' is part of route r
- $x_{(l'_1,l'_2),r}$: is 1 iff lines l'_1 and l'_2 are served directly after each other in route r
- cost_r – the costs of route r
- dur_r – the duration of route r
- length_r – the length of route r
- $f_{a,(u,v)}$ – the number of passenger traveling from u to v using activity a

This model finds a cost-optimal LTS-plan (i.e., line plan, timetable and vehicle schedules). The f variables determine the passenger flow, satisfying the classical flow conservation constraints ((31)-(33)) and creating coupling constraints for the vehicle routes r in (30), determined by the x-variables. The duration and length of the routes are determined in (25) and (26) and then combined in (27) to determine the costs. Of course, the vehicle routes need to satisfy flow conservation as well (see (28)). (34) are the subtour elimination constraints. Constraint (29) ensures that every line is served in both directions.
The model is too large to be solved for realistic instances. One possibility As can be seen in Section 6, the integrated problem cannot be solved even for instances of small size. This is due to its enormous number of variables including a trip for every possible line in the network. Nevertheless, Model 3 can be used if enough variables are fixed. We hence can combine it with Model 2 by fixing the lines in Model 3 to the optimal lines computed by Model 2. This means that we only need to consider the constraints (25)-(28) and (34), additionally guaranteeing that every trip in $\mathcal{L}'$ is covered exactly once. The result is a tractable model for medium-sized instances.
Other possibilities to reduce its size would be to start with a line pool of limited size (e.g. as generated in [14] or from Model 2) or to use column generation approaches as in [2].

6 Experiments

In the computational experiments we implemented the three proposed models with the open source library LinTim (see [1, 16, 31]) and tested them on four different datasets. These datasets are described in in Table 1 and depicted in Figure 5, Appendix A.
We implemented Model 1, Model 1*, Model 2, Model 2* and Model 3 using Gurobi 8.0 as MIP solver with default settings. We tested all implementations on a compute server (6 cores of Intel(R) Xeon(R) CPU X5650 @ 2.67GHz, 78 GB RAM) with a time limit of 3 hours per test case. For each model and each instance we considered two different cases: Either $L^{\text{turn}} = L^{\text{wait}}$ or $L^{\text{turn}} > L^{\text{wait}}$ to distinguish the cases where Model 1* is able to find an optimal solution and where it is not. We obtained the results depicted in Tables 2 and 3. A symbol $^\circ$ denotes that the problem has not been solved to optimality and hence only the best found upper or lower bound is presented.

Table 2 Objective values for the case of $L^{\text{turn}} = L^{\text{wait}}$.

Instance	Model 1		Model 2		Model 3	
	Model 1	Model 1*	Model 2	Model 2*	lb	ub
`Linear`	80	80	80	130	80	80
`Toy`	1424	1424	1424	1696	1270°	1460°
`Grid`	1034	1034	1034	1034	–	–
`Germany`	73321°	84694°	54148°	–	–	–

Table 3 Objective values for the case of $L^{\text{turn}} > L^{\text{wait}}$.

Instance	Model 1		Model 2		Model 3	
	Model 1	Model 1*	Model 2	Model 2*	lb	ub
`Linear`	80	130	130	130	130	130
`Toy`	1424	1474	1424	1696	1288°	1539°
`Grid`	1034	1134	1030°	1140	–	–
`Germany`	74462°	85612°	54148°	–	–	–

For each of the three models there exist two columns. The left column contains a lower bound to (cost-opt LTS), whereas the right column contains an upper bound, i.e., the objective value of the best found feasible solution.
We observe for Model 1 that in the case $L^{\text{turn}} = L^{\text{wait}}$ it almost always finds the optimal objective value within the specified time limit of 3 hours. Only in our biggest instance we cannot get an optimal solution within the time limit (we still have a gap of 13.7% here). For the case $L^{\text{turn}} > L^{\text{wait}}$ there exists a gap between the lower bound and upper bound of Model 1, but this model still obtains the best solutions.
Model 2 can solve the two smallest instances easily, but starts having trouble with the time limit for `Grid`. For `Germany` it is not able to find a feasible solution within the specified time limit. Regarding the solution quality, we see that the lower bound given by Model 2 is only in a single case sharper than the lower bound given by Model 1. On the other hand, the upper bounds found by Model 2* never have smaller objective values than Model 1*.
Model 3 is already on the toy instance not able to find an optimal solution within 3 hours. The obtained objective values for `Linear` and the bounds for `Toy` are consistent with the values given in Models 1 and 2. For the bigger instance, even the precomputation of the complete line pool for Model 3 was not possible anymore.
We illustrate our results on the dataset `Grid` (see [13, 30]). Solutions are evaluated by their costs and their traveling times. The solutions shown in Figure 4 are computed sequentially. We see that the sequential solutions with smallest costs are A4 (computed in [25]) and P5 (computed in [20].) The best possible costs of a feasible solution (computed by solving Model 1) is depicted as a red line and improves the costs by 23%. Note that Model 1 computes a solution with a periodic vehicle schedule, but as shown in [5] an aperiodic schedule would not improve the costs.
The traveling time of the cost-minimal solution is hard to evaluate: Using the best possible paths for the passengers as done for the other solutions in Figure 4 would lead to a traveling time of only 20.57. We did not depict this objective value in the figure since in this solution the passengers are far away from using the paths computed for them in Model 1 and hence the solution would have heavily overloaded vehicles.

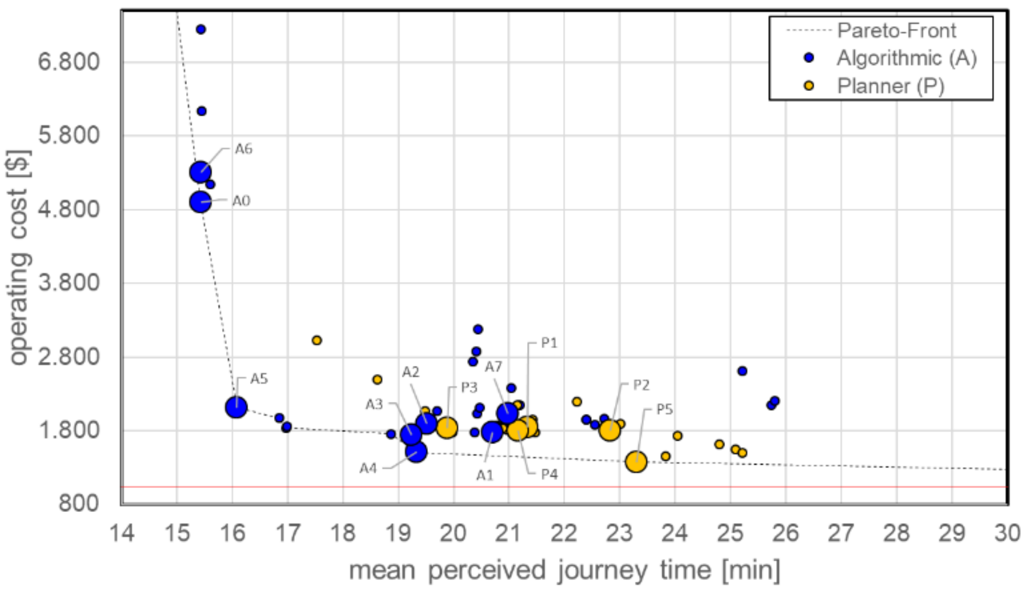


Figure 4 Multiple solutions for `Grid` (see [30]), evaluated by their cost per hour and traveling time (perceived journey time meaning traveling time plus a time penalty for every occurring transfer). With our models we were able to find a cost-minimal solution. Its objective value is depicted by a red line.

Table 4 Runtime improvements with Lemma 4 on `Grid` for $L^{\text{turn}} > L^{\text{wait}}$.

parameters	no cuts		cuts	
	Model 1	Model 1*	Model 1	Model 1*
Nodes explored	46557	26391	2398	3845
Runtime in sec	23.18	12.6	10.61	8.99

We finally investigate the influence of valid inequalities introduced in Lemma 4 on the runtime of Model 1. We restricted this investigation to `Grid`, since the runtime for the smallest two instances is already less than a second, and for `Germany` it is already non-trivial to determine "good" cuts of the network. For `Grid`, however, we took all horizontal and all vertical cuts of the network, whose PTN is depicted in Figure 5, into the model. With this improvement we were able to speed up the solution process significantly with respect to runtime and number of explored MIP nodes, as can be seen in Table 4.

7 Outlook

We propose three models to compute cost-optimal public transport plans. For the first two models we derive optimality conditions and with the third model we present an IP formulation for the integrated exact model. The computational experiments show that the implementation of the models is computationally tractable.

Model 1 is able to compute cost-optimal solutions up to `Grid` outperforming previous approaches to tackle this problem. For large networks the model provides bounds of good quality in a reasonable amount of time. Model 2 finds optimal line-pure LTS-plans. Finally, Model 3 yields a cost-optimal LTS-plan without requiring any further assumptions.

For future work we plan to sharpen the formulation of Model 1 by identifying good cuts. It would hopefully be the case that better cuts lead to a further decrease of the computation time, especially for the large instances.
Furthermore it would be interesting to not only find a solution with minimal costs, but to find a *lexicographic* solution, i.e., the cost-optimal solution with the best traveling time for the passengers. To this end, we can include the passengers' traveling time in Model 3 which will most likely further increase the computation time of the model. To use this model effectively, more work in speed-up techniques is necessary. Promising ideas include column generation and decomposition techniques, similar to the methods presented in [22].

References

1 S. Albert, J. Pätzold, A. Schiewe, P. Schiewe, and A. Schöbel. LinTim - Integrated Optimization in Public Transportation. Homepage. see http://lintim.math.uni-goettingen.de/.

2 R. Borndörfer, M. Grötschel, and M. Pfetsch. A column-generation approach to line planning in public transport. *Transportation Science*, 41:123–132, 2007.

3 R. Borndörfer, H. Hoppmann, and M. Karbstein. Passenger routing for periodic timetable optimization. *Public Transport*, 9(1-2):115–135, 2017.

4 R. Borndörfer, M. Neumann, and M. E. Pfetsch. The line connectivity problem. In *Operations Research Proceedings 2008*, pages 557–562. Springer, 2009.

5 Ralf Borndörfer, Marika Karbstein, Christian Liebchen, and Niels Lindner. A Simple Way to Compute the Number of Vehicles That Are Required to Operate a Periodic Timetable. In Ralf Borndörfer and Sabine Storandt, editors, *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*, volume 65 of *OpenAccess Series in Informatics (OASIcs)*, pages 16:1–16:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2018.16.

6 S. Bunte and N. Kliewer. An overview on vehicle scheduling models. *Public Transport*, 1(4):299–317, 2009.

7 S. Burggraeve, S.H. Bull, R.M. Lusby, and P. Vansteenwegen. Integrating robust timetabling in line plan optimization for railway systems. *Transportation Research C*, 77:134–160, 2017.

8 M.R. Bussieck. *Optimal lines in public transport.* PhD thesis, Technische Universität Braunschweig, 1998.

9 A. Ceder and N.H.M. Wilson. Bus network design. *Transportation Research Part B: Methodological*, 20(4):331–344, 1986.

10 M.T. Claessens, N.M. van Dijk, and P.J. Zwaneveld. Cost optimal allocation of rail passenger lines. *European Journal on Operational Research*, 110:474–489, 1998.

11 G. Desaulniers and M.D. Hickman. Public transit. *Handbooks in operations research and management science*, 14:69–127, 2007.

12 H. Fleischner. X. 1 algorithms for eulerian trails. eulerian graphs and related topics: Part 1. *Annals of Discrete Mathematics*, 50:1–13, 1991.

13 M. Friedrich, M. Hartl, A. Schiewe, and A. Schöbel. Angebotsplanung im öffentlichen Verkehr - planerische und algorithmische Lösungen. In *Heureka'17*, 2017.

14 P. Gattermann, J. Harbering, and A. Schöbel. Line pool generation. *Public Transport*, 9(1):7–32, 2017.

15 Philine Gattermann, Peter Großmann, Karl Nachtigall, and Anita Schöbel. Integrating Passengers' Routes in Periodic Timetabling: A SAT approach. In Marc Goerigk and Renato Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in*

Informatics (OASIcs), pages 3:1–3:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2016.3.

16 M. Goerigk, M. Schachtebeck, and A. Schöbel. Evaluating line concepts using travel times and robustness: Simulations with the lintim toolbox. *Public Transport*, 5(3), 2013.

17 M. Goerigk and A. Schöbel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers and Operations Research*, 40(5):1363–1370, 2013.

18 J. Goossens, C.P.M. van Hoesel, and L.G. Kroon. On solving multi-type railway line planning problems. *European Journal of Operational Research*, 168(2):403–424, 2006.

19 C. Liebchen. *Periodic Timetable Optimization in Public Transport*. dissertation.de – Verlag im Internet, Berlin, 2006.

20 C. Liebchen. Nutzung graphentheoretischer konzepte zur manuellen erstellung effizienter verkehrsangebote. In J. Schönberger & S. Nerlich, editor, *26. Verkehrswissenschaftliche Tage*, pages 309–332, Dresden, Germany, 2018. Technische Universität Dresden, Fakultät Verkehrswissenschaften "Friedrich List".

21 C. Liebchen and R. Möhring. The modeling power of the periodic event scheduling problem: Railway timetables - and beyond. In *Proceedings of 9th meeting on Computer-Aided Scheduling of Public Transport(CASPT 2004)*. Springer, 2004.

22 M. Lübbecke, C. Puchert, P. Schiewe, and A. Schöbel. Detecting structures in network models of integrated traffic planning. Presentation at the Clausthal-Göttingen International Workshop on Simulation Science.

23 K. Nachtigall. *Periodic Network Optimization and Fixed Interval Timetables*. PhD thesis, University of Hildesheim, 1998.

24 Karl Nachtigall and Jens Opitz. Solving Periodic Timetable Optimisation Problems by Modulo Simplex Calculations. In Matteo Fischetti and Peter Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*, volume 9 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2008.1588.

25 Julius Pätzold, Alexander Schiewe, Philine Schiewe, and Anita Schöbel. Look-Ahead Approaches for Integrated Planning in Public Transportation. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 17:1–17:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2017.17.

26 Julius Pätzold and Anita Schöbel. A Matching Approach for Periodic Timetabling. In Marc Goerigk and Renato Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2016.1.

27 L. Peeters. *Cyclic Railway Timetabling Optimization*. PhD thesis, ERIM, Rotterdam School of Management, 2003.

28 L. Peeters and L. Kroon. A cycle based optimization model for the cyclic railway timetabling problem. In S. Voß and J. Daduna, editors, *Computer-Aided Transit Scheduling*, volume 505 of *Lecture Notes in Economics and Mathematical systems*, pages 275–296. Springer, 2001.

29 L. Peeters and L. Kroon. A variable trip time model for cyclic railway timetabling. *Transportation Science*, 37(2):198–212, 2003.

30 DFG research unit FOR 2083. Public tranport networks. https://github.com/FOR2083/PublicTransportNetworks.

31 A. Schiewe, S. Albert, J. Pätzold, P. Schiewe, A. Schöbel, and J. Schulz. LinTim: An integrated environment for mathematical public transport optimization. Documentation. Technical Report 2018-08, Preprint-Reihe, Institut für Numerische und Angewandte Mathematik, Georg-August-Universität Göttingen, 2018.

32 M. Schmidt and A. Schöbel. Timetabling with passenger routing. *OR Spectrum*, 37:75–97, 2015.

33 A. Schöbel. Line planning in public transportation: models and methods. *OR Spectrum*, 34(3):491–510, 2012.

34 A. Schöbel. An eigenmodel for iterative line planning, timetabling and vehicle scheduling in public transportation. *Transportation Research C*, 74:348–365, 2017.

35 A. Schöbel and S. Scholl. Line planning with minimal travel time. In *5th Workshop on Algorithmic Methods and Models for Optimization of Railways*, number 06901 in Dagstuhl Seminar Proceedings, 2006.

36 P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematic*, 2:550–581, 1989.

37 L.P. Veelenturf, L.G. Kroon, and G. Maroti. Passenger oriented railway disruption management by adapting timetables and rolling stock schedules. *Transportation Research C*, 80:133–147, 2017.

38 P.J. Zwaneveld. *Railway Planning — Routing of trains and allocation of passenger lines.* PhD thesis, School of Management, Rotterdam, 1997.

A Figures

B Proofs

Proof of Theorem 3. Let $(\mathcal{L}, \mathcal{R})$ be some feasible solution to (cost-opt LTS). Since the line plan is feasible we can construct some feasible flow from it by setting $f_e^{\min} = |\{l \in \mathcal{L} | e \in l\}|$ and $f_{e,u} = \sum_{p \in P_{\text{all}}: e \in p} w_p$. Now we get for all $i, j \in V$ with $\{i,j\} \in E$

$$\sum_{u \in V} f_{(i,j),u} = \sum_{p \in P_{\text{all}}:(i,j)\in p} w_p \underbrace{\leq}_{\text{by (1)}} f_e^{\min} \cdot \text{Cap}$$

by definition of feasibility of a line plan, i.e., constraint (11) is satisfied. Since the w_p correspond to paths in the PTN the flow conservation constraints (12) and (13) are also satisfied. By setting

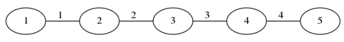
$$\text{dur} = \left\lceil \frac{\sum_{e \in E} 2 f_e^{\min}(L_e^{\text{drive}} + L^{\text{wait}})}{T} \right\rceil$$

we finally have constructed a feasible solution to Model 1.

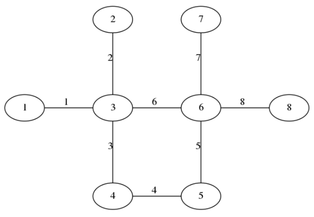
We now show that the objective function value of the constructed solution is better than $g(\mathcal{L}, \mathcal{R}) = c_{\text{time}} \cdot \text{dur}(\mathcal{L}, \mathcal{R}) + c_{\text{length}} \cdot \text{length}(\mathcal{L}, \mathcal{R})$.

We first consider $\text{length}(\mathcal{L}, \mathcal{R})$: We know that for the constructed solution it holds that $f_e^{\min} = |\{l \in \mathcal{L} | e \in l\}|$, hence

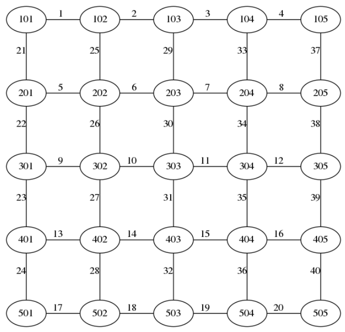
$$\text{length}(\mathcal{L}, \mathcal{R}) \geq \sum_{l' \in \mathcal{L}'} \text{length}_{l'} = \sum_{l \in \mathcal{L}} \sum_{e \in l} 2\text{length}_e \geq \sum_{e \in E} 2\text{length}_e f_e^{\min}.$$



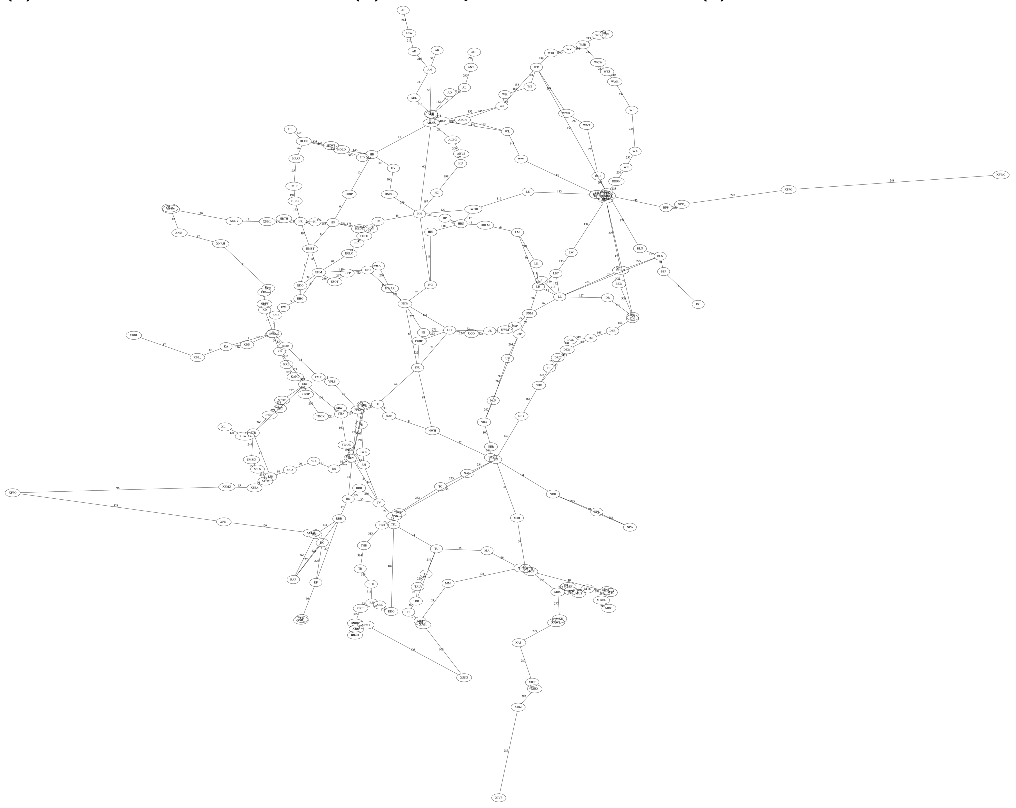
(a) The `Linear` network.



(b) The `Toy` network.



(c) The `Grid` network.



(d) The `Germany` network.

Figure 5 The instances used in the experiments.

For $\operatorname{dur}(\mathcal{L},\mathcal{R})$ we calculate

$$\begin{aligned}
\operatorname{dur}(\mathcal{L},\mathcal{R}) &= \sum_{r\in\mathcal{R}} \operatorname{dur}_r = \sum_{r\in\mathcal{R}} \left\lceil \sum_{l'\in r} (\operatorname{dur}_{l'} + L^{\text{turn}}) \right\rceil_T \geq \left\lceil \sum_{r\in\mathcal{R}} \sum_{l'\in r} (\operatorname{dur}_{l'} + L^{\text{turn}}) \right\rceil_T \\
&\underbrace{=}_{(4)} \left\lceil \sum_{r\in\mathcal{R}} \sum_{l'\in r} \left((|l|-1)L^{\text{wait}} + L^{\text{turn}} + \sum_{e\in l'} L_e^{\text{drive}} \right) \right\rceil_T \\
&= \left\lceil \sum_{l'\in\mathcal{L}} \left(L^{\text{turn}} - L^{\text{wait}} + \sum_{e\in l'} (L_e^{\text{drive}} + L^{\text{wait}}) \right) \right\rceil_T \\
&\geq \left\lceil \sum_{l\in\mathcal{L}} 2 \left(\underbrace{(L^{\text{turn}} - L^{\text{wait}})}_{\geq 0} + \sum_{e\in l} (L_e^{\text{drive}} + L^{\text{wait}}) \right) \right\rceil_T \\
&\underbrace{\geq}_{f_e^{\min} = |\{l\in\mathcal{L} | e\in l\}|} \left\lceil \sum_{e\in E} 2 f_e^{\min} (L_e^{\text{drive}} + L^{\text{wait}}) \right\rceil_T = \operatorname{dur}\cdot T.
\end{aligned}$$

Overall it holds that

$$g(\mathcal{L},\mathcal{R}) = c_{\text{time}}\text{dur}(\mathcal{L},\mathcal{R}) + c_{\text{length}}\text{length}(\mathcal{L},\mathcal{R}) \geq c_{\text{time}}\text{dur}\cdot T + c_{\text{length}}\sum_{e\in E} 2\text{length}_e \cdot f_e^{\min}.$$

Thus every feasible solution to (cost-opt LTS) can be transformed to a solution for Model 1 whose objective is smaller than $g(\mathcal{L},\mathcal{R})$. Hence, the optimal objective function value of Model 1 yields a lower bound to (cost-opt LTS). ◀

Proof of Lemma 4. We start with constraint (12), i.e.,

$$\sum_{i\in V:\{i,v\}\in E} f_{(i,v),u} = W_{uv} + \sum_{i\in V:\{v,i\}\in E} f_{(v,i),u} \quad \forall u\in V \forall v\in V\backslash\{u\}$$

and argue that for any $u\in X$ it holds that

$$\begin{aligned}
&\sum_{v\in Y}\sum_{i\in V:\{i,v\}\in E} f_{(i,v),u} = \sum_{v\in Y}\left(W_{uv} + \sum_{i\in V:\{v,i\}\in E} f_{(v,i),u}\right)\\
\underbrace{\Leftrightarrow}_{V=X\cup Y}\quad &\sum_{v\in Y}\left(\sum_{i\in X:\{i,v\}\in E} f_{(i,v),u} + \sum_{i\in Y:\{i,v\}\in E} \underbrace{f_{(i,v),u}}_{=(*)}\right)\\
&\qquad = \sum_{v\in Y}\left(W_{uv} + \sum_{i\in X:\{v,i\}\in E} f_{(v,i),u} + \sum_{i\in Y:\{v,i\}\in E} \underbrace{f_{(v,i),u}}_{=(*)}\right)\\
\underbrace{\Leftrightarrow}_{(*)\text{ cancel out}}\quad &\sum_{v\in Y}\sum_{i\in X:\{i,v\}\in E} f_{(i,v),u} = \sum_{v\in Y}\left(W_{uv} + \sum_{i\in X:\{v,i\}\in E} f_{(v,i),u}\right)\\
\Leftrightarrow\quad &\sum_{\substack{v\in Y, i\in X:\\ \{v,i\}\in E_{cut}}} f_{(i,v),u} = \sum_{v\in Y} W_{uv} + \sum_{\substack{v\in Y, i\in X:\\ \{v,i\}\in E_{cut}}} f_{(v,i),u}
\end{aligned}$$

Hence we can conclude

$$\sum_{i\in X, v\in Y:\{v,i\}\in E_{cut}} f_{(i,v),u} \geq \sum_{v\in Y} W_{uv} \quad \forall u\in X. \tag{36}$$

Thus we get that

$$\begin{aligned}
\text{Cap}\cdot\sum_{e\in E_{cut}} f_e^{\min} &\underbrace{\geq}_{(11)} \sum_{\substack{i\in X, v\in Y:\\ \{i,v\}\in E_{cut}}}\sum_{u\in V} f_{(i,v),u}\\
&\underbrace{\geq}_{X\subseteq V} \sum_{u\in X}\sum_{\substack{i\in X, v\in Y:\\ \{i,v\}\in E_{cut}}} f_{(i,v),u} \underbrace{\geq}_{(36)} \sum_{u\in X}\sum_{v\in Y} W_{uv}.
\end{aligned}$$

◀

Proof of Theorem 10. Let $(\mathcal{L},\mathcal{R})$ be some feasible solution to (cost-opt LTS). Then we know that we can set $f_e^{\min} = |\{l\in\mathcal{L}|e\in l\}|$ (and $f_{e,u}$ accordingly) as in the proof of Theorem 3 to some feasible flow which satisfies (16). Furthermore we can enumerate all lines with some bijective mapping $\varphi:\mathcal{L}\to[|\mathcal{L}|]$ such that $x_{e,\varphi(l)} = 1$ iff $e\in l$ for all $l\in\mathcal{L}$ and also $y_{s,\varphi(l)} = 1$ iff $s\in e$ for some $e\in l$ and $z_i = 1$ for all $i\in[|\mathcal{L}|]$ and 0 else. Since $\mathcal{L}$ was some feasible line plan all lines are simple paths and hence also constraints (17) to (22) are fulfilled. Now for the objective function it holds that

$$\begin{aligned}\text{length}(\mathcal{L},\mathcal{R}) &= \sum_{l'\in\mathcal{L}'}\text{length}_{l'} + \sum_{r=(l'_1,\dots,l'_{k_r})\in\mathcal{R}}\sum_{i=1}^{k_r-1}\text{length}_{l'_i,l'_{i+1}}\\ &\geq \sum_{l\in\mathcal{L}}\sum_{e\in l}2\text{length}_e = \sum_{l\in\mathcal{L}}\sum_{e\in E}2x_{e,\varphi(l)}\text{length}_e = \sum_{l=1}^{L}\sum_{e\in E}2x_{e,l}\text{length}_e.\end{aligned}$$

For the duration we get

$$\begin{aligned}\text{dur}(\mathcal{L},\mathcal{R}) &= \sum_{r=(l'_1,\dots,l'_{k_r})\in\mathcal{R}}\left\lceil\sum_{i=1}^{k}\text{dur}_{l'_i}+\text{dur}_{l'_i,l'_{i+1}}\right\rceil_T \geq \left\lceil\sum_{r\in\mathcal{R}}\sum_{l'\in r}(\text{dur}_{l'}+L^{\text{turn}})\right\rceil_T\\ &\underbrace{=}_{(4)}\left\lceil\sum_{r\in\mathcal{R}}\sum_{l'\in r}\left((|l|-1)L^{\text{wait}}+L^{\text{turn}}+\sum_{e\in l'}L_e^{\text{drive}}\right)\right\rceil_T\\ &=\left\lceil\sum_{l'\in\mathcal{L}}\left(L^{\text{turn}}-L^{\text{wait}}+\sum_{e\in l'}(L_e^{\text{drive}}+L^{\text{wait}})\right)\right\rceil_T\\ &=\left\lceil\sum_{l=1}^{L}\left(2z_l(L^{\text{turn}}-L^{\text{wait}})+\sum_{e\in E}2(L_e^{\text{drive}}+L^{\text{wait}})\cdot x_{e,l}\right)\right\rceil_T \geq \text{dur}\cdot T\end{aligned}$$

Hence, by finally setting

$$\text{dur} = \left\lceil\frac{\sum_{l=1}^{L}\left(2z_l(L^{\text{turn}}-L^{\text{wait}})+\sum_{e\in E}2(L_e^{\text{drive}}+L^{\text{wait}})\cdot x_{e,l}\right)}{T}\right\rceil$$

we conclude that from any feasible solution $(\mathcal{L},\mathcal{R})$ to (cost-opt LTS) we can construct some feasible solution to Model 2 such that

$$g(\mathcal{L},\mathcal{R}) \geq c_{\text{time}}\text{dur}\cdot T + c_{\text{length}}\sum_{l=1}^{L}\sum_{e\in E}2x_{e,l}\text{length}_e,$$

which means that the objective function value of Model 2 is a lower bound to (cost-opt LTS). On the other hand every feasible solution to Model 2 is a feasible solution to Model 1. This can be seen by setting the three types of variables, $f_e^{\min}$, $f_{e,u}$ and dur, that are contained in both models, to be the same. Hence constraints (11) - (13) are satisfied, and also (10) is satisfied since

$$\text{dur}\cdot T \geq \sum_{l=1}^{L}\left(2z_l\underbrace{(L^{\text{turn}}-L^{\text{wait}})}_{\geq 0}+\sum_{e\in E}2(L_e^{\text{drive}}+L^{\text{wait}})\cdot x_{e,l}\right) \geq \sum_{e\in E}2f_e^{\min}(L_e^{\text{drive}}+L^{\text{wait}}).$$

For the objective functions it additionally holds that

$$\sum_{l=1}^{L}\sum_{e\in E}2x_{e,l}\text{length}_e = \sum_{e\in E}2f_e^{\min}\text{length}_e.$$

This means that every solution to Model 2 can be projected to a solution of Model 1 with smaller objective value in Model 1, meaning that Model 2 is an upper bound to Model 1. ◀

Proof of Theorem 13. Let $\mathcal{L}, \mathcal{R}$ be some line-pure feasible solution to (cost-opt LTS). For the objective value of $(\mathcal{L}, \mathcal{R})$ we know that

$$\text{length}(\mathcal{L}, \mathcal{R}) = \sum_{r=(l'_1,\ldots,l'_{k_r})\in\mathcal{R}} \sum_{i=1}^{k_r} \text{length}_{l'_i} + \underbrace{\text{length}_{l'_i,l'_{i+1}}}_{=0} = \sum_{l\in\mathcal{L}} 2\text{length}_l = \sum_{l\in\mathcal{L}} \sum_{e\in l} 2\text{length}_e,$$

and that

$$\begin{aligned}\text{dur}(\mathcal{L}, \mathcal{R}) &= \sum_{r\in R} \left\lceil \sum_{l'\in r} (\text{dur}_{l'} + L^{\text{turn}}) \right\rceil_T = \sum_{l\in\mathcal{L}} \left\lceil 2(\text{dur}_l + L^{\text{turn}}) \right\rceil_T \\ &= \sum_{l\in\mathcal{L}} \left\lceil 2(L_{\text{turn}} - L_{\text{wait}}) + \sum_{e\in E: e\in l} 2(L_e^{\text{drive}} + L^{\text{wait}}) \right\rceil_T .\end{aligned}$$

We can extend the line plan $\mathcal{L}$ to some feasible solution to Model 2* by again defining a bijective mapping $\varphi : \mathcal{L} \to [|\mathcal{L}|]$ such that $x_{e,\varphi(l)} = 1$ iff $e \in l$ for $l \in \mathcal{L}$ for all $e \in E$. Analogously a solution $x_{e,l}$ can be transformed into some feasible line plan $\mathcal{L}$ by defining a line l to contain exactly all edges $e \in E$ if $x_{e,l} = 1$. Thus there exists a bijection between the set of feasible solutions between (cost-opt LTS) and Model 2* as well as the same objective function for both problems since

$$\sum_{l\in\mathcal{L}} \sum_{e\in l} 2\text{length}_e = \sum_{l\in\mathcal{L}} \sum_{e\in E} 2x_{e,\varphi(l)}\text{length}_e = \sum_{l=1}^{L} \sum_{e\in E} 2x_{e,l}\text{length}_e$$

and

$$\begin{aligned}&\sum_{l\in\mathcal{L}} \left\lceil 2(L_{\text{turn}} - L_{\text{wait}}) + \sum_{e\in E: e\in l} 2(L_e^{\text{drive}} + L^{\text{wait}}) \right\rceil_T \\ &= \sum_{l=1}^{L} \left\lceil 2z_l(L^{\text{turn}} - L^{\text{wait}}) + \sum_{e\in E} 2x_{e,l}\text{length}_e(L_e^{\text{drive}} + L^{\text{wait}}) \right\rceil_T = \sum_{l=1}^{L} d_l.\end{aligned}$$

Hence their optimal objective values coincide. ◀

Changing Lanes on a Highway

Thomas Petig
Qamcom Research and Technology AB, Sweden

Elad M. Schiller
Chalmers University of Technology, Sweden

Jukka Suomela
Aalto University, Finland

Abstract

We study a combinatorial optimization problem that is motivated by the scenario of autonomous cars driving on a multi-lane highway: some cars need to change lanes before the next intersection, and if there is congestion, cars need to slow down to make space for those who are changing lanes. There are two natural objective functions to minimize: (1) how long does it take for all traffic to clear the road, and (2) the total number of maneuvers. In this work, we present an approximation algorithm for solving these problems in the two-lane case and a hardness result for the multi-lane case.

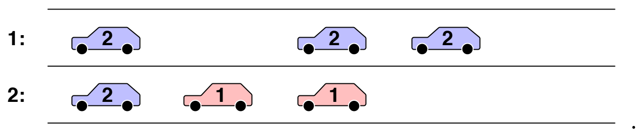
2012 ACM Subject Classification Theory of computation → Discrete optimization

Keywords and phrases Collaborative agents, vehicle scheduling, traffic optimization, approximation algorithms

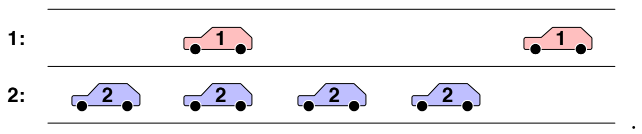
Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.9

1 Introduction

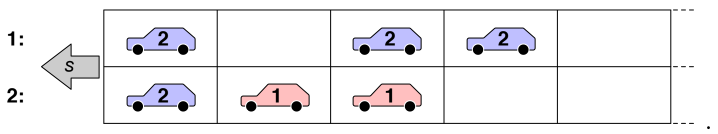
Consider a fleet of autonomous vehicles driving on a two-lane highway:



Each car is labeled with a lane number, 1 or 2, indicating where it needs to be before the next intersection. Our task is to instruct the cars to adjust their speed and change lanes so that all cars with label ℓ are on lane ℓ:

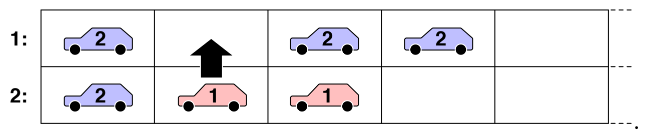


We discretize the traffic by assuming that there is a grid of *slots* that is moving at some fixed speed s (for example, s is the speed limit of the highway), and each car occupies one slot (there are infinitely many free slots behind the last cars):

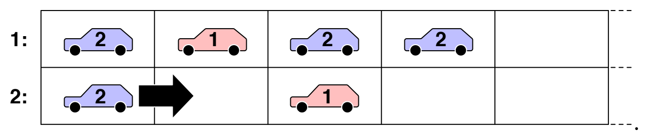


If there are no steering maneuvers, each car will remain in its current slot (i.e., it is driving along the current lane, at a constant speed s). We can use the following maneuvers to alter the relative positions of the cars.

First, a car that is **currently on the wrong lane** can **switch lanes**, assuming there is an empty slot next to it:



Second, any car can **slow down** a bit to move backwards relative to the traffic around it, assuming there is an empty slot behind it:



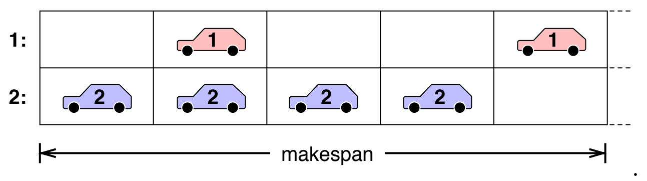
We emphasize that we do not allow cars that are on the right lane to switch lanes any more; permitting such maneuvers would also give rise to an interesting problem formulation to be studied in future work.

1.1 Objectives

It is easy to find a feasible solution by following a simple greedy strategy, e.g., for each car x that is on the wrong lane, slow down all cars behind x on either lane to make space for x to move to the right lane. However, this is not an optimal strategy in the general case.

We will consider the following objective functions that we would like to minimize:

- **Makespan**: What is the last non-empty slot that is occupied by a car in the final configuration? Intuitively, we measure here *how much do we stretch the traffic*, or equivalently, *how long does it take for all traffic to clear the road*:



- **Total cost**: What is the total number of steering maneuvers (switching lanes or slowing down) that we need to solve the problem? Note that in our problem formulation, the number of lane changes is simply equal to the number of cars on the wrong lane, so the interesting question is the number of slow-down operations. Intuitively, we measure here the *average delay for the traffic.*

To focus on the more interesting algorithmic aspects, we present our algorithms from the perspective of a global omniscient entity that has a full control over all vehicles. However, the same ideas can be applied in distributed and online settings.

1.2 Contributions and open questions

We develop a polynomial-time algorithm for the **two-lane** version of the lane-changing problem. The algorithm finds a solution that **minimizes the makespan** and that is also a **1.5-approximation of the minimum total cost**. Moreover, we show that a natural **multi-lane** extension of the problem is **NP-hard**.

Our work also suggests the following natural question for further research: is it possible to find an exact solution for the minimum-cost two-lane version in polynomial time?

2 Related work

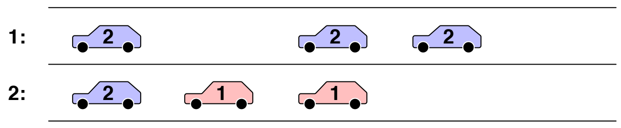
Tile-sliding puzzles. We note the resemblance between the problem studied by us in this work and combinatorial puzzles such as the "15-puzzle" [8,17], which is a game that considers a four by four matrix that has 15 tiles, labeled with numbers from 1 to 15 in an arbitrary order. The goal of the game is to slide the tiles so that the tiles are ordered. For large-scale versions of the n-puzzle, finding an optimum solution is NP-hard [13,14]. Our problem can also be seen as a tile-sliding puzzle, but differs from the 15-puzzle in the following aspects: many tiles may have the same label, the label only determines the final row (and not column), and our moves are more restricted (for example, tiles cannot slide right). Feigenbaum et al. [3] formulated a number of graph problems for the semi-streaming model. Unlike their model, our problem does not allow a pair of nearby agents to swap cells.

Vehicular control. Problems related to lane-change consider traffic streams from the point of view of vehicular control [2,6,12], traffic flow control [9], the scheduling of lane changes for autonomous vehicles [1], assessment of the situation before changing lane [15], and negotiation before lane changing [16] to name a few. It is often the case, as in [12], that these problems consider a small set of nearby vehicles that need to coordinate a single lane-change maneuver. A number of recent efforts, such as the European project AutoNet2030 [16], considers the need to perform lane changes in congested traffic situations, as we do in this paper. Their study focuses on distributed mechanisms, i.e., the communication protocols, for enabling coordinated lane changes whereas this work focuses on the algorithmic question of minimizing the number of maneuvers.

Traffic models. Cellular automata are often used for microscopic traffic flow prediction [10]. These models resembles the one of the studied problem in the sense that each vehicle occupies a single cell. However, Nagel [10] considers cellular automata that move the vehicles forward, whereas our model considers vehicles that can merely change their current lanes or delay – we do not aim at predicting traffic patterns and just aim at minimizing the number of delays and lane changes. The systematic approach presented in [11] shows that their lane change rules can provide "realistic results" with respect to the system ability to offer an accurate traffic prediction. A complementary approach for studying the effect of lane-change behavior via cellular automata [7] is the observation of driver behavior [4,18].

3 Preliminaries

Matrix notation. We will interpret the highway as a matrix with two rows and infinitely many columns; rows correspond to lanes and matrix elements correspond to slots. The rows are numbered $i = 1, 2$ and the columns are numbered $j = 1, 2, \ldots$. We use values 1 and 2 to denote cars with target lanes 1 and 2, respectively, and we use the symbol $\circ$ to emphasize that a slot is empty. For example, the configuration



is represented as a matrix

$$\begin{bmatrix} 2 & \circ & 2 & 2 & \circ & \cdots \\ 2 & 1 & 1 & \circ & \circ & \cdots \end{bmatrix},$$

which we may write for brevity simply as

$$\begin{bmatrix} 2 & \circ & 2 & 2 \\ 2 & 1 & 1 & \circ \end{bmatrix}.$$

Legal moves. In the lane-changing problem, we can apply the following operations to any part of the configuration matrix.

- *Switch* (switch lanes, i.e., move up or down):

$$\begin{bmatrix} 2 \\ \circ \end{bmatrix} \mapsto \begin{bmatrix} \circ \\ 2 \end{bmatrix}, \quad \begin{bmatrix} \circ \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} 1 \\ \circ \end{bmatrix}.$$

- *Delay* (slow down, i.e., move right):

$$\begin{bmatrix} 1 & \circ \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \end{bmatrix}, \quad \begin{bmatrix} 2 & \circ \end{bmatrix} \mapsto \begin{bmatrix} \circ & 2 \end{bmatrix}.$$

Pairs. A *pair* is one column of the configuration; an *input pair* is a column that occurs in the input. For brevity, a column $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ is called an (x, y)-pair.

Solution. A *feasible configuration* is a configuration in which all non-empty slots of row 1 contain label 1, and all non-empty slots of row 2 contain label 2. That is, each car is on its target lane.

A *feasible solution* to the lane-changing problem is a sequence of legal moves that turns the given input configuration into a feasible configuration. The *cost* of a solution is the number of moves. The *makespan* of a solution is the largest j such that column j of the final configuration contains a car.

4 Roadmap

Three problems. To develop an algorithm for solving the lane-changing problem, it will be helpful to also consider two variants of it:

P0. The original lane-changing problem, as defined above.

P1. A restricted version of P0: a solution is feasible only if each car that was involved in a $(2, 1)$-input pair has been delayed at least once.

P2. A relaxed version of P1: multiple cars may occupy the same slot in intermediate configurations.

In P2 we will use notation 1^a2^b to denote a slot with a cars of label 1 and b cars of label 2. A legal P2-move is hence, for example,

$$\begin{bmatrix} 1^3 2^3 & 1^5 2^5 \end{bmatrix} \mapsto \begin{bmatrix} 1^3 2^2 & 1^5 2^6 \end{bmatrix}.$$

The final configuration in P2 has to be feasible in the usual sense: each slot contains at most one car, and each car is in the right lane. Also note that one move can only change the position of one car.

Simple examples. Consider the input

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

A feasible P0-solution might take, e.g., the following steps (cost 3, makespan 2):

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} 2 & \circ \\ \circ & 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ & \circ \\ 2 & 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ 2 & \circ \end{bmatrix}.$$

This would not be a feasible P1-solution, though, as there is a car with label 2 that was part of a $(2,1)$-pair in the input but the car was not delayed. A feasible P1-solution might take the following steps (cost 4, makespan 2):

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} 2 & \circ \\ \circ & 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ & \circ \\ 2 & 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ 2 & \circ \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ \circ & 2 \end{bmatrix}.$$

In a feasible P2-solution we could also take the following route in which we have multiple cars in one slot in an intermediate configuration (but this is not any cheaper; we still have cost 4 and makespan 2):

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ \\ 12 \end{bmatrix} \mapsto \begin{bmatrix} 1 \\ 2 \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ 2 & \circ \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ \circ & 2 \end{bmatrix}.$$

Preliminary observations. We emphasize that problems P1 and P2 are not interesting in their own right; we only care about problem P0. Both P0 and P2 can be seen as relaxations of P1, but they are relaxations of a very different nature:

- A feasible solution to P1 is also a feasible solution to P0, but it might take some additional steps that are only necessary to handle $(2,1)$-pairs.
- A feasible solution to P1 is also a feasible solution to P2, but it might take some additional steps that are only necessary to ensure there is at most one car per slot.

At first, P2 and P0 seem to be incomparable. A P2-solution is not necessarily a P0-solution, or vice versa. But as we will see in this work, an algorithm for solving P2 can be a helpful starting point in solving P0, too.

Key ideas. The key insights of our work are these results that we will prove:

- For P2 there is always a solution that *simultaneously minimizes both makespan and cost.*
- Problem P1 can be solved with the *same makespan and cost* as problem P2.
- A makespan-optimal P1-solution is also a makespan-optimal P0-solution.
- A cost-optimal P1-solution is also a 1.5-approximation of a cost-optimal P0-solution.

We will use the above ideas to solve problem P0 as follows:

- In Section 5.1, we design **algorithm A2** that will find a P2-solution that is simultaneously cost-optimal and makespan-optimal.

- In Section 5.2, we design **algorithm A1** that finds a P1-solution that has the same cost and makespan as the P2-solution returned by A2. As P2 is a relaxation of P1, it follows that A1 returns a cost-optimal and makespan-optimal P1-solution.
- Now it is clear that A1 also returns a P0-solution, as P0 is a relaxation of P1. However, we will still need to prove that the solution returned by A1 is a makespan-optimal P0-solution and also a 1.5-approximation of a cost-optimal P0-solution. The proof is given in Section 5.3.

5 Algorithm details

Notation. Let W be the total number of cars that are on the wrong lane in the input configuration. Any feasible solution contains exactly W switch operations. Hence a minimum-cost solution is a solution that minimizes the number of delay operations.

5.1 Solving problem P2

Flow equations. Let us first develop some *necessary* conditions that characterize feasible solutions for P2 (and hence they are also necessary conditions for a feasible solution of P1).

Consider some feasible solution Y. Let $\ell = 1, 2$ be a label. We will consider the *flow* of cars of label ℓ:

- $s_\ell(j)$ is the number of cars of label ℓ in column j in the input configuration,
- $t_\ell(j)$ is the number of cars of label ℓ in column j in the final configuration,
- $f_\ell(j)$ is the number of times a car of label ℓ is moved from column j to column $j+1$.

Recall that the columns are numbered $j = 1, 2, \dots$, but for convenience, we also define $f_\ell(0) = 0$ so that we can always refer to $f_\ell(j-1)$. Let us now define the grand total of flow that we will need to handle at column j:

$$g_\ell(j) = f_\ell(j-1) + s_\ell(j). \tag{1}$$

As no car is lost or created, flow is conserved:

$$g_\ell(j) = t_\ell(j) + f_\ell(j). \tag{2}$$

In the final configuration we have got at most one car per slot:

$$t_\ell(j) \le 1. \tag{3}$$

Hence by (2) and (3) we necessarily have

$$f_\ell(j) \ge g_\ell(j) - 1. \tag{4}$$

By the definition of problem P1 (and hence P2), cars in $(2,1)$-input pairs are always delayed at least once. To capture this, define the indicator function p as follows:

$$p_\ell(j) = 1 \text{ if there is a } (2,1)\text{-input pair in column } j \text{ in the input configuration.}$$

Using this notation, we have for each $j = 1, 2, \dots$

$$f_\ell(j) \ge p_\ell(j). \tag{5}$$

Now t and f may depend on the particular solution Y, but s and p only depend on the input configuration. For any given input, we can recursively calculate a *minimal* flow f^* that satisfies (1), (4), and (5):

$$g^*_\ell(j) = f^*_\ell(j-1) + s_\ell(j) \qquad \text{for all } j = 1, 2, \ldots, \tag{6}$$

$$f^*_\ell(j) = \max\{p_\ell(j), g^*_\ell(j) - 1\} \qquad \text{for all } j = 1, 2, \ldots. \tag{7}$$

Again we follow the convention that $f^*_\ell(0) = 0$ so that $f^*_\ell(j-1)$ is well-defined for every column j. Note that for all ℓ and j and for any feasible flow f, we have by construction $f^*_\ell(j) \le f_\ell(j)$. Hence we can make the following observations:

▶ **Lemma 1.** *The cost of any feasible P2-solution is at least* $W + \sum_{\ell,j} f^*_\ell(j)$*.*

▶ **Lemma 2.** *For all* ℓ *and* j*, if* $g^*_\ell(j) > 0$*, then the makespan of any feasible P2-solution is at least* j*.*

Algorithm A2. Now it is sufficient to design an algorithm that moves cars precisely according to the minimal flow f^*; if we can do that, the solution will be both cost-optimal and makespan-optimal.

But this is easy: First each car switches to the right lane; this takes W moves. Then we follow (6)–(7) for columns $j = 1, 2, \ldots$ in ascending order: first we move $f^*_\ell(1)$ cars of label ℓ from column 1 to column 2, then we have $g^*_\ell(2)$ cars of label ℓ in column 2, etc. If we always move first those cars that were already present in a given slot in the input configuration, we will satisfy all constraints of problem P2, including the special rule about $(2,1)$-pairs.

We have now algorithm A2 that finds simultaneously cost-optimal and makespan-optimal solutions for P2. However, this is clearly not a solution for P1, as we may have multiple cars in one slot in intermediate configurations.

5.2 Solving problem P1

Idea. We now develop algorithm A1 that follows the same minimal flow f^*, but schedules the operations differently so that it produces a feasible solution to problem P1:

- Algorithm A2 "pushes" cars starting from the first cars.
- Algorithm A1 "pulls" cars starting from the last cars.

Our basic idea is to show that – with a little bit of planning ahead – we can move cars according to f^* without putting multiple cars in one slot.

In the algorithm we will update s, p, f^*, and g^* as we move cars around so that they refer to the current configuration, and not the input configuration. Eventually all cars will be in their final positions, there is no need to move anything, and f^* will be zero.

Trivial and tricky pairs. A pair of type $(\circ, 1)$ or $(2, \circ)$ is called a *trivial pair*. As the first step of the algorithm, each trivial pair will switch lanes; hence we eliminate all trivial pairs. Our algorithm will ensure that whenever we create new trivial pairs, they are also eliminated immediately.

A pair of type $(2, 1)$ is called a *tricky pair*. We will make sure that the algorithm only eliminates tricky pairs and never create new tricky pairs. We will use p to keep track of the tricky pairs that were present in the input: Initially, $p_1(j) = p_2(j) = 1$ if we have a tricky pair in column j. Then whenever we delay a car with label ℓ in column j, we set $p_\ell(j) \leftarrow 0$.

Active and hot columns. We say that a column j is *ℓ-active* if we have $s_\ell(j) > 0$ and $f^*_\ell(j) > 0$. A column is *active* if it is ℓ-active for some ℓ.

A column is *hot* if it is the rightmost (last) active column. The hot column is called *ℓ-hot* if it is ℓ-active. (Note that there is at most one hot column, and the hot column is 1-hot, 2-hot, or both. Also note that the rightmost 1-active column is not necessarily 1-hot, as there might be a 2-active column that is further right, and vice versa.)

Intuitively, an active column contains some cars that are not in their final positions, and the hot column contains the last cars that are not in their final positions. As long as f^* is somewhere nonzero, there has to be an active column, and hence also a hot column.

The following lemmas summarize the key properties that we use.

▶ **Lemma 3.** *Assume that*
- *there are no trivial pairs,*
- *column j is ℓ-hot,*
- *slot (ℓ, j) contains a car with label ℓ.*

Then slot $(\ell, j+1)$ has to be empty.

Proof. Assume w.l.o.g. that $\ell = 1$; the case of $\ell = 2$ is analogous.

If column $j+1$ contains a car of label 1, then $f^*_1(j) + s_1(j+1) \geq 2$, and therefore $f^*_1(j+1) \geq 1$. But this would mean that column $j+1$ is active, which contradicts the assumption that j is hot (i.e., the rightmost active column).

If column $j+1$ does not contain any car of label 1, but slot $(\ell, j+1)$ is not empty, the only possibility is that column $j+1$ contains a pair $(2,2)$. But then we would have $s_2(j+1) \geq 2$ and $f^*_2(j+1) \geq 1$ and again $j+1$ would be active. ◀

▶ **Lemma 4.** *Assume that*
- *column j is hot,*
- *column j contains a tricky pair.*

Then column $j+1$ has to be empty.

Proof. The tricky pair implies that $f^*_1(j) \geq 1$ and $f^*_2(j) \geq 1$. If column $j+1$ contains a car with label ℓ in the current configuration, we will have $s_\ell(j+1) \geq 1$, and hence $g^*_\ell(j+1) \geq 2$ and $f^*_\ell(j+1) \geq 1$. Therefore, column $j+1$ would be active, which contradicts the assumption that j is hot (i.e., the rightmost active column). ◀

Algorithm A1. If we do not have any hot columns, we are done. Otherwise, let j be an ℓ-hot column. Our goal is to show that the algorithm can make progress and delay at least one car in the hot column. We have two cases:

1. Slot (ℓ, j) contains a car with label ℓ: By Lemma 3, we can delay the car with label ℓ in row ℓ.

$$\begin{bmatrix} 1 & \circ \\ x & y \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ x & y \end{bmatrix}, \qquad \begin{bmatrix} x & y \\ 2 & \circ \end{bmatrix} \mapsto \begin{bmatrix} x & y \\ \circ & 2 \end{bmatrix}$$

This cannot create tricky or trivial pairs in column $j+1$. If this resulted in a trivial pair in column j, the algorithm then eliminates it with a switch, e.g.:

$$\begin{bmatrix} 1 & \circ \\ 1 & y \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ 1 & y \end{bmatrix} \mapsto \begin{bmatrix} 1 & 1 \\ \circ & y \end{bmatrix}.$$

2. Slot (ℓ, j) does not contain a car with label ℓ: By the definition of an ℓ-hot column, there has to be a car ℓ somewhere in column j, and as we do not have trivial pairs, we must have a tricky pair. By Lemma 4 column $j+1$ is empty. Hence we can move car 1 from column j to column $j+1$, and this creates trivial pairs in both column j and column $j+1$. Then we perform two switch operations to eliminate the trivial pairs:
$$\begin{bmatrix} 2 & \circ \\ 1 & \circ \end{bmatrix} \mapsto \begin{bmatrix} 2 & \circ \\ \circ & 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ & \circ \\ 2 & 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ 2 & \circ \end{bmatrix}.$$
(Note that here car 2 is not in its final position, $p_2(j)$ is still nonzero, the column remains active, it will eventually become hot, and the car will be moved right.)

Hence in all cases the algorithm can move at least one car, and we can calculate each move efficiently. By construction, both the cost and the makespan of algorithm A1 are the same as in the solution returned by algorithm A2; as A2 solved P2 optimally, and P2 is a relaxation of P1, we conclude that A1 solves P1 optimally.

5.3 Solving problem P0

Recall that P0 is a relaxation of P1. Hence we can directly use algorithm A1 to solve also P0. The following lemma shows that any P1-optimal solution is also a relatively good solution for P0.

▶ **Lemma 5.** *Assume that there is a solution X for P0 that uses W switch operations and D delay operations and has a makespan M. Then it is possible to find a solution Y for P1 that uses W switch operations and at most $D + \min(D, W)$ delay operations and has a makespan M.*

To prove the lemma, we show how to modify X to construct Y. We begin with definitions.

Bad cars and bad blocks. Consider the trajectories of the cars in solution X.

We say that a car is *switch-only* if it only switches lanes once and is never delayed. For example, a switch-only car with label 2 was initially in slot $(1, j)$ for some j, and in the final configuration it is in slot $(2, j)$ for the same j. Note that each column contains at most one switch-only car.

We say that a switch-only car is *bad* if it is part of a $(2, 1)$-input pair. We may have such in P0-solution X but we must not have them in P1-solution Y.

A *bad block of type ℓ* is a range of columns $j, j+1, \dots, k-1$ such that:
1. Column j contains a bad car with label ℓ.
2. Each of columns $j+1, \dots, k-1$ contains a switch-only car with label ℓ. (Some of these cars may also be bad.)
3. Column k does not contain any switch-only cars with label ℓ. (Note that this column may contain a switch-only car of the opposite type, and it may be bad.)

For brevity, we write $[j, k)$ for the range of columns $j, j+1, \dots, k-1$. Note that if we have a bad car in column j, we can always find some k such that $[j, k)$ is a bad block.

Eliminating bad blocks. Our plan is that we identify the first bad block, and manipulate the solution locally so that none of the columns $[j, k)$ contain any bad cars. Then we repeat this until there are no bad blocks (and hence no bad cars) left.

Consider the first bad block $[j, k)$ that we have not yet eliminated; we write $L = k - j$ for the length of the bad block. W.l.o.g., assume that the bad car in column j has label 2; the other case is analogous. The input configuration of the bad block looks like

$$\begin{array}{cccc} {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} \\ \left[\begin{array}{c} 2 \\ 1 \end{array}\right. & \begin{array}{c} 2 \\ ? \end{array} & \begin{array}{c} \cdots \\ \cdots \end{array} & \left.\begin{array}{c} 2 \\ ? \end{array}\right] \end{array},$$

and an output configuration of the block looks like

$$\begin{array}{cccc} {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} \\ \left[\begin{array}{c} ? \\ 2 \end{array}\right. & \begin{array}{c} ? \\ 2 \end{array} & \begin{array}{c} \cdots \\ \cdots \end{array} & \left.\begin{array}{c} ? \\ 2 \end{array}\right] \end{array}.$$

Consider the trajectory of the car 1 that was originally in column j; this is called the *leading car*. The leading car was moved from column j to column $j + 1$ before the switch-only cars in columns j and $j + 1$ moved. It was also moved from column $j + 1$ to column $j + 2$ before the switch-only cars in columns $j + 1$ and $j + 2$ moved, etc. In the final configuration the leading car has to be outside the bad block. Inside the bad block, solution X performs at least L switch operations (L switch-only cars) and at least L delay operations (one leading car moved L times). Note that in our bookkeeping, we associate the cost of a delay operation with the source column.

Case 1: Empty slot follows. Now first consider the possibility that slot $(2, k)$ is empty in the final configuration. Then we can eliminate all bad cars within the bad block with L additional delay operations: delay the cars in $(2, k - 1)$, $(2, k - 2)$, . . . , $(2, j)$ in this order. In essence, we turn

$$\left[\begin{array}{cccc|c} {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} & {\scriptstyle k} \\ ? & ? & \cdots & ? & ? \\ 2 & 2 & \cdots & 2 & \circ \end{array}\right]$$

into

$$\left[\begin{array}{cccc|c} {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} & {\scriptstyle k} \\ ? & ? & \cdots & ? & ? \\ \circ & 2 & \cdots & 2 & 2 \end{array}\right].$$

Note that we modify column k which is outside the current bad block, and there might be another bad block starting at column k. However, it can be verified that what we do with block $[j, k)$ is compatible with what we do with the block starting at k.

Case 2: Non-empty slot follows. Now assume that slot $(2, k)$ is occupied in the final configuration; let us call it the *trailing car*. It has to be a car with label 2:

$$\left[\begin{array}{cccc|c} {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} & {\scriptstyle k} \\ ? & ? & \cdots & ? & ? \\ 2 & 2 & \cdots & 2 & 2 \end{array}\right].$$

By definition, it is not a switch-only car. Also the trailing car was not there initially; otherwise it would have blocked the path of the leading car.

Working backwards from the final position of the trailing car, we can see that the trailing car had to be the last car that occupied slot $(2, k - 1)$ before the switch-only car in column

$k-1$ moved there, and it also had to be the last car that occupied slot $(2, k-2)$ before the switch-only car in column $k-2$ moved there, etc. The trailing car could not have been originally position in any of these slots, as it would have blocked the way of the leading car. Hence at some point the trailing car followed the path $(2, j-1) \to (2, j) \to \ldots \to (2, k)$, and in each column the switch-only cars moved only after the trailing car was gone.

To recap, we have got in total $L+1$ cars with label 2 that were in some intermediate configuration placed as follows; we denote the trailing car with 2^*:

$$\begin{array}{c|cccc|c} j-1 & j & j+1 & & k-1 & k \\ ? & 2 & 2 & \cdots & 2 & ? \\ 2^* & ? & ? & \cdots & ? & ? \end{array}. \tag{8}$$

The trailing car moves rightwards, and the switch-only car in column $k-1$ switches lanes. At some point we reach the following configuration; here 2? denotes a slot that is either empty (a switch-only car has not switched yet) or it contains a car with label 2:

$$\begin{array}{c|cccc|c} j-1 & j & j+1 & & k-1 & k \\ ? & ? & ? & \cdots & \circ & ? \\ ? & 2? & 2? & \cdots & 2 & 2^* \end{array}. \tag{9}$$

Finally, the cars end up in the following positions:

$$\begin{array}{c|cccc|c} j-1 & j & j+1 & & k-1 & k \\ ? & ? & ? & \cdots & ? & ? \\ ? & 2 & 2 & \cdots & 2 & 2^* \end{array}. \tag{10}$$

The key observation is this: at all points between configurations (8) and (9), the switch-only cars and the trailing car together form a barrier that blocks both lanes. Let us make this a bit more formal. Classify the cars in (8) as follows:

$$\begin{array}{ccc|cccc|ccc} & j-2 & j-1 & j & j+1 & & k-1 & k & k+1 & \\ \cdots & \text{left} & \text{left} & \text{middle} & \text{middle} & \cdots & \text{middle} & \text{right} & \text{right} & \cdots \\ \cdots & \text{left} & \text{middle} & \text{right} & \text{right} & \cdots & \text{right} & \text{right} & \text{right} & \cdots \end{array}.$$

Now *left cars* cannot move beyond column $k-2$ until we reach configuration (9), while all *right cars* will be in columns $k, k+1, \ldots$ in configuration (9). The left and the right cars do not interact between (8) and (9); they are always separated by the *middle cars* (which do not move beyond column k).

Let us modify the solution as follows: we skip all moves related to left and middle cars between (8) and (9); only right cars are permitted to move. We will reach the following configuration instead of (9); note that we have cleared the part below the switch-only cars:

$$\begin{array}{c|cccc|c} j-1 & j & j+1 & & k-1 & k \\ ? & 2 & 2 & \cdots & 2 & ? \\ 2^* & \circ & \circ & \cdots & \circ & \circ \end{array}.$$

Then we move the rightmost switch-only car down and right:

$$\begin{array}{c|cccc|c} j-1 & j & j+1 & & k-1 & k \\ ? & 2 & 2 & \cdots & \circ & ? \\ 2^* & \circ & \circ & \cdots & \circ & 2 \end{array}.$$

We repeat this for each switch-only car in columns $k-2$, $k-3$, $\ldots$, j, and finally we move the trailing car right. We reach the following configuration:

$$\begin{array}{c|cccc|c} j-1 & j & j+1 & & k-1 & k \\ ? & \circ & \circ & \cdots & \circ & ? \\ \circ & 2^* & 2 & \cdots & 2 & 2 \end{array}.$$

Now we have handled right cars (following their original schedule) and middle cars (following a new schedule). Finally we perform all operations between (8) and (9) that were related to the left cars, and we reach a configuration like this:

$$\begin{array}{c|cccc|c} j-1 & j & j+1 & & k-1 & k \\ ? & ? & ? & \cdots & \circ & ? \\ ? & 2^* & 2 & \cdots & 2 & 2 \end{array}.$$

Then we continue with the operations after (9), skipping those related to the middle cars, and we reach configuration of the following form:

$$\begin{array}{c|cccc|c} j-1 & j & j+1 & & k-1 & k \\ ? & ? & ? & \cdots & ? & ? \\ ? & 2^* & 2 & \cdots & 2 & 2 \end{array}.$$

The only difference in comparison with (10) is that the trailing car is left in column j, and switch-only cars have moved right by one step. Hence none of them are bad any more.

Now let us see what we achieved. We constructed another solution in which one bad block of length L was eliminated. We performed L additional delay operations with the switch-only cars, but on the other hand we saved L delay operations with the trailing car; hence the new solution has the same cost as the original solution.

Concluding the proof. For each bad block we do either L or 0 additional delay operations, and the block already contained at least L delay and L switch operations. Summing over all bad blocks, if they contain D delay and W switch operations in total, we do at most $\min(D, W)$ additional delay operations. The claim related to the number of operations follows.

Finally, we observe that the modified solution has the same makespan as the original solution; note that if we have a bad block $[j, k)$, then the makespan of the solution has to be at least k, and we do not move any cars beyond column k. This concludes the proof of Lemma 5.

▶ **Corollary 6.** *Let Y be a solution for P1 that is simultaneously makespan-optimal and cost-optimal. Then Y is also a feasible makespan-optimal solution for P0. Furthermore, the total number of moves in Y is at most 1.5 times the total number of moves in a cost-optimal P0 solution.*

Proof. If the optimum cost of P0 is $W + D$ moves, by Lemma 5 there is a solution for P1 with at most $W + D + \min(D, W) \leq 1.5(W + D)$ moves. Hence the optimum of P1 is at most 1.5 times as expensive as the optimum of P0. By Lemma 5 we also have the same makespan. ◀

In particular, if can use algorithm A1 to find an optimal P1-solution Y, and then apply Corollary 6 to show that the solution is a good approximation also for P0.

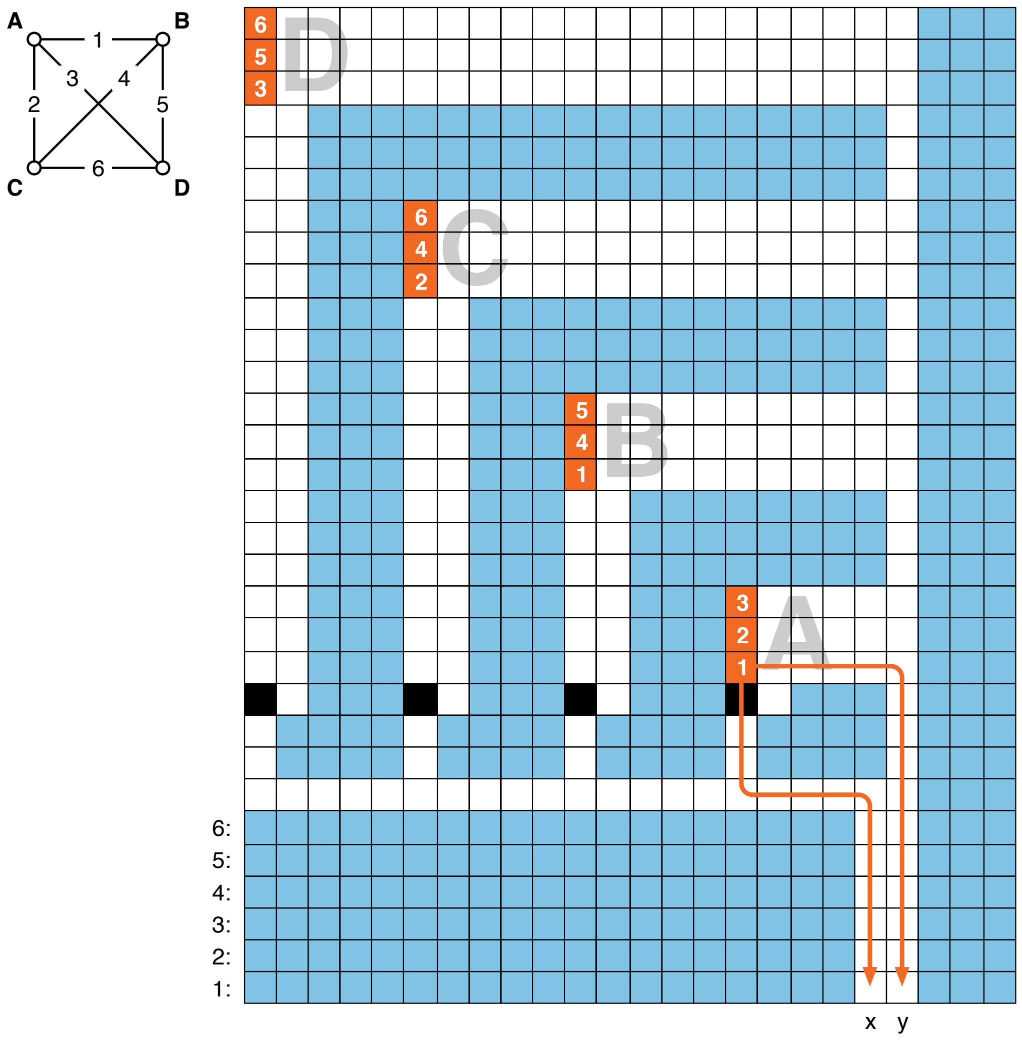


Figure 1 Reduction from the minimum vertex cover problem in 3-regular graphs. For each node (here labeled A, B, C, D) we construct a *cavity* that holds three orange cars, one per incident edge. Blue and black cells are cars that are already on their target lanes and hence they can only move right (delay). We label the edges arbitrarily with numbers $1, 2, \dots, m$; these correspond to the lowest m lanes. If edge number ℓ connects nodes u and v, then there is one orange car with label ℓ in cavity u and one orange car with label ℓ in cavity v. These will need to reach lane ℓ. There are two good routes, shown with orange arrows, one that takes the orange car to column x and one that takes the orange car to column y. To reach column x we will need to delay the black car that is blocking the way. One of the cars has to reach column x; hence for each edge $\{u, v\}$ we will need to move the black car in front of cavity u or cavity v. The set of cavities in which we have moved black cars forms a vertex cover; conversely, if we have a vertex cover of size k it is sufficient to move only k black cars. To complete the proof, one has to check that the blue "walls" are sufficiently thick so that any solution that involves moving blue car is strictly worse than a solution that only moves black and orange cars.

6 Hardness of the multi-lane version

To conclude this work, we will briefly look at what happens when we generalize the lane-changing problem from two lanes to multiple lanes. Assume that we have κ lanes, and the cars are labeled with targets $\{1, 2, \dots, \kappa\}$. The operations are a natural generalization of the two-lane case: we can *delay* car if there is empty slot after it, and we can move an agent sideways if there is empty space in an adjacent lane. We can only move agents sideways towards their target lane, not away from it.

We will now show that minimizing the total cost for this generalization is NP-hard; we only sketch the key ideas of the argument. The proof is by reduction from the minimum vertex cover problem in 3-regular graphs – this special case of the vertex cover problem is known to be NP-hard [5]. Given a 3-regular graph G with n nodes, we construct a multi-lane instance as shown in Figure 1. If and only if there is a vertex cover of size at most k for graph G, we can route the orange cars to their target lanes so that (1) none of the blue cars are moved, (2) exactly k black cars are delayed once. The construction has sufficiently thick "walls" formed by blue cars such that if we try to move blue cars to make space for orange cars, the cost will be higher than the solution obtained by the above strategy for the trivial vertex cover of size $k = n$.

References

1 Maksat Atagoziyev, Klaus W. Schmidt, and Ece G. Schmidt. Lane change scheduling for autonomous vehicles. In *Proc. 14th IFAC Symposium on Control in Transportation Systems (CTS 2016)*, volume 49(3) of *IFAC-PapersOnLine*, pages 61–66. Elsevier, 2016. doi:10.1016/j.ifacol.2016.07.011.

2 Wonshik Chee and Masayoshi Tomizuka. Vehicle lane change maneuver in automated highway systems. Research Report UCB-ITS-PRR-94-22, UC Berkeley, California Partners for Advanced Transportation Technology, 1994.

3 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2–3):207–216, 2005. doi:10.1016/j.tcs.2005.09.013.

4 Li Feng, Li Gao, and Yun-hui Li. Research on information processing of intelligent lane-changing behaviors for unmanned ground vehicles. In *Proc. 9th International Symposium on Computational Intelligence and Design (ISCID 2016)*, volume 2, pages 38–41. IEEE, 2016. doi:10.1109/ISCID.2016.2018.

5 G. H. Fricke, S. T. Hedetniemi, and D. P. Jacobs. Independence and irredundance in k-regular graphs. *Ars Combinatoria*, 49:271–2719, 1998.

6 Cem Hatipoğlu, Ümit Özgüner, and Konur A. Ünyelioğlu. On optimal design of a lane change controller. In *Proc. Intelligent Vehicles '95 Symposium*, pages 436–441. IEEE, 1995. doi:10.1109/IVS.1995.528321.

7 Ding-wei Huang. Lane-changing behavior on highways. *Physical Review E*, 66(2), 2002. doi:10.1103/PhysRevE.66.026124.

8 Wm. Woolsey Johnson and William E. Story. Notes on the "15" puzzle. *American Journal of Mathematics*, 2(4):397–404, 1879. doi:10.2307/2369492.

9 Jorge A. Laval and Carlos F. Daganzo. Lane-changing in traffic streams. *Transportation Research Part B: Methodological*, 40(3):251–264, 2006. doi:10.1016/j.trb.2005.04.003.

10 Kai Nagel. *High-Speed Microsimulations of Traffic Flow.* PhD thesis, University of Cologne, Germany, 1994.

11 Kai Nagel, Dietrich E. Wolf, Peter Wagner, and Patrice Simon. Two-lane traffic rules for cellular automata: A systematic approach. *Physical Review E*, 58(2), 1998. `doi:10.1103/PhysRevE.58.1425`.

12 José Eugenio Naranjo, Carlos González, Ricardo García, and Teresa de Pedro. Lane-change fuzzy control in autonomous vehicles for the overtaking maneuver. *IEEE Transactions on Intelligent Transportation Systems*, 9(3):438–450, 2008. `doi:10.1109/TITS.2008.922880`.

13 Daniel Ratner and Manfred K. Warmuth. Finding a shortest solution for the $n \times n$ extension of the 15-puzzle is intractable. In Tom Kehler, editor, *Proc. 5th National Conference on Artificial Intelligence (AAAI 1986)*, pages 168–172. Morgan Kaufmann, 1986.

14 Daniel Ratner and Manfred K. Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–138, 1990. `doi:10.1016/S0747-7171(08)80001-6`.

15 Robin Schubert, Karsten Schulze, and Gerd Wanielik. Situation assessment for automatic lane-change maneuvers. *IEEE Transactions on Intelligent Transportation Systems*, 11(3):607–616, 2010. `doi:10.1109/TITS.2010.2049353`.

16 F. Visintainer, L. Altomare, A. Toffetti, A. Kovacs, and A. Amditis. Towards manoeuver negotiation: AutoNet2030 project from a car maker perspective. In *Proc. 6th Transport Research Arena (TRA 2016)*, volume 14 of *Transportation Research Procedia*, pages 2237–2244. Elsevier, 2016. `doi:10.1016/j.trpro.2016.05.239`.

17 Richard M Wilson. Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory, Series B*, 16(1):86–96, 1974. `doi:10.1016/0095-8956(74)90098-7`.

18 Tay Wilson and W. Best. Driving strategies in overtaking. *Accident Analysis & Prevention*, 14(3):179–185, 1982. `doi:10.1016/0001-4575(82)90026-4`.

Parameterized Algorithms and Data Reduction for Safe Convoy Routing

René van Bevern
Department of Mechanics and Mathematics, Novosibirsk State University
Ulitsa Pirogova 2, 630090 Novosibirsk, Russian Federation

Sobolev Institute of Mathematics of the Siberian Branch of the Russian Academy of Sciences
Prospekt Akademika Koptyuga 4, 630090 Novosibirsk, Russian Federation
rvb@nsu.ru
https://orcid.org/0000-0002-4805-218X

Till Fluschnik
Institut für Softwaretechnik und Theoretische Informatik, TU Berlin
Ernst-Reuter-Platz 7, 10587 Berlin, Germany
till.fluschnik@tu-berlin.de

Oxana Yu. Tsidulko
Sobolev Institute of Mathematics of the Siberian Branch of the Russian Academy of Sciences
Prospekt Akademika Koptyuga 4, 630090 Novosibirsk, Russian Federation

Department of Mechanics and Mathematics, Novosibirsk State University
Ulitsa Pirogova 2, 630090 Novosibirsk, Russian Federation
tsidulko@math.nsc.ru

Abstract

We study a problem that models safely routing a convoy through a transportation network, where any vertex adjacent to the travel path of the convoy requires additional precaution: Given a graph $G = (V, E)$, two vertices $s, t \in V$, and two integers k, ℓ, we search for a simple s-t-path with at most k vertices and at most ℓ neighbors. We study the problem in two types of transportation networks: graphs with small crossing number, as formed by road networks, and tree-like graphs, as formed by waterways. For graphs with constant crossing number, we provide a subexponential $2^{O(\sqrt{n})}$-time algorithm and prove a matching lower bound. We also show a polynomial-time data reduction algorithm that reduces any problem instance to an equivalent instance (a so-called problem kernel) of size polynomial in the vertex cover number of the input graph. In contrast, we show that the problem in general graphs is hard to preprocess. Regarding tree-like graphs, we obtain a $2^{O(\mathrm{tw})} \cdot \ell^2 \cdot n$-time algorithm for graphs of treewidth tw, show that there is no problem kernel with size polynomial in tw, yet show a problem kernel with size polynomial in the feedback edge number of the input graph.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases NP-hard problem, fixed-parameter tractability, problem kernelization, shortest path, secluded solution

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.10

Related Version Full version available at arXiv:1806.09540.

Funding Till Fluschnik was supported by the German Research Foundation, grant NI 369/18. René van Bevern and Oxana Yu. Tsidulko were supported by the Russian Foundation for Basic Research (RFBR), grant 18-501-12031 NNIO_a, while working on Sections 2.2, 2.3, 3.3, and 3.2. While working on Sections 2.1 and 3.1, René van Bevern was supported by RFBR grant 16-31-

60007 mol_a_dk and Oxana Yu. Tsidulko was supported by the Ministry of Science and Higher Education of the Russian Federation under the 5-100 Excellence Programme.

Acknowledgements This research was initiated during a research visit of René van Bevern and Oxana Yu. Tsidulko at TU Berlin.

1 Introduction

Finding shortest paths is a fundamental problem in route planning and has extensively been studied with respect to efficient algorithms, including data reduction and preprocessing [1]. In this work, we study the following NP-hard variant of finding shortest s-t-paths.

▶ **Problem 1.1** (Short Secluded Path (SSP)).
Input: An undirected, simple graph $G = (V, E)$ with two distinct vertices $s, t \in V$, and two integers $k \geq 2$ and $\ell \geq 0$.
Question: Is there an s-t-path P in G such that $|V(P)| \leq k$ and $|N(V(P))| \leq \ell$?

Herein, $V(P)$ denotes the set of vertices on path P and $N(V(P))$ denotes their set of neighbors (not lying on P).

The problem can be understood as finding short and safe routes for a convoy through a transportation network: each neighbor of the convoy's travel path requires additional precaution. Thus, we seek to minimize not only the length of the convoy's travel path, but also its number of neighbors. In our work, we study the above basic, unweighted variant, as well as a weighted variant of the problem, in which each vertex has two weights: one counts towards the path length, the other models the cost of precaution that has to be taken when the vertex occurs as the neighbor of the travel path.

Almost planar and tree-like transportation networks. The focus of our work is two-fold. Firstly, since the problem is NP-hard, we search for efficient algorithms in graphs that are likely to occur as transportation networks: almost planar graphs, which occur as road networks, and tree-like graphs, which arise as waterways (ignoring the few man-made canals, natural river networks form forests [23]). Secondly, given the effect that preprocessing and data reduction had to fundamental routing problems like finding shortest paths [1], we study the possibilities of polynomial-time data reduction with *provable performance guarantees* for SSP.

In order to measure the running time of our algorithms with respect to the "degree of planarity" or the "tree-likeness" of a graph, as well as to analyze the power of data reduction algorithms, we employ parameterized complexity theory, which provides us with the concepts of *fixed-parameter algorithms* and *problem kernelization* [16, 20, 39, 12]. Fixed-parameter algorithms have recently been applied to numerous NP-hard routing problems [30, 28, 29, 27, 3, 42, 41, 4, 15, 5, 26]. In particular, they led to subexponential-time algorithms for fundamental NP-hard routing problems in planar graphs [33] and to algorithms for hard routing problems that work efficiently on real-world data [3].

Fixed-parameter algorithms. The main idea of fixed-parameter algorithms is to accept the exponential running time seemingly inherent to solving NP-hard problems, yet to restrict the combinatorial explosion to a parameter of the problem, which can be small in applications. We call a problem *fixed-parameter tractable* if it can be solved in $f(k) \cdot n^{O(1)}$ time on inputs of length n and some function f depending only on some parameter k. In contrast to an algorithm that merely runs in polynomial time for fixed k, fixed-parameter algorithms can solve NP-hard problems quickly if k is small.

Table 1 Overview of our results. Herein, n, tw, vc, fes, cr, and Δ denote the number of vertices, treewidth, vertex cover number, feedback edge number, the crossing number, and maximum degree of the input graph, respectively. "const." abbreviates "constant".

	On almost planar graphs (Sec. 2)	on tree-like graphs (Sec. 3)
exact solution	$2^{O(\sqrt{n})}$ time in graphs with const. cr (Thm. 2.1)	$2^{O(\mathrm{tw})} \cdot \ell^2 \cdot n$ time (Thm. 3.2)
problem kernel	size $\mathrm{vc}^{O(r)}$ in $K_{r,r}$-free graphs (Thm. 2.5)	size $\mathrm{fes}^{O(1)}$ (Thm. 3.13)
lower bounds	No kernel with size $\mathrm{poly}(\mathrm{vc} + r)$ in $K_{r,r}$-free graphs and WK[1]-hard when parameterized by $\mathrm{vc} + r$ (Thm. 2.14)	No kernel with size $\mathrm{poly}(\mathrm{tw} + k + \ell)$ even in planar graphs with const. Δ (Thm. 3.10)

Provably effective polynomial-time data reduction. Kernelization allows for provably effective polynomial-time data reduction. Note that a result of the form "our polynomial-time data reduction algorithm reduces the input size by at least one bit, preserving optimality of solutions" is impossible for NP-hard problems unless P = NP. In contrast, a kernelization algorithm reduces a problem instance into an equivalent one (the *problem kernel*) whose size depends only (ideally polynomially) on some problem parameter. Problem kernelization has been successfully applied to obtain effective polynomial-time data reduction algorithms for many NP-hard problems [25, 34] and also led to techniques for proving the limits of polynomial-time data reduction [7, 38, 9].

1.1 Our contributions

We study SSP (and a weighted variant) in two main classes of graphs: almost planar graphs and tree-like graphs. We refer to Table 1 for an overview on our main results. Regarding almost planar graphs, in graphs of constant crossing number, we show that (even the weighted version of) SSP is solvable in subexponential $2^{O(\sqrt{n})}$-time. Moreover, we prove that SSP is not solvable in $2^{o(\sqrt{n})}$-time in planar graphs unless the Exponential Time Hypothesis fails. In $K_{r,r}$-free graphs, which comprise the graphs with crossing number $O(r^3)$ [40], we show a problem kernel for SSP with size $\mathrm{vc}^{O(r)}$, where vc is the vertex cover number of the input graph. We prove that, unless the polynomial-time hierarchy collapses, there is no problem kernel of size polynomial in $\mathrm{vc} + r$. Moreover, we prove that, unless the classes FPT and WK[1] coincide, SSP does not even allow for *Turing kernels* with size polynomial in $\mathrm{vc} + r$; that is, we could not solve SSP in polynomial time even if we precomputed all answers to subproblems of size polynomial in $\mathrm{vc} + r$ and could look them up in constant time. Regarding tree-like graphs, we prove that SSP is solvable in $2^{O(\mathrm{tw})} \cdot \ell^2 \cdot n$ time in graphs of treewidth tw and that there is no problem kernel with size polynomial in tw. Instead, we show a problem kernel of size $\mathrm{fes}^{O(1)}$, where fes is the feedback edge number of the input graph.

Due to space constraints, results marked with ($\star$) are deferred to a full version of the paper.

1.2 Related work

Several classical graph optimization problems have been studied in the "secluded" (small closed neighborhood) and the "small secluded" (small set with small open neighborhood) variants [2]. Luckow and Fluschnik [37] first defined SSP and analyzed its parameterized complexity with respect to the parameters k and ℓ. In contrast, we study problem parameters that describe the structure of the input graphs and are small in transportation networks. Chechik et al. [11] introduced the SECLUDED PATH problem, that, given an undirected graph $G = (V, E)$ with two designated vertices $s, t \in V$, vertex-weights $w : V \to \mathbb{N}$, and two integers $k, C \in \mathbb{N}$, asks whether there is an s-t-path P such that the size of the *closed* neighborhood $|N[V(P)]| \leq k$ and the weight of the closed neighborhood $w(N[V(P)]) \leq C$. Fomin et al. [21], in particular, prove that SECLUDED PATH does not admit problem kernels with size polynomial in the vertex cover number vc. Our negative results on kernelization for SSP are significantly stronger: not only do we show that there is no problem kernel of size polynomial in vc$+r$ even in bipartite $K_{r,r}$-free graphs, we also show that SSP is WK[1]-hard parameterized by vc $+ r$. Golovach et al. [24] studied the "small secluded" scenario for finding connected induced subgraphs parameterized by the size ℓ of the open neighborhood. Their results obviously does not generalize to SSP, since SSP is NP-hard even for $\ell = 0$ [37].

1.3 Preliminaries

Graph Theory. We use basic notation from graph theory [14]. We study simple, finite, undirected graphs $G = (V, E)$. We denote by $V(G) := V$ the set of *vertices of* G and by $E(G) := E$ the set of *edges of* G. We denote $n := |V|$ and $m := |E|$. For any subset $U \subseteq V$ of vertices, we denote by $N_G(U) = \{w \in V \setminus U \mid \exists v \in U : \{v, w\} \in E\}$ the *open neighborhood* of U in G. When the graph G is clear from the context, we drop the subscript G. A set $U \subseteq V$ of vertices is a *vertex cover* if every edge in E has an endpoint in U. The size of a minimum vertex cover is called *vertex cover number vc*(G) of G. A set $F \subseteq E$ of edges is a *feedback edge set* if the graph $(G, E \setminus F)$ is a forest. The minimum size of a feedback edge set in a connected graph is $m - n + 1$ and is called the *feedback edge number fes*(G) of G. The *crossing number cr*(G) of G is the minimum number of crossings in any drawing of G into the plane (where only two edges are allowed to cross in each point). A path $P = (V, E)$ is a graph with vertex set $V = \{x_0, x_1, \ldots, x_p\}$ and edge set $E = \{\{x_i, x_{i+1}\} \mid 0 \leq i < p\}$. We say that P is an x_0-x_p-path of length p. We also refer to x_0, x_p as the *end points* of P, and to all vertices $V \setminus \{x_0, x_p\}$ as the *inner* vertices of P. A $K_{r,r}$ is a complete bipartite graph $G = (U \uplus V, E)$ with $|U| = |V| = r$. We say that a graph is *$K_{r,r}$-free* if it does not contain $K_{r,r}$ as a subgraph.

Parameterized Complexity Theory. For more details on parameterized complexity, we refer to the text books [16, 20, 39, 12]. Let Σ be a finite alphabet. A *parameterized problem* L is a subset $L \subseteq \Sigma^* \times \mathbb{N}$. An instance $(x, k) \in \Sigma^* \times \mathbb{N}$ is a *yes-instance* for L if and only if $(x, k) \in L$. We call x the *input* and k the *parameter*.

▶ **Definition 1.2** (fixed-parameter tractability, FPT)**.** A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is *fixed-parameter tractable* if there is a *fixed-parameter algorithm* deciding $(x, k) \in L$ in time $f(k) \cdot |x|^{O(1)}$. The complexity class *FPT* consists of all fixed-parameter tractable problems.

▶ **Definition 1.3** (kernelization)**.** Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. A *kernelization* is an algorithm that maps any instance $(x, k) \in \Sigma^* \times \mathbb{N}$ to an instance $(x', k') \in \Sigma^* \times \mathbb{N}$ in poly$(|x| + k)$ time such that

(i) $(x, k) \in L \iff (x', k') \in L'$, and
(ii) $|x'| + k' \leq f(k)$ for some computable function f.

We call (x', k') the *problem kernel* and f its *size*.

Basic observations. We may assume our input graph to be connected due to the following obviously correct and linear-time executable data reduction rule.

▶ **Reduction Rule 1.4.** If G has more than one connected component, then delete all but the component containing both s and t or return no if such a component does not exist.

2 Almost planar graphs

Many transportation networks such as rail and street networks are planar or at least have a small *crossing number* – the minimum number of edge crossings in a plane drawing of a graph. Unfortunately, SSP remains NP-hard even in planar graphs with maximum degree four and $\ell = 0$ [37].

In this section, we present algorithms for SSP in graphs with constant crossing number. These, in particular, apply to planar graphs. First, in Section 2.1, we present a subexponential-time algorithm and a matching lower bound. Second, in Section 2.2, we present a provably effective data reduction algorithm. Finally, in Section 2.3, we show the limits of data reduction algorithms for SSP in graphs with small but non-constant crossing number.

2.1 A subexponential-time algorithm

In this section, we describe how to solve SSP in subexponential time in graphs with constant crossing number.

▶ **Theorem 2.1.** Short Secluded Path *is solvable in* $2^{O(\sqrt{n})}$ *time on graphs with constant crossing number.*

We will also see a matching lower bound. To prove Theorem 2.1, we exploit that graphs with constant crossing number are H*-minor free* for some graph H.

▶ **Definition 2.2** (graph minor)**.** A graph H is a *minor* of a graph G if H can be obtained from G by a sequence of vertex deletions, edge deletions, and edge contractions. If a graph G does not contain H as a minor, then G is said to be H*-minor free.*

Bokal et al. [10] showed that, if a graph G contains $K_{r,r}$ as a minor, then the crossing number of G is $\operatorname{cr}(G) \geq \frac{1}{2}(r-2)^2$. Thus, any graph G is $K_{r,r}$-minor free for $r > \sqrt{2\operatorname{cr}(G)}+2$, which goes in line with the well-known fact that planar graphs are $K_{3,3}$-minor free [43]. Demaine and Hajiaghayi [13] showed that, for any graph H, all H-minor free graphs have treewidth $\text{tw} \in O(\sqrt{n})$.[1] To prove Theorem 2.1, it thus remains to show that SSP is solvable in $2^{O(\text{tw})} \cdot \text{poly}(n)$ time, which is the main technical work deferred to Section 3.1.

Complementing Theorem 2.1, we can show a matching lower bound using the Exponential Time Hypothesis (ETH).

▶ **Conjecture 2.3** (Exponential Time Hypothesis (ETH), Impagliazzo et al. [32])**.** *There is a constant c such that n-variable* 3-Sat *cannot be solved in* $2^{c(n+m)}$ *time.*

The ETH was introduced by Impagliazzo et al. [32] and since then has been used to prove running time lower bounds for various NP-hard problems (we refer to Cygan et al. [12, Chapter 14] for an overview). We use it to prove that Theorem 2.1 can be neither significantly improved in planar graphs nor generalized to general graphs.

▶ **Theorem 2.4.** *Unless the Exponential Time Hypothesis fails,* Short Secluded Path *has no* $2^{o(\sqrt{n})}$*-time algorithm in planar graphs and no* $2^{o(n+m)}$*-time algorithm in general.*

[1] In fact, they showed $\text{tw} \in O(\sqrt{q})$ for any graph parameter q that is $\Omega(p)$ on a $(\sqrt{p} \times \sqrt{p})$-grid and does not increase when taking minors. For example, the vertex cover number or feedback vertex number.

Proof. Assume that there is a $2^{o(\sqrt{n})}$-time algorithm for SSP in planar graphs and a $2^{o(n)}$-time algorithm for SSP in general graphs. Luckow and Fluschnik [37] give a polynomial-time many-one reduction from Hamiltonian Cycle to SSP that maintains planarity and increases the number of vertices and edges by at most a constant. Thus, we get a $2^{o(\sqrt{n})}$-time algorithm for Hamiltonian Cycle in planar graphs and a $2^{o(n+m)}$-time algorithm in general graphs. This contradicts ETH [12, Theorems 14.6 and 14.9]. ◀

2.2 Effective data reduction

In the previous section, we have shown a subexponential-time algorithm for SSP in graphs with constant crossing number. There, we exploited the fact that graphs with crossing number cr are $K_{r,r}$-minor free for $r > \sqrt{2\text{cr}} + 2$. Of course, this means that they neither contain $K_{r,r}$ as subgraph (indeed, one can show this even for $r \geq 3.145 \cdot \sqrt[3]{\text{cr}}$ using bounds from Pach et al. [40]).

In this section, we show how to reduce any instance of SSP in $K_{r,r}$-free graphs to an equivalent instance with size polynomial in the vertex cover number of the input graph. In the next section, we prove that this does not generalize to general graphs.

▶ **Theorem 2.5.** *For each constant $r \in \mathbb{N}$,* Short Secluded Path *in $K_{r,r}$-free graphs admits a problem kernel with size polynomial in the vertex cover number of the input graph.*

The proof of Theorem 2.5 consists of three steps. First, in linear time, we transform an n-vertex instance of SSP into an equivalent instance of an auxiliary vertex-weighted version of SSP with $O(\text{vc}^r)$ vertices. Second, using a theorem of Frank and Tardos [22], in polynomial time, we reduce the vertex weights to $2^{O(\text{vc}^{3r})}$ so that the total instances size (in bits) becomes $O(\text{vc}^{4r})$. Finally, since SSP is NP-complete in planar, and, hence, in $K_{3,3}$-free graphs, we can, in polynomial time, reduce the shrunk instance back to an instance of the unweighted SSP in $K_{r,r}$-free graphs. Due to the polynomial running time of the reduction, there is at most a polynomial blow-up of the instance size.

Our auxiliary variant of SSP allows each vertex to have two weights: one weight counts towards the length of the path, the other counts towards the number of neighbors:

▶ **Problem 2.6** (Vertex-Weighted Short Secluded Path (VW-SSP)).
Input: An undirected, simple graph $G = (V, E)$ with two distinct vertices $s, t \in V$, two integers $k \geq 2$ and $\ell \geq 0$, and vertex weights $\kappa : V \to \mathbb{N}$ and $\lambda : V \to \mathbb{N}$.
Question: Does G have an s-t-path P with $\sum_{v \in V(P)} \kappa(v) \leq k$ and $\sum_{v \in N(V(P))} \lambda(v) \leq \ell$?
Note that an instance of SSP can be considered to be an instance of VW-SSP with unit weight functions κ and λ. Our data reduction will be based on removing *twins*.

▶ **Definition 2.7** (twins). Two vertices u and v are called *(false) twins* if $N(u) = N(v)$.

As the first step towards proving Theorem 2.5, we will show that the following data reduction rule, when applied to a $K_{r,r}$-free instance of SSP for constant r, leaves us with an instance of VW-SSP with $O(\text{vc}^r)$ vertices.

▶ **Reduction Rule 2.8.** Let $(G, s, t, k, \ell, \kappa, \lambda)$ be an VW-SSP instance with unit weights, where $G = (V, E)$ is a $K_{r,r}$-free graph.

For each maximal set $U \subseteq V \setminus \{s, t\}$ of twins such that $|U| > r$, delete $|U| - r + 1$ vertices of U from G, and, for an arbitrary remaining vertex $v \in U$, set $\lambda(v) := |U| - r$ and $\kappa(v) := k + 1$.

▶ **Lemma 2.9** (⋆). *Reduction Rule 2.8 is correct and can be applied in linear time.*

We now prove a size bound for the instances remaining after Reduction Rule 2.8.

▶ **Proposition 2.10.** *Applied to an instance of* SSP *with a $K_{r,r}$-free graph with vertex cover number vc, Reduction Rules 2.8 and 1.4 yield an instance of* VW-SSP *on at most* $(vc+2)+r(vc+2)^r$ *vertices in linear time.*

Proof. Let $(G', s, t, k, \ell, \lambda', \kappa')$ be the instance obtained from applying Reduction Rules 2.8 and 1.4 to an instance $(G, s, t, k, \ell, \lambda, \kappa)$.

Let C be a minimum-cardinality vertex cover for G' that contains s and t, and let the vertex set of G' be $V = C \uplus Y$. Since G' is a subgraph of G, one has $|C| \leq \text{vc}(G') + 2 \leq \text{vc}(G) + 2 = \text{vc} + 2$. It remains to bound $|Y|$. To this end, we bound the number of vertices of degree at least r in Y and the number of vertices of degree exactly i in Y for each $i \in \{0, \ldots, r-1\}$. Note that vertices in Y have neighbors only in C.

Since Reduction Rule 1.4 has been applied, there are no vertices of degree zero in Y.

Since Reduction Rule 2.8 has been applied, for each $i \in \{1, \ldots, r-1\}$ and each subset $C' \subseteq C$ with $|C'| = i$, we find at most r vertices in Y whose neighborhood is C'. Thus, for each $i \in \{1, \ldots, r-1\}$, the number of vertices with degree i in Y is at most $r \cdot \binom{|C|}{i}$.

Finally, since G is $K_{r,r}$-free, any r-sized subset of the vertex cover C has at most $r-1$ common neighbors. Hence, since vertices in Y have neighbors only in C, the number of vertices in Y of degree greater or equal to r is at most $(r-1) \cdot \binom{|C|}{r}$. We conclude that

$$|V'| \leq |C| + (r-1) \cdot \binom{|C|}{r} + r \cdot \sum_{i=1}^{r-1} \binom{|C|}{i} \leq (\text{vc}+2) + r(\text{vc}+2)^r. \qquad \blacktriangleleft$$

This completes the first step of the proof of Theorem 2.5. Note that our data reduction works by "hiding" an unbounded number of twins in vertices of unbounded weights. The second step is thus reducing the weights of an VW-SSP instance. To this end, we are going to apply a theorem by Frank and Tardos [22], which was successfully applied in kernelizing weighted problems before [17].

▶ **Proposition 2.11** (Frank and Tardos [22]). *There is an algorithm that, on input $w \in \mathbb{Q}^d$ and integer N, computes in polynomial time a vector $\bar{w} \in \mathbb{Z}^d$ with $\|\bar{w}\|_\infty \leq 2^{4d^3} N^{d(d+2)}$ such that $\text{sign}(w^\top b) = \text{sign}(\bar{w}^\top b)$ for all $b \in \mathbb{Z}^d$ with $\|b\|_1 \leq N-1$, where*

$$\text{sign}(x) = \begin{cases} +1 & \textit{if } x > 0, \\ 0 & \textit{if } x = 0, \textit{ and} \\ -1 & \textit{if } x < 0. \end{cases}$$

▶ **Observation 2.12.** *For $N \geq 2$, Proposition 2.11 gives $\text{sign}(w^\top e_i) = \text{sign}(\bar{w}^\top e_i)$ for each $i \in \{1, \ldots, d\}$, where $e_i \in \mathbb{Z}^d$ is the vector that has 1 in the i-th coordinate and zeroes in the others. Thus, one has $\text{sign}(w_i) = \text{sign}(\bar{w}_i)$ for each $i \in \{1, \ldots, d\}$. That is, when reducing a weight vector from w to $\bar{w}$, Proposition 2.11 maintains the signs of weights.*

We apply Proposition 2.11 and Observation 2.12 to the weights of VW-SSP.

▶ **Lemma 2.13.** *An instance $I = (G, s, t, k, \ell, \lambda, \kappa)$ of* VW-SSP *on an n-vertex graph $G = (V, E)$ can be reduced in polynomial time to an instance $I' = (G, s, t, k', \ell', \lambda', \kappa')$ of* VW-SSP *such that*

i) *$\{k', \kappa'(v), \ell', \lambda'(v)\} \subseteq \{0, \ldots, 2^{4(n+1)^3} \cdot (n+2)^{(n+1)(n+3)}\}$, for each vertex $v \in V$, and*

ii) *I is a yes-instance if and only if I' is a yes-instance.*

Proof. In this proof, we will conveniently denote the weight functions $\lambda, \lambda', \kappa$, and κ' as vectors in $\mathbb{N}^n$ such that $\lambda_i = \lambda(i)$ for each $i \in V$, and similarly for the other weight functions.

We apply Proposition 2.11 with $d = n+1$ and $N = n+2$ to the vectors $(\lambda, \ell) \in \mathbb{N}^{n+1}$ and $(\kappa, k) \in \mathbb{N}^{n+1}$ to obtain vectors $(\kappa', k') \in \mathbb{Z}^{n+1}$ and $(\lambda', \ell') \in \mathbb{Z}^{n+1}$ in polynomial time.

(i) This follows from Proposition 2.11 with $d = n+1$ and $N = n+2$, and from Observation 2.12 since (λ, ℓ) and (κ, k) are vectors of nonnegative numbers.

(ii) Consider an arbitrary s-t-path P in G and two associated vectors $x, y \in \mathbb{Z}^n$, where

$$x_v = \begin{cases} 1 & \text{if } v \in N(V(P)), \\ 0 & \text{otherwise,} \end{cases} \qquad y_v = \begin{cases} 1 & \text{if } v \in V(P) \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Observe that $\|(x,-1)\|_1 \leq n+1$ and $\|(y,-1)\|_1 \leq n+1$. Since $n+1 \leq N-1$, Proposition 2.11 gives $\operatorname{sign}((\lambda,\ell)^\top(x,-1)) = \operatorname{sign}((\lambda',\ell')^\top(x,-1))$ and $\operatorname{sign}((\kappa,k)^\top(y,-1)) = \operatorname{sign}((\kappa',k')^\top(y,-1))$, which is equivalent to

$$\sum_{v \in N(V(P))} \lambda(v) \leq \ell \iff \sum_{v \in N(V(P))} \lambda'(v) \leq \ell' \qquad \text{and} \qquad \sum_{v \in P} \kappa(v) \leq k \iff \sum_{v \in P} \kappa'(v) \leq k'. \qquad \blacktriangleleft$$

We have finished two steps towards the proof of Theorem 2.5: we reduced SSP in $K_{r,r}$-free graphs for constant r to instances of VW-SSP with $O(\mathrm{vc}^r)$ vertices using Proposition 2.10 and shrunk its weights to encoding-length $O(\mathrm{vc}^{3r})$ using Lemma 2.13. To finish the proof of Theorem 2.5, it remains to reduce VW-SSP back to SSP on $K_{r,r}$-free graphs.

2.3 Limits of data reduction

In Section 2.2, we have seen that SSP allows for problem kernels with size polynomial in vc if the input graph is $K_{r,r}$-free for some constant r. A natural question is whether one can loosen the requirement of r being *constant*.

The following Theorem 2.14(i) shows that, under reasonable complexity-theoretic assumptions, this is not the case: we cannot get problem kernels whose size bound depends polynomially on both vc and r. Moreover, the following Theorem 2.14(ii) shows that, unless WK[1] = FPT, SSP does not even have Turing kernels with size polynomial in vc + r [31]. That is, we could not even solve SSP in polynomial time if we had precomputed all answers to SSP instances with size polynomial in vc + r and could look them up in constant time.

Both results come surprisingly: finding a standard shortest s-t-path is easy, whereas finding a short secluded path in general graphs is so hard that not even preprocessing helps.

▶ **Theorem 2.14** (⋆). *Even in bipartite graphs,* Short Secluded Path

i) *has no problem kernel with size polynomial in vc + r unless coNP ⊆ NP/poly and*
ii) *is WK*[1]*-hard when parameterized by vc + r,*

where vc is the vertex cover number of the input graph and r is the smallest number such that the input graph is $K_{r,r}$-free.

The proof exploits that Multicolored Clique is WK[1]-hard parameterized by $k \log n$ [31]:

▶ **Problem 2.15** (Multicolored Clique).
Input: A k-partite n-vertex graph $G = (V, E)$, where $V = \biguplus_{i=1}^k V_i$ for independent sets V_i.
Question: Does G contain a clique of size k?

We transfer the WK[1]-hardness of Multicolored Clique to SSP using the following type of reduction [31]:

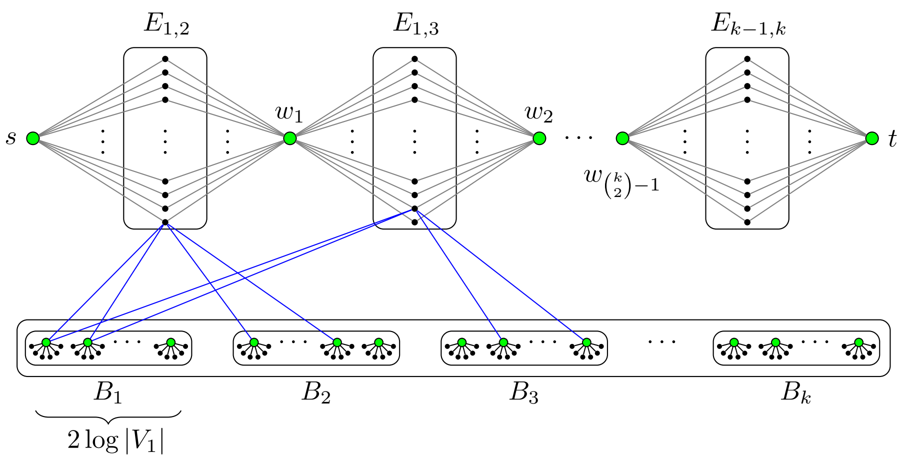


Figure 2.1 Illustration of the polynomial parameter transformation. Non-black (green) vertices indicate the vertices in the vertex cover.

▶ **Definition 2.16** (polynomial parameter transformation). Let $L, L' \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A *polynomial parameter transformation* from L to L' is an algorithm that maps any instance $(x, k) \in \Sigma^* \times \mathbb{N}$ to an instance $(x', k') \in \Sigma^* \times \mathbb{N}$ in $\text{poly}(|x|+k)$ time such that

(i) $(x, k) \in L \iff (x', k') \in L'$, and
(ii) $k' \leq \text{poly}(k)$.

Our polynomial parameter transformation of MULTICOLORED CLIQUE into SSP uses the following gadget.

▶ **Definition 2.17** (z-binary gadget). A *z-binary gadget* for some power z of two is a set $B = \{u_1, u_2, \dots, u_{(2\log z)}\}$ of vertices. We say that a vertex v is *p-connected to* B for some $p \in \{0, \dots, z-1\}$ if v is adjacent to $u_q \in B$ if and only if there is a "1" in position q of the string that consists of the binary encoding of p followed by its complement.

▶ **Example 2.18.** The binary encoding of 5 followed by its complement is 101010. Thus, a vertex v is 5-connected to an 8-binary gadget $\{u_1, \dots, u_6\}$ if and only if v is adjacent to u_1, u_3, and u_5. Also observe that, if a vertex v is q-connected to a z-binary gadget B, then v is adjacent to exactly half of the vertices of B, that is, to $\log z$ vertices of B.

The following reduction from MULTICOLORED CLIQUE to SSP is illustrated in Figure 2.1.

▶ **Construction 2.19.** Let $G = (V, E)$ be a MULTICOLORED CLIQUE instance, where $|V| = n$ and $V = V_1 \uplus V_2 \uplus \cdots \uplus V_k$. Without loss of generality, assume that $V_i = \{v_i^1, v_i^2, \dots, v_i^{\tilde{n}}\}$ for each $i \in \{1, \dots, k\}$, where $\tilde{n}$ is some power of two (we can guarantee this by adding isolated vertices to G). We construct an equivalent instance (G', s, t, k', ℓ') of SSP, where

$$k' := \binom{k}{2} + 1, \qquad \ell' := |E| - \binom{k}{2} + k \log \tilde{n},$$

and the graph $G' = (V', E')$ is as follows. The vertex set V' consists of vertices s, t, a vertex v_e for each edge $e \in E$, vertices w_h for $h \in \{1, \dots, \binom{k}{2} - 1\}$, and mutually disjoint $\tilde{n}$-binary vertex gadgets $B_1, \dots, B_k$, each vertex in which has $\ell' + 1$ neighbors of degree one. We denote

$$E^* := \{v_e \in V' \mid e \in E\} \qquad B := B_1 \uplus B_2 \uplus \cdots \uplus B_k,$$
$$E_{ij} := \{v_{\{x,y\}} \in E^* \mid x \in V_i, y \in V_j\}, \quad \text{and} \quad W := \{w_h \mid 1 \leq h \leq \tbinom{k}{2} - 1\}.$$

The edges of G' are as follows. For each edge $e = \{v_i^p, v_j^q\} \in E$, vertex $v_e \in E_{ij}$ of G' is p-connected to B_i and q-connected to B_j. Vertex $s \in V'$ is adjacent to all vertices in $E_{1,2}$ and vertex $t \in V'$ is adjacent to all vertices in $E_{k-1,k}$. Finally, to describe the edges incident to vertices in W, consider any ordering of pairs $\{(i,j) \mid 1 \leq i < j \leq k\}$. Then, vertex $w_h \in W$ is adjacent to all vertices in E_{ij} and to all vertices in $E_{i'j'}$, where (i,j) is the h-th pair in the ordering and (i',j') is the $(h+1)$-st. This finishes the construction.

To prove Theorem 2.14, we show that Construction 2.19 is a polynomial-time many-one reduction that generates bipartite $K_{r,r}$-free graphs with $r + \mathrm{vc} \in \mathrm{poly}(k \log n)$.

▶ **Lemma 2.20.** *The graph created by Construction 2.19 from an n-vertex instance $G = (V_1 \uplus V_2 \uplus \dots V_k, E)$ of* Multicolored Clique *is bipartite, $K_{r,r}$-free for $r := 2k \log n + \binom{k}{2} + 2$, and admits a vertex cover of size $r - 1$.*

Proof. The constructed graph $G' = (V', E')$ is bipartite with $V' = X \uplus Y$, where

$$X = \{s,t\} \cup W \cup B \qquad \text{and} \qquad Y = N(B) \cup E^*.$$

Hence, X is a vertex cover of size at most $r - 1$ in G'. Finally, consider any $K_{r,r}$ whose vertex set is partitioned into two independent sets $X' \uplus Y' \subseteq V'$. Since $|X'| = |Y'| = r$, $|X' \cap X| \leq r - 1$, and $|Y' \cap X| \leq r - 1$, we find $u \in X' \cap Y$ and $v \in Y' \cap Y$. Observe that $\{u, v\}$ is an edge in the $K_{r,r}$ but not in G'. Thus, the $K_{r,r}$ is not a subgraph of G'. ◀

▶ **Lemma 2.21.** *Construction 2.19 is a polynomial parameter transformation of* Multicolored Clique *parameterized by $k \log n$ into* SSP *in $K_{r,r}$-free graphs parameterized by $\mathrm{vc} + r$.*

Proof. Let $I' := (G', s, t, k', \ell')$ be the SSP instance created by Construction 2.19 from an Multicolored Clique instance $G = (V, E)$. In Lemma 2.20, we already showed $\mathrm{vc} + r \in \mathrm{poly}(k \log n)$. Thus, it remains to show that G is a yes-instance if and only if I' is.

($\Rightarrow$) Let C be the edge set of a clique of size k in G. For each $1 \leq i < j \leq k$, C contains exactly one edge e between V_i and V_j. Thus, $E_C := \{v_e \in E^* \mid e \in C\}$ is a set of $\binom{k}{2}$ vertices – exactly one vertex of E_{ij} for each $1 \leq i < j \leq k$. Thus, by Construction 2.19, G' contains an s-t-path $P = (V_P, E_P)$ with $|V_P| \leq k'$: its inner vertices are $E_C \cup W$, alternating between these two sets. To show that I' is a yes-instance, it remains to show $|N(V_P)| \leq \ell'$.

Since P contains all vertices of W, one has $N(V_P) \subseteq B \cup (E^* \setminus E_C)$, where $|E^* \setminus E_C| = |E| - \binom{k}{2}$. To show $|N(V_P)| \leq \ell'$, it remains to show that $|N(V_P) \cap B| \leq k \log \tilde{n}$. To this end, we show that $|N(V_P) \cap B_i| \leq \log \tilde{n}$ for each $i \in \{1, \dots, k\}$.

The vertices in $W \cup \{s,t\}$ have no neighbors in B. Thus, consider arbitrary vertices $v_{e_1}, v_{e_2} \in E_C$ such that $N(v_{e_1}) \cap B_i \neq \emptyset$ and $N(v_{e_2}) \cap B_i \neq \emptyset$ for some $i \in \{1, \dots, k\}$ (possibly, $e_1 = e_2$). Then, $e_1 = \{v_i^p, v_j^q\}$ and $e_2 = \{v_i^{p'}, v_{j'}^{q'}\}$. Since C is a clique, e_1 and e_2 are incident to the same vertex of V_i. Thus, we have $p = p'$. Both v_{e_1} and v_{e_2} are therefore p-connected to B_i and hence have the same $\log \tilde{n}$ neighbors in B_i. It follows that $N(V_P) \leq \ell'$ and, consequently, that I' is a yes-instance.

($\Leftarrow$) Let $P = (V_P, E_P)$ be an s-t-path in G' with $|V_P| \leq k'$ and $|N(V_P)| \leq \ell'$. The path P does not contain any vertex of B, since each of them has $\ell' + 1$ neighbors of degree one. Thus, the inner vertices of P alternate between vertices in W and in E^* and we get $N(V_P) = (E^* \setminus V_P) \cup (N(V_P) \cap B)$. Since P contains one vertex of E_{ij} for each $1 \leq i < j \leq k$, we know $|E^* \setminus V_P| = |E| - \binom{k}{2}$. Thus, since $|N(V_P)| \leq \ell'$, we have $|N(V_P) \cap B| \leq k \log \tilde{n}$. We exploit this to show that the set $C := \{e \in E \mid v_e \in V_P \cap E^*\}$ is the edge set of a clique in G. To this end, it is enough to show that, for each $i \in \{1, \dots, k\}$, any two edges $e_1, e_2 \in C$ with $e_1 \cap V_i \neq \emptyset$ and $e_2 \cap V_i \neq \emptyset$ have the same endpoint in V_i: then C is a set of $\binom{k}{2}$ edges on k vertices and thus forms a k-clique.

For each $1 \leq i < j \leq k$, P contains exactly one vertex $v \in E_{ij}$, which has exactly $\log \tilde{n}$ neighbors in each of B_i and B_j. Thus, from $|N(V_P) \cap B| \leq k \log \tilde{n}$ follows $|N(V_P) \cap B_i| = \log \tilde{n}$ for each $i \in \{1, \dots, k\}$. It follows that, if two vertices v_{e_1} and v_{e_2} on P both have neighbors in B_i, then both are p-connected to B_i for some p, which means that the edges e_1 and e_2 of G share endpoint v_i^p.

We conclude that C is the edge set of a clique of size k in G. Hence, G is a yes-instance. ◀

To prove Theorem 2.14, it is now a matter of putting together Lemma 2.21 and the fact that Multicolored Clique parameterized by $k \log n$ is WK[1]-complete.

Proof of Theorem 2.14. By Lemma 2.21, Construction 2.19 is a polynomial parameter transformation from Multicolored Clique parameterized by $k \log n$ to SSP parameterized by $\mathrm{vc} + r$ in $K_{r,r}$-free graphs.

Multicolored Clique parameterized by $k \log n$ is known to be WK[1]-complete [31] and hence, does not admit a polynomial-size problem kernel unless coNP $\subseteq$ NP/poly. From the polynomial parameter transformation in Construction 2.19, it thus follows that SSP is WK[1]-hard parameterized by $\mathrm{vc} + r$ and does not admit a polynomial-size problem kernel unless coNP $\subseteq$ NP/poly. ◀

3 Tree-like graphs

In this section, we present results for SSP in tree-like graphs. Such graphs naturally arise as waterways: when ignoring the few man-made canals, the remaining, natural waterways usually form a forest [23].

Moreover, graphs of small *treewidth* (formally defined in Section 3.1) are interesting since, as described in Section 2.1, graphs with constant crossing number have treewidth at most $\sqrt{q}$ for many graph parameters q. Thus, one can derive subexponential-time algorithms for these parameters from single-exponential algorithms for treewidth, like we did in Section 2.1.

First, in Section 3.1, we describe an algorithm that efficiently solves SSP on graphs of small treewidth. Second, in Section 3.2, we show that SSP allows for no problem kernel with size polynomial in the treewidth of the input graph. Third, in Section 3.3, we complement this negative result by a problem kernel with size polynomial in the feedback edge number of the input graph.

3.1 Fixed-parameter algorithm for graphs with small treewidth

In this section, we sketch a $2^{O(\mathrm{tw})} \cdot \ell^2 \cdot n$-time algorithm for SSP in graphs of treewidth tw, which will also conclude the proof of the $2^{O(\sqrt{n})}$-time algorithm for SSP in graphs with constant crossing number (Theorem 2.1). Before describing the algorithm, we formally introduce the treewidth concept.

▶ **Definition 3.1** (tree decomposition, treewidth)**.** A *tree decomposition* $\mathbb{T} = (T, \beta)$ of a graph $G = (V, E)$ consists of a tree T and a function $\beta \colon V(T) \to 2^V$ that associates each *node* x of the tree T with a subset $B_x := \beta(x) \subseteq V$, called a *bag*, such that

i) for each vertex $v \in V$, there is a node x of T with $v \in B_x$,
ii) for each edge $\{u, v\} \in E$, there is a node x of T with $\{u, v\} \subseteq B_x$,
iii) for each $v \in V$ the nodes x with $v \in B_x$ induce a subtree of T.

The *width* of $\mathbb{T}$ is $w(\mathbb{T}) := \max_{x \in V(T)} |B_x| - 1$. The *treewidth* of G is $\mathrm{tw}(G) := \min\{w(\mathbb{T}) \mid \mathbb{T}$ is a tree decomposition of $G\}$.

▶ **Theorem 3.2.** Short Secluded Path *is solvable in* $2^{O(tw)} \cdot \ell^2 \cdot n$ *time in graphs of treewidth tw.*

Bodlaender et al. [8] proved that a tree decomposition of width $O(\mathrm{tw}(G))$ of a graph G is computable in $2^{O(\mathrm{tw})} \cdot n$-time. Applying the following Proposition 3.3 to such a tree decomposition yields Theorem 3.2:

▶ **Proposition 3.3** ($\star$)**.** Vertex-Weighted Short Secluded Path *is solvable in* $n \cdot \ell^2 \cdot tw^{O(1)} \cdot (2 + 12 \cdot 2^{\omega})^{tw}$ *time when a tree decomposition of width tw is given, where* $\omega < 2.2373$ *is the matrix multiplication exponent.*

To prove Theorem 3.2, it thus remains to prove Proposition 3.3. Note that Proposition 3.3 actually solves the weighted problem VW-SSP (Problem 2.6), where the term ℓ^2 is only pseudo-polynomial for VW-SSP. It is a true polynomial for SSP since we can assume $\ell \le n$.

3.1.1 Assumptions on the tree decomposition

Our algorithm will work on simplified tree decompositions, which can be obtained from a classical tree decomposition of width tw in $n \cdot \mathrm{tw}^{O(1)}$ time without increasing its width [6].

▶ **Definition 3.4** (nice tree decomposition)**.** A *nice tree decomposition* $\mathbb{T}$ is a tree decomposition with one special bag r called *the root* and in which each bag is of one of the following types.

Leaf node: a leaf x of $\mathbb{T}$ with $B_x = \emptyset$.
Introduce vertex node: an internal node x of $\mathbb{T}$ with one child y such that $B_x = B_y \cup \{v\}$ for some vertex $v \notin B_y$. This node is said to *introduce vertex* v.
Introduce edge node: an internal node x of $\mathbb{T}$ labeled with an edge $\{u, v\} \in E$ and with one child y such that $\{u, v\} \subseteq B_x = B_y$. This node is said to *introduce edge* $\{u, v\}$.
Forget node: an internal node x of $\mathbb{T}$ with one child y such that $B_x = B_y \setminus \{v\}$ for some node $v \in B_y$. This node is said to *forget* v.
Join node: an internal node x of $\mathbb{T}$ with two children y and z such that $B_x = B_y = B_z$.

We additionally require that each edge is introduced at most once and make the following, problem specific assumptions on tree decompositions.

▶ **Assumption 3.5.** When solving VW-SSP, we will assume that the source s and destination t of the sought path are contained in all bags of the tree decomposition and that the root bag contains only s and t. This ensures that

- every bag contains vertices of the sought solution, and that
- s and t are never forgotten nor introduced.

Such a tree decomposition can be obtained from a nice tree decomposition by rooting it at a leaf (an empty bag) and adding s and t to all bags. This will increase the width of the tree decomposition by at most two.

Our algorithm will be based on computing partial solutions for subgraphs induced by a node of a tree decomposition by means of combining partial solutions for the subgraphs induced by its children. Formally, these subgraphs are the following.

▶ **Definition 3.6** (subgraphs induced by a tree decomposition)**.** Let $G = (V, E)$ be a graph and $\mathbb{T}$ be a nice tree decomposition for G with root r. Then, for any node x of $\mathbb{T}$,

$$V_x := \{v \in V \mid v \in B_y \text{ for a descendant } y \text{ of } x\}, \text{ and}$$
$$G_x := (V_x, E_x), \text{ where } E_x = \{e \in E \mid e \text{ is introduced in a descendant of } x\}.$$

Herein, we consider each node x of $\mathbb{T}$ to be a descendent of itself.

Having defined subgraphs induced by subtrees, we can define partial solutions in them.

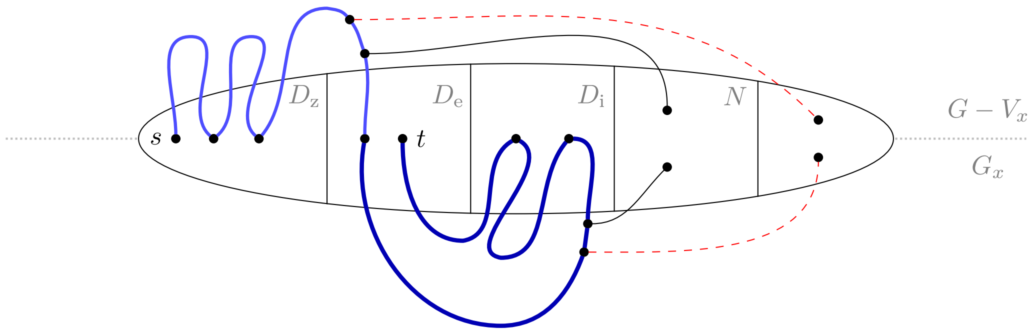


Figure 3.1 Illustration of a partial solution: the (blue) thick edges are an overall solution, where the darker edges are the part of the solution in G_x. The (red) dashed edges are forbidden to exist.

3.1.2 Partial solutions

Assume that we have a solution path P to VW-SSP. Then, the part of P in G_x is a collection $\mathcal{P}$ of paths (some might consist of a single vertex). When computing a partial solution for a parent y of x, we ideally want to check which partial solutions for x can be continued to partial solutions for y. However, we cannot try all possible partial solutions for G_x – there might be too many. Moreover, this is not necessary: by Definition 3.1(ii)–(iii), vertices in bag B_y cannot be vertices of and cannot have edges to vertices of $V_x \setminus B_x$. Thus, it is enough to know the states of vertices in bag B_x in order to know which partial solutions of x can be continued to y. The state of such vertices is characterized by

- which vertices of B_x are end points of paths in $\mathcal{P}$, inner vertices of paths in $\mathcal{P}$, or paths of zero length in $\mathcal{P}$,
- which vertices of B_x are allowed to be neighbors of the solution path P,
- how many neighbors the solution path P is allowed to have in G_x, and
- which vertices of B_x belong to the same path of $\mathcal{P}$.

▶ **Definition 3.7** (partial solution)**.** Let $(G, s, t, k, \ell, \kappa, \lambda)$ be an instance of VW-SSP. For a set $\mathcal{P}$ of paths in G and a set N of vertices in G, let

$$\Lambda(\mathcal{P}, N) := \sum_{P\in\mathcal{P}} \sum_{v\in N(V(P))} \lambda(v) + \sum_{v\in N} \lambda(v) \qquad \text{and} \qquad K(\mathcal{P}) := \sum_{P\in\mathcal{P}} \sum_{v\in V(P)} \kappa(v).$$

Moreover, let $\mathbb{T}$ be a tree decomposition for G, x be a node of $\mathbb{T}$, $D_{\mathrm{z}} \uplus D_{\mathrm{e}} \uplus D_{\mathrm{i}} \uplus N \subseteq B_x$ such that $\{s, t\} \subseteq D_{\mathrm{z}} \cup D_{\mathrm{e}}$, p be a partition of $D := D_{\mathrm{z}} \cup D_{\mathrm{e}} \cup D_{\mathrm{i}}$, and $l \leq \ell$.

Then, we call $(D_{\mathrm{z}}, D_{\mathrm{e}}, D_{\mathrm{i}}, N, l)$ a *pre-signature* and $S = (D_{\mathrm{z}}, D_{\mathrm{e}}, D_{\mathrm{i}}, N, l, p)$ a *solution signature* at x. A set $\mathcal{P}$ of paths in G_x is a *partial solution* of *cost* $K(\mathcal{P})$ for S if

i) D_{z} are exactly the vertices of zero-length paths $P \in \mathcal{P}$,
ii) D_{e} are exactly the end points of non-zero-length paths $P \in \mathcal{P}$,
iii) D_{i} are exactly those vertices in B_x that are inner vertices of paths $P \in \mathcal{P}$,
iv) for each path $P \in \mathcal{P}$, $N(V(P)) \cap B_x \subseteq N$,
v) $\Lambda(\mathcal{P}, N) \leq l$, and
vi) $\mathcal{P}$ consists of exactly $|p|$ paths such that each two vertices $u, v \in D$ belong to the same path of $\mathcal{P}$ if and only if they are in the same set of the partition p.

For a solution signature S at a node x, we denote

$$\mathcal{E}_x(S) := \{\mathcal{P} \mid \mathcal{P} \text{ is a partial solution for } S\},$$
$$\mathrm{minK}_x(S) := \min\{K(\mathcal{P}) \mid \mathcal{P} \in \mathcal{E}_x(S)\}.$$

Because of Assumption 3.5, our input instance to VW-SSP is a yes-instance if and only if

$$\operatorname{minK}_r(\emptyset, \{s,t\}, \emptyset, \emptyset, \ell, \{\{s,t\}\}) \le k. \tag{3.1}$$

Therefore, our aim is computing this cost. The naive dynamic programming approach is:

- compute $\operatorname{minK}_x(S)$ for each solution signature S and each leaf node x,
- compute $\operatorname{minK}_x(S)$ for each solution signature S and each inner node x under the assumptions that $\operatorname{minK}_y(S')$ has already been computed for all solution signatures S' at children y of x.

However, this approach is not suitable to prove Proposition 3.3, since the number of possible solution signatures is too large: the number of different partitions p of tw vertices is the tw-th Bell number, whose best known upper bound is $O(\mathrm{tw}^{\mathrm{tw}}/\log \mathrm{tw})$.

3.1.3 Reducing the number of partitions

To reduce the number of needed partitions, we use an approach developed by Bodlaender et al. [6], which also proved its effectivity in experiments [18]. We will replace the task of computing (3.1) for all possibly partitions by computing only sets of *weighted partitions* containing the needed information.

▶ **Definition 3.8** (sets of weighted partitions). Let $\Pi(U)$ be the set of all partitions of U. A *set of weighted partitions* is a set $\mathcal{A} \subseteq \Pi(U) \times \mathbb{N}$. For a *weighted partition* $(p, w) \in \mathcal{A}$, we call w its *weight*.

Using sets of weighted partitions, we can reformulate our task of computing $\operatorname{minK}_x(S)$ for all bags B_x and all solution signatures S as follows. Consider a pre-signature $S = (D_{\mathrm{z}}, D_{\mathrm{i}}, D_{\mathrm{e}}, N, l)$ for a node x of a tree decomposition. Then, for each $p \in \Pi(D_{\mathrm{z}} \cup D_{\mathrm{i}} \cup D_{\mathrm{e}})$, (S, p) is a solution signature. Thus, we can consider

$$\mathcal{A}_x(S) := \left\{ \left(p, \min_{\mathcal{P} \in \mathcal{E}_x(S,p)} K(\mathcal{P})\right) \,\middle|\, p \in \Pi(D_{\mathrm{z}} \cup D_{\mathrm{i}} \cup D_{\mathrm{e}}) \wedge \mathcal{E}_x(S,p) \neq \emptyset \right\}. \tag{3.2}$$

Now, our problem of verifying (3.1) at the root node r of a tree decomposition is equivalent to checking whether $\mathcal{A}_r(\emptyset, \{s,t\}, \emptyset, \emptyset, \ell)$ contains a partition $\{\{s,t\}\}$ of weight at most k. Thus we can, in a classical dynamic programming manner

- compute $\mathcal{A}_x(S)$ for each pre-signature S and each leaf node x,
- compute $\mathcal{A}_x(S)$ for each pre-signature S and each inner node x under the assumption that $\mathcal{A}_y(S')$ has already been computed for all pre-signatures S' at children y of x.

Yet we will not work with the full sets $A_x(S)$ but with "representative" subsets of size $2^{O(\mathrm{tw})}$. Since the number of pre-signatures is $2^{O(\mathrm{tw})} \cdot \ell$, this will allow us to prove Proposition 3.3.

In order to describe the intuition behind representative sets of weighted partitions, we need some notation.

▶ **Definition 3.9** (partition lattice). The set $\Pi(U)$ is semi-ordered by the coarsening relation $\sqsubseteq$, where $p \sqsubseteq q$ if every set of p is included in some set of q. We also say that q is *coarser* than p and that p is *finer* than q.

For two partitions $p, q \in \Pi(U)$, by $p \sqcup q$ we denote the (unique) finest partition that is coarser than both p and q.

To get an intuition for the $p \sqcup q$ operation, recall from Definition 3.7 that we will use a partition p to represent connected components of partial solutions: two vertices are connected if and only if they are in the same set of p. In these terms, if $p \in \Pi(U)$ are the vertex sets of the connected components of a graph (U, E) and $q \in \Pi(U)$ are the vertex sets of the connected components of a graph (U, E'), then $p \sqcup q$ are the vertex sets of the connected components of the graph $(U, E \cup E')$.

Now, assume that there is a solution P to VW-SSP in a graph G and consider an arbitrary node x of a tree decomposition. Then, the subpaths $\mathcal{P}$ of P that lie in G_x are a partial solution for some solution signature (D_z, D_e, D_i, N, l, p) at x. The partition p of $D := D_z \cup D_e \cup D_i$ consists of the sets of vertices of D that are connected by paths in $\mathcal{P}$. Since, in the overall solution P, the vertices in D are all connected, the vertices of D are connected in $G \setminus E_x$ according to a partition q of D such that $p \sqcup q = \{D\}$. Now, if in P, we replace the subpaths $\mathcal{P}$ by any other partial solution $\mathcal{P}'$ to a solution signature $(D_z, D_e, D_i, N, l, p')$ such that $K(\mathcal{P}') \leq K(\mathcal{P})$ and $p' \sqcup q = \{D\}$, then we obtain a solution P' for G with at most the cost of P. Thus, one of the two weighted partitions $(p, K(p))$ and $(p', K(p'))$ in $\mathcal{A}_x(D_z, D_e, D_i, N, l)$ is redundant.

This concept of redundancy can be formalized as *representative sets* and representative sets of size $2^{O(\text{tw})}$ can be efficiently computed using results of Bodlaender et al. [6]. To prove Proposition 3.3, it is enough to derive a recurrence relation for (3.2) that plays well together with the framework of Bodlaender et al. [6].

3.2 Hardness of kernelization for graphs of small treewidth

In the previous section, we have seen that SSP is efficiently solvable in tree-like graphs, namely, in graphs of small treewidth. We can complement this result as follows.

▶ **Theorem 3.10** ($\star$). Short Secluded Path *has no problem kernel with size polynomial in tw* $+ k + \ell$*, even on planar graphs with maximum degree six, where tw is the treewidth, unless coNP* $\subseteq$ *NP/poly and the polynomial-time hierarchy collapses to the third level.*

To prove Theorem 3.10, we use a special kind of reduction called *cross composition* [9].

▶ **Definition 3.11** (cross composition)**.** A *polynomial equivalence relation* $\sim$ is an equivalence relation over Σ^* such that

- there is an algorithm that decides $x \sim y$ in polynomial time for any two instances $x, y \in \Sigma^*$, and such that
- the number of equivalence classes of $\sim$ over any *finite* set $S \subseteq \Sigma^*$ is polynomial in $\max_{x \in S} |x|$.

A language $K \subseteq \Sigma^*$ *cross-composes* into a parameterized language $L \subseteq \Sigma^* \times \mathbb{N}$ if there is a polynomial-time algorithm, called *cross composition*, that, given a sequence $x_1, \dots, x_p$ of p instances that are equivalent under some polynomial equivalence relation, outputs an instance (x^*, k) such that

- k is bounded by a polynomial in $\max_{i=1}^{p} |x_i| + \log p$ and
- $(x^*, k) \in L$ if and only if there is an $i \in \{1, \dots, p\}$ such that $x_i \in K$.

Cross compositions can be used to rule out problem kernels of polynomial size using the following result of Bodlaender et al. [9].

▶ **Proposition 3.12** (Bodlaender et al. [9])**.** *If a NP-hard language* $K \subseteq \Sigma^*$ *cross-composes into the parameterized language* $L \subseteq \Sigma^* \times \mathbb{N}$*, then there is no polynomial-size problem kernel for* L *unless coNP* $\subseteq$ *NP/poly and the polynomial-time hierarchy collapses to the third level.*

Using a cross composition, Luckow and Fluschnik [37] proved that SSP on planar graphs of maximum degree six does not admit a problem kernel with size polynomial in $k + \ell$. To prove Theorem 3.10, one can show that the graph created by their cross composition has treewidth at most $3n + 3$, where n is the number of vertices in each input instance to their cross composition.

3.3 Effective data reduction for graphs with small feedback edge set

In the previous section, we have seen that SSP has no problem kernel with size polynomial in the treewidth of the input graph. We can complement this result by proving a polynomial-size problem kernel for another parameter that measures the tree-likeness of a graph: the *feedback edge number* of a graph is the smallest number of edges one has to delete to obtain a forest. Formally, we can prove the following theorem.

▶ **Theorem 3.13** ($\star$). Short Secluded Path *has a problem kernel with size polynomial in the feedback edge number of the input graph.*

The outline of the proof of Theorem 3.13 is similar that of the proof of Theorem 2.5: we first, in linear time, produce a weighted instance with $O(\text{fes})$ vertices, then reduce the weights using Lemma 2.13, and finally transform the weighted instance back into an instance for SSP.

Towards the first step, we apply data reduction rules that reduce the number of degree-one vertices and the length of paths of degree-two vertices to $O(\text{fes})$. Because of Reduction Rule 1.4, the graph without the fes edges of a feedback edge set is a tree. Thus, its overall number of vertices and edges will be bounded by $O(\text{fes})$.

4 Conclusion

Concluding, we point out that our algorithms for VW-SSP on graphs of bounded treewidth (Theorem 3.2) can easily be generalized to a problem variant where also edges have a weight counting towards the path length, and so can our subexponential-time algorithms in planar graphs (Theorem 2.1). Moreover, the technique of Bodlaender et al. [6] that our algorithm is based on has experimentally been proven to be practically implementable [18].

In contrast, we observed SSP to be a problem for which provably effective polynomial-time data reduction is rather hard to obtain (Theorems 2.14 and 3.10). Therefore, studying relaxed models of data reduction with performance guarantees like approximate [36, 19] or randomized kernelization [35] seems worthwhile.

Indeed, our few positive results on kernelization, that is, our problem kernels of size $\text{vc}^{O(r)}$ in $K_{r,r}$-free graphs and of size $\text{fes}^{O(1)}$ in graphs of feedback edge number fes for SSP (Theorems 2.5 and 3.13), for now, can be mainly seen as a proof of concept, since they employ the quite expensive weight reduction algorithm of Frank and Tardos [22] and we have no "direct" way of reducing VW-SSP back to SSP. On the positive side, our solution algorithms also work for VW-SSP, so that reducing weights or reducing back to SSP may be unnecessary from a practical point of view.

References

1 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. In *Algorithm Engineering: Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016. doi:10.1007/978-3-319-49487-6_2.

2 René van Bevern, Till Fluschnik, George B. Mertzios, Hendrik Molter, Manuel Sorge, and Ondřej Suchý. The parameterized complexity of finding secluded solutions to some classical optimization problems on graphs. *Discrete Optimzation*, 2018. In press, available on arXiv:1606.09000v5. doi:10.1016/j.disopt.2018.05.002.

3 René van Bevern, Christian Komusiewicz, and Manuel Sorge. A parameterized approximation algorithm for the mixed and windy capacitated arc routing problem: Theory and experiments. *Networks*, 70(3):262–278, 2017. WARP 2 special issue. doi:10.1002/net.21742.

4 René van Bevern, Rolf Niedermeier, Manuel Sorge, and Mathias Weller. Complexity of arc routing problems. In *Arc Routing: Problems, Methods, and Applications*, volume 20 of *MOS-SIAM Series on Optimization*. SIAM, 2014. doi:10.1137/1.9781611973679.ch2.

5 Hans-Joachim Böckenhauer, Juraj Hromkovič, Joachim Kneis, and Joachim Kupke. The parameterized approximability of TSP with deadlines. *Theory of Computing Systems*, 41(3):431–444, 2007. doi:10.1007/s00224-007-1347-x.

6 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

7 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

8 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and MichałPilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016. doi:10.1137/130947374.

9 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014. doi:10.1137/120880240.

10 Drago Bokal, Gasper Fijavz, and Bojan Mohar. The minor crossing number. *SIAM Journal on Discrete Mathematics*, 20(2):344–356, 2006. doi:10.1137/05062706X.

11 Shiri Chechik, Matthew P. Johnson, Merav Parter, and David Peleg. Secluded connectivity problems. *Algorithmica*, 79(3):708–741, 2017. doi:10.1007/s00453-016-0222-z.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 Erik D. Demaine and Mohammadtaghi Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008. doi:10.1007/s00493-008-2140-4.

14 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 4th edition, 2010. doi:10.1007/978-3-662-53622-3.

15 Frederic Dorn, Hannes Moser, Rolf Niedermeier, and Mathias Weller. Efficient algorithms for Eulerian Extension and Rural Postman. *SIAM Journal on Discrete Mathematics*, 27(1):75–94, 2013. doi:10.1137/110834810.

16 Rod G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

17 Michael Etscheid, Stefan Kratsch, Matthias Mnich, and Heiko Röglin. Polynomial kernels for weighted problems. *Journal of Computer and System Sciences*, 84(Supplement C):1–10, 2017. doi:10.1016/j.jcss.2016.06.004.

18 Stefan Fafianie, Hans L. Bodlaender, and Jesper Nederlof. Speeding up dynamic programming with representative sets: An experimental evaluation of algorithms for steiner tree on tree decompositions. *Algorithmica*, 71(3):636–660, 2015. doi:10.1007/s00453-014-9934-0.

19 Michael R. Fellows, Ariel Kulik, Frances A. Rosamond, and Hadas Shachnai. Parameterized approximation via fidelity preserving transformations. *Journal of Computer and System Sciences*, 93:30–40, 2018. doi:10.1016/j.jcss.2017.11.001.

20 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. doi:10.1007/3-540-29953-X.

21 Fedor V. Fomin, Petr A. Golovach, Nikolay Karpov, and Alexander S. Kulikov. Parameterized complexity of secluded connectivity problems. *Theory of Computing Systems*, 61(3):795–819, 2017. doi:10.1007/s00224-016-9717-x.

22 András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987. doi:10.1007/BF02579200.

23 Achille Giacometti. River networks. In *Complex Networks*, Encyclopedia of Life Support Systems (EOLSS), pages 155–180. EOLSS Publishers/UNESCO, 2010.

24 Petr A. Golovach, Pinar Heggernes, Paloma T. Lima, and Pedro Montealegre. Finding connected secluded subgraphs. In *Proceedings of the 12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, volume 89 of *LIPIcs*, pages 18:1–18:13. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.18.

25 Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007. doi:10.1145/1233481.1233493.

26 G. Gutin and V. Patel. Parameterized traveling salesman problem: Beating the average. *SIAM Journal on Discrete Mathematics*, 30(1):220–238, 2016. doi:10.1137/140980946.

27 Gregory Gutin, Mark Jones, and Bin Sheng. Parameterized complexity of the k-arc chinese postman problem. *Journal of Computer and System Sciences*, 84:107–119, 2017. doi:10.1016/j.jcss.2016.07.006.

28 Gregory Gutin, Mark Jones, and Magnus Wahlström. The mixed chinese postman problem parameterized by pathwidth and treedepth. *SIAM Journal on Discrete Mathematics*, 30(4):2177–2205, 2016. doi:10.1137/15M1034337.

29 Gregory Gutin, Gabriele Muciaccia, and Anders Yeo. Parameterized complexity of k-Chinese Postman Problem. *Theoretical Computer Science*, 513:124–128, 2013. doi:10.1016/j.tcs.2013.10.012.

30 Gregory Gutin, Magnus Wahlström, and Anders Yeo. Rural Postman parameterized by the number of components of required edges. *Journal of Computer and System Sciences*, 83(1):121–131, 2017. doi:10.1016/j.jcss.2016.06.001.

31 Danny Hermelin, Stefan Kratsch, Karolina Sołtys, Magnus Wahlström, and Xi Wu. A completeness theory for polynomial (turing) kernelization. *Algorithmica*, 71(3):702–730, 2015. doi:10.1007/s00453-014-9910-8.

32 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

33 Philip N. Klein and Daniel Marx. A subexponential parameterized algorithm for Subset TSP on planar graphs. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, pages 1812–1830. Society for Industrial and Applied Mathematics, 2014. doi:10.1137/1.9781611973402.131.

34 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/285.

35 Stefan Kratsch and Magnus Wahlström. Compression via matroids: A randomized polynomial kernel for odd cycle transversal. *ACM Transactions on Algorithms*, 10(4):20:1–20:15, 2014. doi:10.1145/2635810.

36 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2017)*, pages 224–237. ACM, 2017. doi:10.1145/3055399.3055456.

37 Max-Jonathan Luckow and Till Fluschnik. On the computational complexity of length- and neighborhood-constrained path problems. Available on arXiv:1808.02359, 2018.

38 Neeldhara Misra, Venkatesh Raman, and Saket Saurabh. Lower bounds on kernelization. *Discrete Optimization*, 8(1):110–128, 2011. doi:10.1016/j.disopt.2010.10.001.

39 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. doi:10.1093/acprof:oso/9780198566076.001.0001.

40 Janos Pach, Rados Radoicic, Gabor Tardos, and Geza Toth. Improving the crossing lemma by finding more crossings in sparse graphs. *Discrete & Computational Geometry*, 36(4):527–552, 2006. doi:10.1007/s00454-006-1264-9.

41 Manuel Sorge, René van Bevern, Rolf Niedermeier, and Mathias Weller. From few components to an Eulerian graph by adding arcs. In *Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'11)*, volume 6986 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2011. doi:10.1007/978-3-642-25870-1_28.

42 Manuel Sorge, René van Bevern, Rolf Niedermeier, and Mathias Weller. A new view on Rural Postman based on Eulerian Extension and Matching. *Journal of Discrete Algorithms*, 16:12–33, 2012. doi:10.1016/j.jda.2012.04.007.

43 K. Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114(1):570–590, 1937. doi:10.1007/BF01594196.

A Neighborhood Search and Set Cover Hybrid Heuristic for the Two-Echelon Vehicle Routing Problem

Youcef Amarouche[1]
Sorbonne universités, Université de technologie de Compiègne, CNRS, Heudiasyc UMR 7253
CS 60 319, 60 203 Compiègne cedex, France
youcef.amarouche@hds.utc.fr

Rym N. Guibadj
LISIC, Laboratoire d'Informatique Signal et Image de la Côte d'Opale, ULCO, Université Lille Nord-de-France, France
rym.guibadj@univ-littoral.fr

Aziz Moukrim
Sorbonne universités, Université de technologie de Compiègne, CNRS, Heudiasyc UMR 7253
CS 60 319, 60 203 Compiègne cedex, France
aziz.moukrim@hds.utc.fr

Abstract

The Two-Echelon Vehicle Routing Problem (2E-VRP) is a variant of the classical vehicle routing problem arising in the context of city logistics. In the 2E-VRP, freight from a main depot is delivered to final customers using intermediate facilities, called satellites. In this paper, we propose a new hybrid heuristic method for solving the 2E-VRP that relies on two components. The first component effectively explores the search space in order to discover a set of interesting routes. The second recombines the discovered routes into high-quality solutions. Experimentations on benchmark instances show the performance of our approach: our algorithm achieves high-quality solutions in short computational times and improves the current best known solutions for several large scale instances.

2012 ACM Subject Classification Mathematics of computing → Optimization with randomized search heuristics, Applied computing → Transportation

Keywords and phrases Two-Echelon Vehicle Routing Problem, City Logistics, hybrid method, integer programming

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.11

Funding This work is supported by the French Environment and Energy Management Agency (ADEME), the Hauts-de-France region and the European Regional Development Fund (ERDF). It was carried out within the framework of the Labex MS2T, funded by the French Government (Reference ANR-11-IDEX-0004-02). It is also partially supported by TCDU project (Collaborative Transportation in Urban Distribution, ANR-14-CE22-0017).

[1] Agence de l'environnement et de la Maîtrise de l'Energie 20, avenue du Grésillé - BP 90406, 49004, Angers Cedex 01, France

1 Introduction

Freight transportation is a key factor underpinning economic growth. However, it is also a major nuisance, especially in urban areas where congestion and environmental effects disturb people's well-being. As demand for freight transportation increases, new transport policies and better traffic management become essential to limit its effects. The concept of city logistics is one approach to solving the problem. It aims to optimize freight transportation within city areas while considering traffic congestion and environmental issues as well as costs and benefits to the freight shippers [18]. Some of the most used models in city logistics are multi-echelon distribution systems, especially two-echelon systems.

In a two-echelon distribution system, delivery from one or more depots to the customers is managed by shipping and consolidating freight through intermediate depots called *satellites*. Freight is first moved from the depots to the satellites using large trucks. Then, freight is delivered from the satellites to the customers using smaller vehicles. Proceeding like this allows to shape more conveniently the fleet of vehicles to be used, as larger trucks are more cost efficient whereas smaller ones are preferable in city centers. Because the flow of freight in each echelon depends on that in the other echelon, routing problems arising in two-echelon distribution systems must be studied as a whole; they cannot be merely decomposed into two separate sub-problems. The problem that studies how to efficiently route freight in such systems is known as the *Two-Echelon Vehicle Routing Problem (2E-VRP)*.

In this paper, we consider the basic version of the 2E-VRP. It is characterized by a single depot and a set of satellites. A fleet of homogeneous vehicles of known size is available at each echelon. Vehicle capacities are limited. Only one type of product is to be shipped and split deliveries are only allowed at the first level. The objective is to minimize the total routing cost in both levels.

To address this problem, we propose a hybrid heuristic that relies on two components embedded in an iterative framework. The first component aims to generate a set of promising routes using destroy and repair operators combined with an efficient local search procedure. The second component recombines the generated routes by solving a set covering problem to obtain a high quality solution. Computational experiments conducted on the test instances of the literature show the performances of our approach, as it reached high quality solutions in short computing times, and was able to improve the current best known solution for several large instances.

The remainder of this paper is organized as follows. In Section 2, an overview of the related literature is given. The problem is described in Section 3, and the proposed approach is explained in Section 4. Section 5 presents computational results and compares them to the best known solutions of the literature. Finally, Section 6 concludes and discusses possible directions for future research.

2 Related work

The first definition of the Two-Echelon Vehicle Routing Problem (2E-VRP) was introduced by Perboli et al. [11, 12] who proposed a flow-based formulation, and solved the problem using a Branch-&-Cut algorithm (B&C). Since then, several exact methods have been developed to solve the 2E-VRP. Jepsen et al. [9] presented a different model for the 2E-VRP and solved it using a B&C that relies on a MILP relaxation of the problem, a feasibility test, and a specialized branching scheme to branch on infeasible solutions. Santos et al. [15] developed two Branch-&-Price (B&P) algorithms to solve the 2E-VRP. The first considers routes that satisfy elementary constraints while the second relaxes such conditions when pricing. Later, Santos et al. [16] implemented a Branch-&-Cut-&-Price algorithm by incorporating valid inequalities into their B&P. Currently, the best exact method for the 2E-VRP was introduced in [1] and solves the problem by decomposing it into a set of Multi-Depot VRP with side constraints. It relies on a new integer linear programming (ILP) formulation, a bounding procedure based on dynamic programming, and a dual ascent method.

Various heuristics were also proposed to solve the 2E-VRP. Crainic et al. [3, 4] addressed the 2E-VRP by separating the first and the second echelon into two sub-problems, and then solving them sequentially. Components of these heuristics were later used in a hybrid GRASP with path re-linking in [5]. Hemmelmayr et al. [8] implemented an Adaptive Large Neighborhood Search (ALNS) that uses various repair and destroy operators specifically designed to solve the 2E-VRP. They also introduced new large instances on which they tested their algorithm. Zeng et al. [20] presented a hybrid two phase heuristic composed of a GRASP and Variable Neighborhood Descent (VND). Breunig et al. [2] developed a Large Neighborhood Search (LNS) for the 2E-VRP that was able to improve the best known solutions for several instances of the literature. More recently, Wang et al. [19] implemented a hybrid algorithm for the *2E-VRP with Environmental Considerations (2E-CVRP-E)*. Their algorithm comprises of a Variable Neighborhood Search (VNS) followed by a post optimization step based on the resolution of a linear program. Their algorithm further improves some best known solutions for 2E-VRP instances.

Related work may include other variants of the 2E-VRP (see [7, 17, 21]), and similar problems like the Two-Echelon Location Routing Problem (2E-LRP) and the Truck and Trailer Routing Problem (TTRP). For a more detailed survey on the subject, we invite the reader to refer to Cuda et al. [6].

3 Problem definition

The 2E-VRP is defined on a weighted undirected graph $G = (V, A)$, where V is the set of nodes and A the set of arcs. Set V is partitioned as $V = \{v_0\} \cup V_{sat} \cup V_{cust}$. Node v_0 represents the depot, subset V_{sat} contains n_{sat} satellites and subset V_{cust} contains n_{cust} customers. Set $A = A_1 \cup A_2$ is divided into two subsets. $A_1 = \{(i, j) : i, j \in \{v_0\} \cup V_{sat}, i \neq j\}$ contains the arcs that can be taken by first level vehicles: trips between the depot and the satellites and trips between pairs of satellites. $A_2 = \{(i, j) : i, j \in V_{sat} \cup V_{cust}, (i, j) \notin V_{sat} \times V_{sat}, i \neq j\}$ contains the arcs that can be taken by second level vehicles: trips between customers and satellites and trips between pairs of customers. A travel cost c_{ij}, $(i, j) \in A$, is associated with each arc. We assume that the matrix (c_{ij}) satisfies the triangle inequality.

Each customer $i \in V_{cust}$ demands d_i units of freight to be delivered. The demand of a customer cannot be split among several vehicles, that is, a customer must be served exactly once. Moreover, customer demands cannot be delivered by direct shipping from the depot and must be consolidated at a satellite. Satellite demands are not explicitly given but considered to be the sum of all the customer demands that are served trough the satellite. We assume that it can exceed vehicle capacity and thus, we allow for it to be split among different vehicles e.i. a satellite can be served by more than one vehicle. A satellite may also have a demand equal to zero and, in this case, not be visited by any vehicle. Consolidating shipments at satellite $s \in V_{sat}$ incurs handling costs equal to h_s times the quantity of handled goods.

A fleet f_1 of m_1 identical vehicles of capacity Q_1 is located at the depot v_0 and is used to deliver goods to the satellites. Additionally, a fleet f_2 of m_2 identical vehicles of capacity Q_2 is available for serving the customers. Each of the m_2 vehicles can be located at any satellite $s \in V_{sat}$ as long as the number of vehicles at one satellite does not exceed a limit k_s.

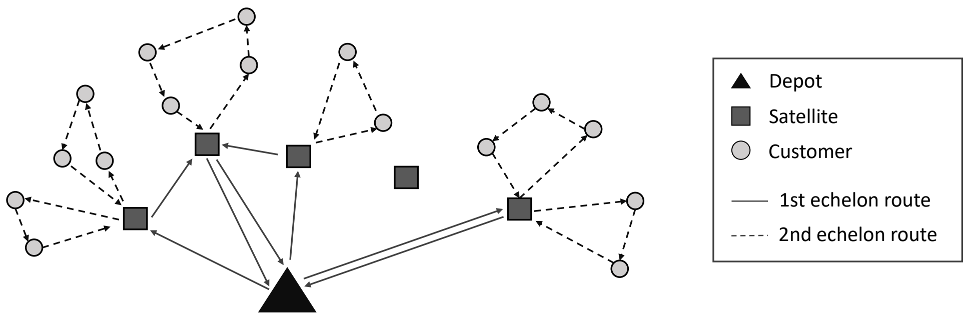


Figure 1 Example of a 2E-VRP solution.

We define a first-level route as a route performed by a first-level vehicle that starts at the depot, visits one or several satellites then returns to the depot. In a same way, we define a second-level route as a route run by a second-level vehicle that starts at satellite $s \in V_{sat}$, visits a subset of customers before returning to s. Routes must respect vehicle capacities, that is, the sum of deliveries made by a first-level route to the satellites it visits must not exceed Q_1 and the total demand of the customers visited by a second-level route must not exceed Q_2. Each vehicle performs only one tour, and each route has a cost equal to the sum of the costs of the arcs used.

The objective of the 2E-VRP is to find a set of routes at both levels such that each costumer is visited exactly once, the capacity constraints are respected, the quantity delivered to costumers from each satellite is equal to the quantity received from the depot, and the total routing and handling costs are minimized. Figure 1 shows a solution example for the 2E-VRP.

4 Solution method

We propose a hybrid heuristic that relies on a neighborhood search to generate good feasible solutions, and a integer programming (IP) method to recombine the routes from those solutions into a better one. Algorithm 1 summarizes the steps of our method.

At each iteration of the algorithm, the route generation heuristic takes an initial solution S and tries to improve it while exploring the solution space and storing new routes in the pool. After that, the recombination component uses the discovered routes to construct a better solution by solving a Set Cover based formulation of the 2E-VRP. If the recombination fails to produce a better solution, the algorithm constructs a different one from scratch and uses it as initial solution for the route generation heuristic during the next iteration. The idea of the approach is to use the integer program as a mean to find better quality solutions missed by the route generation heuristic while guiding the search process towards different regions of the solution space.

Exact models are usually used as post-optimization techniques after the heuristic resolution process as was done in [14, 19]. This is due to the exponential worst case performance of the model. What is new in our proposal is that we iteratively apply a Set Cover (SC) based formulation of the problem as a refinement technique rather than focusing on the local search results. We show that is possible to combine efficiently heuristic and exact algorithms to explore the search space within short runtimes.

Algorithm 1 Neighborhood Search and Set Cover hybrid heuristic for the 2E-VRP.

```
S_best :=  BestInsertionHeuristic();
S := S_best;
pool := {};
While (!Stopping criteria) Do
Begin
    S := RouteGenerationHeuristic(S,pool);
    If (cost(S) < cost(S_best)) Then
    S_best := S;

    S := RouteRecombination(pool);
    If (cost(S) < cost(S_best)) Then
        S_best := S;
    Else
        S := GreedyInsertionHeuristic();    /* Restart */
    End;
Return S_best;
```

4.1 Route generation heuristic

The neighborhood search we use to explore the solution space is based on the destroy-and-repair principle. At each iteration, a part of the solution is destroyed by removing a limited number of customers using a *destroy operator*. The removed customers are then re-inserted into the solution with a *repair operator*. The structure of this heuristic is described in Algorithm 2.

Starting from an initial solution, a random number $\eta \in [1, \tau]$ of customers is removed from the second echelon. The maximum number of customers to be removed τ is first initialized to τ_{min} and then increased after each non-improving iteration until it reaches τ_{max}. As soon as an improvement is found, τ is reset to τ_{min}. Slowly varying the value of τ during the execution allows to intensify the search around promising solutions and then to slowly increase diversification as the search converges toward a local optima. Once the solution is destroyed, the removed customers are reinserted using a *repair operator* and the obtained solution is passed to a *local search* to improve the second echelon routes. After that, the satellite demands are computed and the first echelon routes are constructed to obtain a complete solution. If the new solution has a better objective value than S, it is accepted as the new incumbent. Moreover, after i_{max} consecutive iterations without improving the incumbent solution, the best-known solution is updated and the configuration of the available satellites is modified using $perturb(S)$ to allow the search procedure to explore a different region of the solution space. The solution obtained after the perturbation becomes the new incumbent. The algorithm ends after $iter_{repeat}$ consecutive iterations have been performed without improving the best-found solution.

4.1.1 Destruction

The destroy procedure only considers the second level routes. At each iteration, it randomly chooses one of the following operators and removes a random number of customers η in $[1, \tau]$.

a. **Random removal operator:** removes η randomly chosen customers from the solution.
b. **Worst removal operator:** removes the customers with the highest increase in solution cost. More precisely, it calculates for each customer k located between i and j a saving

Algorithm 2 Route generation heuristic.

```
S := S_0;  /* Initial solution */
S_best := S;
tau := tau_min;  iter := 0;
Repeat
    i := 1;
    While(i < i_max) Do
    Begin
        S_tmp := destroy(S,tau);
        S_tmp := localSearch(repair(S_tmp));
        S_tmp := firstLevelReconstruction(S_tmp);
        pool := update(pool,S_tmp); /* Add routes to pool */
        If(cost(S_tmp) < cost(S)) Then
        Begin
            S:= S_tmp;
            i := 1;
            tau := tau_min;
        End;
        Else
        Begin
            i := i + 1;
            increment(tau);
        End;
    End;
    If(cost(S) < cost(S_best)) Then
    Begin
        S_best := S;
        iter := 0;
    End;
    Else iter := iter + 1;

    S := perturb(S);

    Until (iter >= iter_repeat);
Return  S_best;
```

value $c_{ik} + c_{kj} - c_{ij}$. Savings are then normalized by the average cost of the incident arcs of the corresponding customer and altered by a random factor between 0.8 and 1.2 as in [8]. Finally, customers are sorted in decreasing order of their normalized savings and the η first customers are removed from the solution. Normalizing the savings serves to avoid repeatedly removing the customers that are isolated from the others.

c. **Sequence removal operator:** removes a sequence of η consecutive customers from a randomly chosen route. If η is larger than the chosen route, the whole route is destroyed and the remaining number of customers is removed from a second route.

4.1.2 Repair and first level reconstruction

Repair is performed by using two heuristics : *Best Insertion Heuristic* (BIH) and *Greedy Insertion Heuristic* (GIH). When repairing an incomplete solution, we first use BIH. This constructive heuristic identifies among all the unrouted customers the one that increases the least the total solution cost and inserts it at its best position. It repeats the process until all customers are routed. If one or more customers remain unrouted because their demands are higher than the largest remaining capacity of any vehicle, the repair process is restarted using GIH. The *Greedy Insertion Heuristic* inserts customers in a random order one after the other at their cheapest possible position in the solution. If the GIH fails, the customers are randomly reordered and the heuristic restarts. We observed that proceeding this way is sufficient to achieve feasible solutions after a small number of tries. These repair heuristics consider feasible insertions in already existing routes. If the maximum number of vehicles is not yet reached, the creation of new empty routes from open satellites is also tested.

The construction of the first-echelon routes is achieved by means of a heuristic similar to GIH. The heuristic starts by creating for each satellite with a demand greater than Q_1 enough back-and-forth trips so that its remaining demand becomes smaller than Q_1. Once it is done, the heuristic proceeds to insert of the remaining demands the same way as GIH.

4.1.3 Local search

The local search procedure consists of the following operators : $2-opt$, $2-opt^*$, $Relocate(\lambda)$, and $Swap(\lambda_1, \lambda_2)$ with $\lambda, \lambda_1, \lambda_2 \in \{1, 2\}$. The $2-opt$ operator [10] removes arcs $(i, i+1)$ and $(j, j+1)$ from the same route and reconnects arcs (i, j) and $(i+1, j+1)$. The $2-opt^*$ operator [13] is performed on each pair of routes u and v originating from the same satellite. It replaces arcs $(i, i+1)$ from u and $(j, j+1)$ from v by arcs $(i, j+1)$ and $(j, i+1)$ or by arcs (i, j) and $(i+1, j+1)$. *Relocate* moves sequences of λ customers to their best positions in the solution. Finally, *Swap* exchanges the positions of two sequences of λ_1 and λ_2 customers from the same route or from two different routes. At each iteration, the local search procedure randomly applies one of the above operators. If the chosen operator does not improve the solution, it is discarded, otherwise the set of operators is reset. The process continues until all operators have been discarded. Moves from each operator are performed in a first-improvement manner until no improving move can be found in the neighborhood.

4.1.4 Perturbation

In order to explore different regions of the search space, we temporarily close satellites and reopen them using the *Close Satellites* and *Open Satellites* operators.

a. **Close Satellites:** randomly chooses one satellite among the open ones having at least one route originating from them and closes it. The routes of the chosen satellite are reassigned to an open satellite that keeps their cost to a minimum. When the number of open satellites becomes less than the minimum required to serve all customers, the operator chooses a random satellite among the closed ones and opens it.
b. **Open Satellites:** chooses a random number of satellites among those that are closed and opens them. In order to allow the number of open satellites to decrease, especially at the beginning when most of them are open, the number of satellites to be opened can be nil.

4.2 Recombination method

The route recombination component uses a pool of routes collected during the search process and recombines them to obtain a high-quality solution by solving a set cover based formulation of the problem. In the following, we introduce the notations used in the IP model.

Let $\mathcal{M}$ be the set of all the possible first level routes, and $\mathcal{M}_s \subseteq \mathcal{M}$ the subset of first-level routes that serve satellite $s \in V_{sat}$. We note g_r the cost of route $r \in \mathcal{M}$. Let $\mathcal{R}$ be the set of all the possible second-level routes, and $\mathcal{R}_s$ the subset or routes passing through $s \in V_{sat}$, thus $\mathcal{R} = \bigcup_{s \in V_{sat}} \mathcal{R}_s$. We associate to each route $r \in \mathcal{R}$ a cost c_r, and a load $w_r = \sum_{c \in r} d_c$ equal to the total demand of customers visited in route r. The binary parameter δ_{ri} is equal to 1 if and only if route $r \in \mathcal{R}$ visits customer $i \in V_{cust}$, and 0 otherwise. The second-level routes having been extracted from valid solutions, they all satisfy the vehicle capacity constraints.

Let $y_r \in \{0,1\}$ be a binary decision variable equal to 1 if and only if first-level route $r \in \mathcal{M}$ is in the solution, $x_r \in \{0,1\}$ a binary decision variable equal to 1 if and only if second-level route $r \in \mathcal{R}$ is in the solution, and q_{sr} a non-negative variable representing the amount of goods delivered by route $r \in \mathcal{M}$ to satellite $s \in V_{sat}$. We assume that $q_{sr} = 0$ if satellite s is not visited in route r. Parameter h_s represents handling costs at satellite $s \in V_{sat}$. The route recombination model can be formulated as follows:

$$\min \ z = \sum_{r \in \mathcal{R}} c_r \cdot x_r + \sum_{r \in \mathcal{M}} g_r \cdot y_r + \sum_{s \in V_{sat}} \sum_{r \in \mathcal{M}_s} h_s \cdot q_{sr} \tag{1}$$

$$\text{s.t.} \ \sum_{r \in \mathcal{R}} \delta_{ri} \cdot x_r \geq 1 \quad , \quad \forall i \in V_{cust} \tag{2}$$

$$\sum_{r \in \mathcal{R}_s} x_r \leq k_s \quad , \quad \forall s \in V_{sat} \tag{3}$$

$$\sum_{r \in \mathcal{R}} x_r \leq m_2 \tag{4}$$

$$\sum_{r \in \mathcal{M}} y_r \leq m_1 \tag{5}$$

$$\sum_{r \in \mathcal{M}_s} q_{sr} = \sum_{r \in \mathcal{R}_s} w_r \cdot x_r \quad , \quad \forall s \in V_{sat} \tag{6}$$

$$\sum_{s \in V_{sat}} q_{sr} \leq Q_1 \cdot y_r \quad , \quad \forall r \in \mathcal{M} \tag{7}$$

$$x_r \in \{0,1\}, \quad r \in \mathcal{R} \tag{8}$$

$$y_r \in \{0,1\}, \quad r \in \mathcal{M} \tag{9}$$

$$q_{sr} \in \mathbb{R}_+, \quad s \in V_{sat}, \ r \in \mathcal{M} \tag{10}$$

The objective function (1) states to minimize routing costs on both levels plus handling costs at each satellite. Constraints (2) ensure that each customer is visited at least once. Constraints (3) limit the number of second-level vehicles per satellite. Constraints (4) and (5) impose upper bounds on the number of vehicles used to implement first and second level routes. Balance between the quantity delivered by first-level routes to a satellite and the customer demands supplied from said satellite is imposed by constraints (6). Constraints (7) ensure that the capacity of first-level vehicles in not exceeded. Because the total amount of goods that need to be supplied to each satellite is not known beforehand, we cannot assume that capacity constraints are respected by first-level routes like we did for second-level routes. We need to explicitly state them in the formulation. Finally, constraints (8), (9), and (10) define the values domain for the decision variables.

Note that the model we use in our recombination component is a relaxation of the 2E-VRP. Constraints (2) require that each customer is visited at least once, instead of exactly once. However, since the distance matrix satisfies the triangle inequality, the two formulations remain equivalent as the resolution process will naturally lean towards solutions with the least possible amount of visits to a same customer. If the pool contains all the possible routes, solving the formulation with the relaxed model will still result in an optimal solution where each customer is visited exactly once. The idea of relaxing the problem stems from the fact that the recombination pool only contains a limited subset of routes, thus the solutions it finds may be few. To increase the number of combinations that can be made, we choose to allow combining routes that share common customers, as it can lead to better objective values. Even though the resulting combination may not be a valid solution to the 2E-VRP, removing the extra visits to each customer makes it feasible while producing new routes and further lowering the objective value.

4.2.1 Pool management and initialization

The performance of the route recombination component strongly depends on the size of the pool of routes. A larger size increases the chances of finding high-quality solutions but also induces higher computation times, whereas a small size reduces computation times but makes finding improved solutions less likely. Thus, pool size must be fixed in order to offer a good trade-off between solution quality and computation efforts. Furthermore, to account for the lesser number of available routes, it is better to keep inside the pool only routes that are more likely to be in high-quality solutions. To this end, we assign each route a priority based on the cost of the solution it was extracted from, thus favoring routes that belong to the best found solutions. When the pool capacity is reached, routes with lower priority are removed and replaced by the new ones. If a route already exists inside the pool, its priority is updated if it is extracted from a better solution.

The pool is initialized with the routes of x different solutions generated by the *Greedy Insertion Heuristic* described in Section 4.1.2 and improved with the local search procedure described in Section 4.1.3. Furthermore, for each satellite s we add m^1 copies of round trip routes to s from the depot to account for the possibility of it being served more than once.

4.2.2 Correcting heuristic

When the route recombination model is solved, some customers might be visited more than once. In this case, we use a correcting heuristic to remove the extra visits and produce a valid solution. The algorithm starts by establishing the set V_{cm} of customers that are visited more than once. It then computes for each visit v of each customer $i \in V_{cm}$ its removal gain δ_{iv}, removes the visit with the highest gain and updates the gains for the remaining ones. When the number of visits to a customer drops to one, it is removed from V_{cm}. The procedure is repeated until V_{cm} becomes empty. During our tests, we observed that only a few customers tend to be visited multiple times. Thus, this simple heuristic proves to be enough to provide good results with limited computational effort.

5 Computational results

Our algorithm was coded in C++ using the Standard Template Library (STL) for data structures, and IBM ILOG CPLEX 12.6.3 to solve the IP. The algorithm is compiled with the GNU GCC compiler in a Linux environment and tested on an Intel Xeon E5-2670v2 CPU at 2.50GHz with similar performance to the ones used in the literature.

We conducted extensive computational experiments on the benchmark instances for the 2E-VRP. There are currently six instance sets available. The size of the instances ranges from 12 customers and 2 satellites, to 200 customers and 10 satellites. The main characteristics of the benchmark instances are listed in Table 5 of Appendix A. Note that the small instances of *Set 1* are no longer used for testing, thus they are not included. For our tests we used the files provided by Breunig et al. [2].

Table 1 Parameter settings.

Parameter	Description	Value
i_{max}	max. nb. of non-improving iterations before perturbing the solution	$0.2n$
$iter_{repeat}$	max. nb. of non-improving iterations for the route generation	10
τ_{min}, τ_{max}	max. nb. of customers to be removed	$0.15n, 0.45n$
S_{pool}	size of the pool	$\frac{\sum d_c}{m^2} * 15$
N_{algo}	max. nb. of non-improving iterations in the global algorithm	n

5.1 Parameter tuning

The proposed approach has six parameters: (1) i_{max}, $iter_{repeat}$, τ_{min} and τ_{max} in the route generation heuristic; (2) the size of the pool (S_{pool}) in the route recombination component; and (3) the stopping criterion of the iterative framework. We carried out a series of preliminary experiments to set the parameter values: we tested our algorithm on a subset of instances while varying parameter values, and kept those that offered the best trade-off between solution quality and runtime. The stopping criteria is set according to previous literature. Breunig et al. [2] set the maximum runtime of their algorithm to 60s for small instances and 900s for larger ones. Wang et al. [19] use the maximum runtime and the maximum number of iterations N_{algo} without improving the best found solution as stopping rules. They set them so that the maximum runtime of their algorithm does not exceed 1500s. To show the performance of our method we restrict our runtime to 60s and 900s as do Breunig et al. [2]. The remaining parameter settings are given in Table 1.

5.2 Comparison with the literature

In order to investigate the effectiveness of the proposed algorithm, we compare its performance, when applicable, with that of the ALNS by Hemmelmayr et al. [8], the LNS by Breunig et al. [2] and the VNS by Wang et al. [19] as well as the current best-known solution for each instance from the literature. All the results were obtained through five independent runs of the algorithm and are summarized in Table 2. The results of our Neighborhood Search and Set Cover Hybrid Heuristic are listed in column “NS-SC”. The columns “ALNS”, “LNS”, and “VNS” show the results of the methods proposed by Hemmelmayr et al. [8], Breunig et al. [2], and Wang et al. [19], respectively. The average and the best objective value of the five runs are given in columns “Avg. 5” and “Best 5”, respectively. Column “CPU” shows the average runtime of the algorithm in seconds. The column “BKS” refers to the best-known solution of that set of instances. As was observed by Breunig et al. [2], there exist some small differences in objective values that can be explained by a different rounding convention or the small optimality gap of CPLEX. Table 3 summarizes the gaps obtained by each algorithm on each benchmark. Columns “Avg. %” and “Best %” show the average and best gap, respectively, expressed as a percentage. The overall gap is calculated by considering the number of instances in each benchmark.

Instances in Sets 2 and 3, are relatively easy to solve and all algorithms are able to find the best known solutions at least one time out of five. Instances in Set 4, while not bigger than some instances of Sets 2 and 3, are more difficult to solve due to customer distribution [2]. Our “NS-SC” only misses four of the current best known solutions, and still achieves high quality solutions with gaps less than 0.04%. Instances of Set 5 are the largest of the literature and those where the gaps and the runtimes are more important. On these instances, all the algorithms fail to achieve the best known solutions for several instances,

Table 2 Computational results for 2E-VRP instances.

Instances		$\|V_{cust}\|$	ALNS			LNS			VNS			NS-SC			BKS
			Avg 5	Best 5	CPU	Avg 5	Best 5	CPU	Avg 5	Best 5	CPU	Avg	Best 5	CPU	
Set 2	a	21	410.69	410.69	35	410.69	410.69	60	410.69	410.69	1	410.69	410.69	1	410.69
		32	744.49	744.49	69	744.49	744.49	60	744.49	744.49	4	744.49	744.49	6	744.49
	b	50	559.20	559.20	147	559.20	559.20	60	559.20	559.20	18	559.20	559.20	22	559.20
		50	530.10	530.10	150	530.10	530.10	60	530.10	530.10	13	530.10	530.10	19	530.10
	c	50				628.65	628.65	60	628.65	628.65	42	628.65	628.65	24	628.65
		50				566.52	566.52	60	566.52	566.52	25	566.52	566.52	22	566.52
Set 3	a	21	512.61	512.61	40	512.61	512.61	60	512.61	512.61	1	512.61	512.61	1	512.61
		32	669.99	669.99	78	669.99	669.99	60	669.99	669.99	6	669.99	669.99	5	669.99
	b	50	714.75	714.75	161	714.75	714.75	60	714.75	714.75	37	714.75	714.75	28	714.75
	c	50				668.40	668.40	60	668.93	668.40	70	668.40	668.40	27	668.40
Set 4	a	50				1527.13	1526.86	60	1527.80	1527.55	38	1527.23	1526.87	25	1526.86
		50				1415.66	1415.66	60	1416.69	1415.85	67	1416.12	1415.70	23	1415.65
		50				1317.36	1317.34	60	1318.34	1317.48	85	1318.16	1317.34	31	1317.33
	b	50	1524.85	1524.70	248	1516.01	1515.61	60	1516.70	1516.27	36	1516.18	1515.61	33	1515.60
		50	1378.16	1377.85	139	1375.68	1375.68	60	1376.37	1375.79	36	1375.82	1375.68	29	1375.67
		50	1301.17	1300.33	121	1299.93	1299.88	60	1300.23	1299.96	63	1300.28	1299.91	37	1299.87
Set 5	5_1	100	1067.11	1058.27	373	1058.94	1057.58	900	1059.37	1058.26	953	1059.32	1056.75	582	1056.74
	5_2	100	960.87	950.01	421	961.41	956.64	900	951.70	949.47	1452	955.21	951.45	557	946.82
	5_3	200	1386.45	1357.14	974	1375.70	1357.79	900	1343.60	1340.87	1500	1355.42	1348.62	900	1338.35
Set 6	A_50	50				639.64	639.64	60	640.29	640.06	155	639.89	639.64	35	639.64
	A_75	75				947.85	946.35	900	946.54	946.32	730	946.86	946.86	235	946.32
	A_100	100				1150.05	1147.34	900	1147.05	1146.18	1216	1147.31	1146.32	560	1146.18
	B_50	50				826.48	826.48	60	826.59	826.48	141	826.59	826.59	33	826.48
	B_75	75				1461.94	1461.54	900	1461.33	1461.29	578	1461.53	1461.29	363	1461.29
	B_100	100				1741.67	1738.57	900	1736.80	1734.02	1322	1736.53	1733.92	712	1733.36
Avg.			904.65	900.78	227	984.83	983.53	295	983.01	982.45	344	983.51	982.72	172	982.06

Table 3 Summary of average and best gaps on 2E-VRP benchmarks.

	ALNS		LNS		VNS		NS-SC	
	Avg. %	Best %	Avg. %	Best %	Avg. %	Best %	Avg. %	Best %
Set 2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Set 3	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00
Set 4a			0.01	0.00	0.07	0.02	0.04	0.00
Set 4b	0.30	0.26	0.01	0.00	0.05	0.02	0.03	0.00
Set 5	2.00	0.63	1.51	0.86	0.39	0.20	0.80	0.42
Set 6 A			0.16	0.04	0.06	0.02	0.07	0.02
Set 6 B			0.17	0.11	0.07	0.01	0.11	0.03
Overall	1.27	1.20	0.18	0.09	0.08	0.03	0.10	0.04

Table 4 New best known solution values found by NS-SC.

	Instance	$\lvert V_{cust}\rvert$	$\lvert V_{sat}\rvert$	m_1	m_2	Former best known	New best known
Set 5	100-5-1b	100	5	5	15	1108.62	1103.55
	100-10-1b	100	10	5	18	916.25	911.8
	100-10-3b	100	10	5	17	850.92	849.73
	200-10-1	200	10	5	62	1539.29	1538.35
	200-10-1b	200	10	5	30	1186.78	1175.81
	200-10-3	200	10	5	63	1780.67	1779.68
	200-10-3b	200	10	5	30	1197.9	1196.93

mainly due to the bigger numbers of customers and satellites that constitute the instances. The ALNS and the LNS can achieve an average relative gap of 2.00% and 1.51%, respectively. The VNS achieves an average relative gap of 0.39%, but is slower than the other algorithms. Our algorithm, on the other hand, offers good compromise between solutions quality and runtime, as it achieves an average relative gap of 0.80% while being significantly faster than both the LNS and the VNS. It was also able to improve the current best known solutions for a total of seven instances from Set 5 during our experiments. Only Breunig et al. [2] and Wang et al. [19] report results on the instances of Set 6. The LNS, the VNS, and our NS-SC are all able to obtain very low average relative gaps on both Set 6a and Set 6b, but once again our algorithm has a smaller runtime.

Overall, our algorithm is able to achieve the current best known solutions for 216 out of 234 instances with an overall average relative gap of 0.10% and running times smaller than those of the literature. During our experiments, NS-SC found seven new best known solution values for the instances of Hemmelmayr et al. [8]. The values of these newly found solutions are reported in Table 4. The Set designation and the names of the instances are displayed in the first two columns. Column $|V_{cust}|$ represents the number of customers in the instance, $|V_{sat}|$ is the number of satellites, and m_1 and m_2 indicate the number of available first-level and second-level vehicles, respectively. Based on the above results, our approach is very effective in solving the 2E-VRP.

6 Conclusions

In this paper, we presented a hybrid heuristic for the 2E-VRP. The algorithm uses an effective neighborhood search to explore the solution space and discover high quality solutions. By keeping trace of the exploration steps, the heuristic generates a set of routes which are then recombined using an integer programming model. Solving this model serves as way to find better solutions that were missed by the neighborhood search procedure and to faster lead the algorithm towards promising regions of the solution space. Computational experiments on the standard benchmark instances demonstrate the competitiveness of our approach. Our algorithm consistently achieves high quality solutions with an overall average relative gap of 0.10%, while requiring less running time than other algorithms, and improves the current best known solutions for seven instances for which no optimal solution is known.

In summary, the results presented in this paper are encouraging for the application of our approach to optimize other two-echelon routing problems. Its components can be adapted and additional ones can be integrated to account for different constraints. Future work will primarily focus on the extension of the algorithm to variants of the 2E-VRP and similar routing problems, mainly to accommodate more practical constraints and more realistic cost structures.

References

1. Roberto Baldacci, Aristide Mingozzi, Roberto Roberti, and Roberto Wolfler Calvo. An exact algorithm for the two-echelon capacitated vehicle routing problem. *Operations research*, 61(2):298–314, 2013.
2. U. Breunig, V. Schmid, R. F. Hartl, and T. Vidal. A large neighbourhood based heuristic for two-echelon routing problems. *Computers and Operations Research*, 76:208–225, 2016.
3. Teodor Gabriel Crainic, Simona Mancini, Guido Perboli, and Roberto Tadei. Clustering-based heuristics for the two-echelon vehicle routing problem. *CIRRELT-2008-46*, 2008.
4. Teodor Gabriel Crainic, Simona Mancini, Guido Perboli, and Roberto Tadei. Multi-start heuristics for the two-echelon vehicle routing problem. In Peter Merz and Jin-Kao Hao, editors, *Evolutionary Computation in Combinatorial Optimization*, pages 179–190, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
5. Teodor Gabriel Crainic, Simona Mancini, Guido Perboli, and Roberto Tadei. GRASP with Path Relinking for the Two-Echelon Vehicle Routing Problem. In *Advances in Metaheuristics*, pages 113–125. Springer New York, New York, NY, 2013.
6. R Cuda, G Guastaroba, and M G Speranza. A survey on two-echelon routing problems. *Computers & Operations Research*, 55:185–199, 2015.
7. Philippe Grangier, Michel Gendreau, Fabien Lehuédé, and Louis-Martin Rousseau. An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. *European Journal of Operational Research*, 254(1):80–91, 2016.
8. Vera C Hemmelmayr, Jean-François Cordeau, and Teodor Gabriel Crainic. An adaptive large neighborhood search heuristic for Two-Echelon Vehicle Routing Problems arising in city logistics. *Computers & Operations Research*, 39(12):3215–3228, 2012.
9. Mads Jepsen, Simon Spoorendonk, and Stefan Ropke. A branch-and-cut algorithm for the symmetric two-echelon capacitated vehicle routing problem. *Transportation Science*, 47(1):23–37, 2013.
10. S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.
11. Guido Perboli and Roberto Tadei. New families of valid inequalities for the two-echelon vehicle routing problem. *Electronic notes in discrete mathematics*, 36:639–646, 2010.
12. Guido Perboli, Roberto Tadei, and Daniele Vigo. The two-echelon capacitated vehicle routing problem: Models and math-based heuristics. *Transportation Science*, 45(3):364–380, 2011.

13 Jean-Yves Potvin and Jean-Marc Rousseau. An exchange heuristic for routing problems with time windows. *Journal of the Operational Research Society*, 46(12):1433–1446, 1995.

14 Yves Rochat and Éric D Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167, 1995.

15 Fernando Afonso Santos, Alexandre Salles da Cunha, and Geraldo Robson Mateus. Branch-and-price algorithms for the two-echelon capacitated vehicle routing problem. *Optimization Letters*, 7(7):1537–1547, 2013.

16 Fernando Afonso Santos, Geraldo Robson Mateus, and Alexandre Salles da Cunha. A branch-and-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem. *Transportation Science*, 49(2):355–368, 2014.

17 Mehmet Soysal, Jacqueline M. Bloemhof-Ruwaard, and Tolga Bektaş. The time-dependent two-echelon capacitated vehicle routing problem with environmental considerations. *International Journal of Production Economics*, 164:366–378, jun 2015.

18 Eiichi Taniguchi, Russell G. Thompson, Tadashi Yamada, and J.H.R. van Duin. *City logistics –Network modelling and intelligent transport systems.* Pergamon, Oxford, elsevier edition, 2001.

19 Kangzhou Wang, Yeming Shao, and Weihua Zhou. Matheuristic for a two-echelon capacitated vehicle routing problem with environmental considerations in city logistics service. *Transportation Research Part D*, 57(October):262–276, 2017.

20 Zheng Yang Zeng, Wei Sheng Xu, Zhi Yu Xu, and Wei Hui Shao. A Hybrid GRASP+VND heuristic for the two-echelon vehicle routing problem arising in city logistics. *Mathematical Problems in Engineering*, 2014, 2014.

21 Lin Zhou, Roberto Baldacci, Daniele Vigo, and Xu Wang. A Multi-Depot Two-Echelon Vehicle Routing Problem with Delivery Options Arising in the Last Mile Distribution. *European Journal of Operational Research*, 265(2):765–778, 2018.

A Benchmark instances for the 2E-VRP

There are six sets of benchmark instances for the Two-Echelon Vehicle Routing Problem (2E-VRP). The size of the instances ranges from 12 customers and 2 satellites, to 200 customers and 10 satellites. For our tests, we used the instances provided by [2]. Table 5 displays the main characteristics of the different sets. Column *Nb.* represents the number of instances of the set, $|V_{cust}|$ is the number of customers, $|V_{sat}|$ is the number of satellites, m_1 and m_2 the number of available first-level and second-level vehicles, respectively, and k_s represents the maximum number of second-level routes per satellite. Note that the small instances of *set 1* are no longer used for testing, thus they are not included in the table.

Table 5 Characteristics of the benchmark instances for the 2E-VRP.

Set	Subset	Nb.	$\|V_{cust}\|$	$\|V_{sat}\|$	m_1	m_2	k_s
Set 2	a	6	21	2	3	4	-
		6	32	2	3	4	-
	b	6	50	2	3	5	-
		3	50	4	4	5	-
	c	6	50	2	3	5	-
		3	50	4	4	5	-
Set 3	a	6	21	2	3	4	-
		6	32	2	3	4	-
	b	6	50	2	3	5	-
	c	6	50	2	3	5	-
Set 4	a	18	50	2	3	6	4
		18	50	3	3	6	3
		18	50	5	3	6	2
	b	18	50	2	3	6	-
		18	50	3	3	6	-
		18	50	5	3	6	-
Set 5		6	100	5	5	$[15, 32]$	-
	-	6	100	10	5	$[17, 35]$	-
		6	200	10	5	$[30, 63]$	-
Set 6	a	9	50	$[4, 6]$	2	50	-
		9	75	$[4, 6]$	3	75	-
		9	100	$[4, 6]$	4	100	-
	b	9	50	$[4, 6]$	2	50	-
		9	75	$[4, 6]$	3	75	-
		9	100	$[4, 6]$	4	100	-

Multi-Source Multi-Sink Nash Flows over Time[*]

Leon Sering
Institute of Mathematics, Technische Universität Berlin
Straße des 17. Juni 136, 10623 Berlin, Germany
sering@math.tu-berlin.de

Martin Skutella
Institute of Mathematics, Technische Universität Berlin
Straße des 17. Juni 136, 10623 Berlin, Germany
martin.skutella@tu-berlin.de

Abstract

Nash flows over time describe the behavior of selfish users eager to reach their destination as early as possible while traveling along the arcs of a network with capacities and transit times. Throughout the past decade, they have been thoroughly studied in single-source single-sink networks for the deterministic queuing model, which is of particular relevance and frequently used in the context of traffic and transport networks. In this setting there exist Nash flows over time that can be described by a sequence of static flows featuring special properties, so-called 'thin flows with resetting'. This insight can also be used algorithmically to compute Nash flows over time. We present an extension of these results to networks with multiple sources and sinks which are much more relevant in practical applications. In particular, we come up with a subtle generalization of thin flows with resetting, which yields a compact description as well as an algorithmic approach for computing multi-terminal Nash flows over time.

2012 ACM Subject Classification Mathematics of computing → Network flows, Theory of computation → Network games

Keywords and phrases Network congestion, Nash equilibrium, dynamic routing game, deterministic queuing model

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.12

Related Version A full version of this paper is available at `https://arxiv.org/abs/1807.01098`.

Acknowledgements The authors are much indebted to Roberto Cominetti and José Correa for interesting discussions and for sharing their insights and thoughts on the topic of this paper.

1 Introduction

With the emergence of novel navigation and vehicle technologies (including, e.g., self-driving/smart vehicles) along with the availability of massive amounts of data in todays and future traffic and transportation networks, increasing attention is given to the mathematical modeling and algorithmic solution of the interplay of individual agents in such networks. We study the behavior of selfish users who wish to travel through a traffic or transportation network. While there is already a vast amount of literature and results on steady states of such systems (see, e.g., Roughgarden [13] and the references therein), much less is known about the often more realistic but also much more complex situation of such systems evolving and changing over time.

[*] This research was carried out in the framework of Matheon supported by Einstein Foundation Berlin.

Flows over time. Flows over time provide an excellent mathematical model for agents (flow particles) traveling through a network over time, with capacities and transit times (delays) on the arcs. Flows over time have been introduced in a seminal paper by Ford and Fulkerson [4] and can also be found in their classic textbook [5]. For a given single-source single-sink network with capacities and transit times on the arcs and a given time horizon, they show how to efficiently construct a maximum flow over time, that is, a way of sending as much flow as possible from the source to the sink within the given time horizon. The underlying algorithm is based on a static min-cost flow computation in the given network where arc transit times are interpreted as costs. A decomposition of the static flow into flows along source-sink-paths then provides an optimal strategy for sending flow over time from the source to the sink by using each path as long as possible. Surprisingly, and in contrast to the situation known for classic (i.e., static) network flows, the problem of balancing given supplies and demands in a network with several sources and/or sinks by sending flow within a given time horizon turns out to be considerably more difficult and complicated. Following the work of Ford and Fulkerson, it took almost four decades before Hoppe and Tardos [8] came up with an efficient algorithm for solving this transshipment over time problem; see also Hoppe's PhD thesis [7]. Their algorithm, however, while being theoretically efficient, relies on parametric submodular function minimization, leading to unpleasant and usually unrealistic running times for networks of practical sizes. Only recently, Schlöter and Skutella [14] presented a slight improvement of this result. Another somewhat surprising evidence for the increased difficulty of flow over time problems compared to static flow problems is the fact that the computation of (fractional) multicommodity flows over time constitutes an NP-hard problem [6]. We refer to [15] for a recent survey on and thorough introduction to flows over time.

Nash equilibria for the deterministic queuing model. The flow over time problems discussed in the previous paragraph are all based on the assumption that flow particles are controlled by a central authority who decides the route choices and schedules of the particles. In most realistic traffic situations, however, the lack of coordination among flow particles necessitates an additional game theoretic perspective. We assume that each flow particle is an individual agent that seeks to arrive at a destination in the least possible time. Such models have mostly been studied in the transportation literature; see, e.g., the book by Ran and Boyce [12] for an overview.

In this paper we study Nash equilibria for flows over time in the deterministic queuing model that is also at the core of many large-scale agent-based traffic simulations such as, e.g., MATSim; see [9]. Here the actual transit time of a flow particle along an arc is the sum of the arc's free-flow transit time plus the waiting time spent in a queue that builds up whenever more flow tries to use an arc than the arc's capacity can handle. In particular, the first-in-first-out (FIFO) principle holds. We refer to Section 3 for a detailed definition.

For a single-source single-sink network, Koch and Skutella [10] characterize Nash flows over time featuring a special and very useful structure: Their derivatives are piece-wise constant, therefore constituting a sequence of particular static source-sink flows, so-called *thin flows with resetting.* Exploiting this key concept of thin flows with resetting, Cominetti, Correa, and Larré [2] provide a constructive proof for the existence and uniqueness of equilibria in this setting, using a fixed-point formulation. Furthermore, for the more general case of multiple origin-destination pairs, they provide a non-constructive existence proof. For the single-source single-sink setting, Cominetti, Correa, and Olver [3] show that, for networks with sufficient capacity, a dynamic equilibrium reaches a steady state in finite time.

Our contribution. Our structural and algorithmic understanding of Nash flows over time is limited to the very restrictive special case of single-source single-sink networks. Moreover, in contrast to the classical case of static flows, single-commodity flows over time in multi-source multi-sink networks with given supplies and demands cannot easily be reduced by introducing a super-source and a super-sink; see, e.g., the work of Hoppe and Tardos [8] discussed above. Nevertheless, we show that such a reduction is possible, albeit non-trivial, when considering a particularly meaningful model of Nash flows over time in such networks. This leads to an interesting generalization of the structural and algorithmic results known for the single-source single-sink case; see [10, 2, 3]. In particular, we present an appropriate generalization of 'thin flows with resetting' and prove that a Nash flow over time can be described and algorithmically obtained via a sequence of these static flows. As another interesting aspect of this work, we show how to get rid of the identification of flow particles with the time they enter the network which has been used in previous work on the single-source single-sink case. In our more general model, all flow is waiting in front of the sources of the network right from the beginning, a subtle point that turns out to be crucial for being able to handle multiple source nodes.

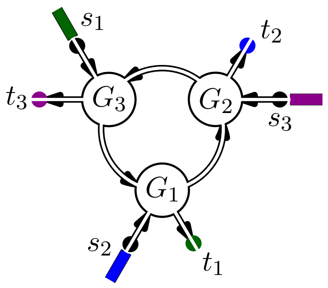
Outline. In Section 2 we informally describe several different settings for dynamic routing games with multiple sources and sinks and identify a suitable model for our purposes. Section 3 introduces the necessary concepts and notations for describing Nash flows over time. Then, Section 4 explains how to deal with multiple source nodes. Finally, in Section 5 multiple sinks are considered as well.

2 Settings for routing games with multiple sources and sinks

There are several different settings for dynamic routing games when considering multiple sources and multiple sinks. We discuss the most meaningful interpretations in the following.

Nash flows over time are mainly motivated by dynamic traffic assignments which naturally lead to the consideration of multiple commodities with independent origin-destination-pairs (s_i, t_i) and inflow rates $r_i \geq 0$, for $i = 1, \dots, n$. At each origin s_i, a flow enters the network with rate r_i and every infinitesimal small particle of this flow has the goal to reach destination t_i as early as possible while considering all other particles from the past and the future. For every commodity, there are time dependent in- and outflow rates for every arc that must satisfy flow conservation at every node. A dynamic equilibrium then consists of a flow over time with n commodities, where each particle chooses a combination of fastest routes from s_i to t_i as strategy. Note that queues build up on arcs whenever the inflow rate exceeds the arc's capacity. This causes a delay of all subsequent particles, therefore influencing the traversing time of all routes using this arc. Cominetti et al. [2] prove that these dynamic equilibria exist by using variational inequalities for the path-based formulation. Unfortunately, the known techniques for single commodity flows are not sufficient for analyzing or algorithmically constructing such dynamic multi-commodity Nash flows over time. The fact that each commodity has different earliest arrival times at the nodes is the main difficulty as this causes cyclic interdependencies between the commodities. Each particle entering the network has to take into account not only all flow that previously entered the network, but also flow entering the network subsequently; an illustrative example is given in the left part of Figure 1.

When we relax the pairing of origins and destinations, however, the route choice of each particle only depends on flow that previously entered the network. We stick to individual inflow rates for the sources, but instead of matching the sources to destinations, we consider



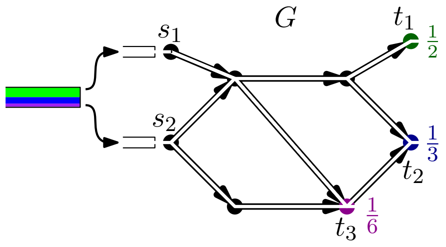


Figure 1 *Left:* Illustration of cyclic interdependencies of commodities with different origin-destination-pairs. The waiting times for flow from s_1 to t_1 within subnetwork G_1 depend on flow starting later from s_2 to t_2. The cyclic symmetry implies that a particle has to take into account not only previous but also future flow from all sources. *Right:* An example of the setting considered in this article. Each flow particle may choose whether to enter the network at s_1 or s_2, but the flow is partitioned according to the demands at the sinks. Here one half of the flow has t_1 as its destination, one third wants to reach t_2, and the rest of one sixth aims at t_3.

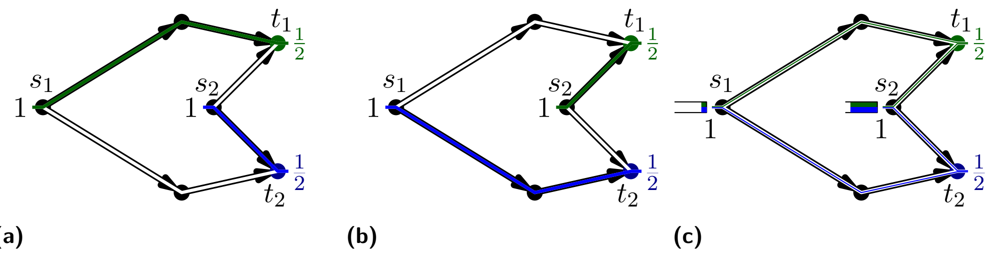


Figure 2 Assume that, at each point in time, the total inflow is equally divided according to the demands. Then, in this symmetric instance, a possible dynamic equilibrium sends all flow from s_1 to t_1 and from s_2 to t_2 (a). Alternatively, the destinations might be swapped (b). Equilibrium (a), however, heavily benefits sink t_2 by serving it earlier as the path from s_1 to t_1 is longer than the path from s_2 to t_2. Symmetrically, equilibrium (b) benefits t_1. This is the reason for considering queues in front of the sources such that a Nash flow over time is forced to behave symmetrically in this instance, that is, sinks with the same demand are treated equally in every equilibrium; see (c).

m sinks $t_1, \dots, t_m$ with demands $d_1, \dots, d_m \geq 0$, such that $d_1 + \dots + d_m = 1$. The value d_j denotes the share of the total flow entering the network that has t_j as destination. In terms of traffic networks this means that each road user has a predetermined destination, but may choose between multiple origins to enter the network; see right side of Figure 1. In order to obtain well defined Nash flows over time with unique arrival times we exclude situations as described in Figure 2 by considering queues in front of the sources. In other words, there is essentially one flow $\mathbb{R}_{\geq 0}$ consisting of a continuum of infinitesimally small particles $\phi \in \mathbb{R}_{\geq 0}$, where each splittable particle chooses, in the order given by $<$, a convex combination of fastest routes from the sources to the sinks as strategy. The sum of the coefficients of all paths to sink t_j has to be equal to demand d_j. Each particle is then split according to these coefficients and each part is sent along its route. How these choices of strategies can be constructed, and what structure these Nash flows over time have, is discussed in this paper.

In the case of one source and multiple sinks with given demands, the two settings presented above are equivalent: given a multi-origin-destination instance with one source but n commodities, we can construct an equivalent multi-source multi-sink instance by setting the inflow at the source to $r := r_1 + \dots + r_n$ and the demand at sink t_j to $d_j := r_j/r$. It is easy to see that these settings are also equivalent in the case of multiple sources and one sink.

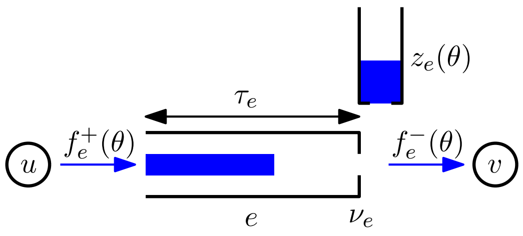


Figure 3 A snapshot of arc $e = uv$ at time θ, with transit time τ_e, capacity ν_e, inflowrate $f_e^+(\theta)$, outflowrate $f_e^-(\theta)$, and queue $z_e(\theta)$.

3 Flow dynamics

In this section we present all necessary definitions of a *fluid queuing network*. The model is a modified version used by Koch and Skutella [10] and Cominetti et al. [1, 2, 3] that matches the multi-source multi-sink setting.

Throughout this paper we consider a directed graph $G = (V, E)$ with transit times $\tau_e \geq 0$ and capacities $\nu_e > 0$ on every arc $e \in E$, a set of $n \geq 1$ sources $S^+ = \{ s_1, \dots, s_n \} \subseteq V$ with inflow rates $r_1, \dots, r_n > 0$, and a set of $m \geq 1$ sinks $S^- = \{ t_1, \dots, t_m \} \subseteq V$. The corresponding demands will be introduced in Section 5. We assume that every node is reachable by a source and can itself reach at least one sink. Furthermore, we assume that the sum of transit times along every directed cycle is positive.

Flows over time. A flow over time is specified by locally integrable and bounded functions $f_e^+ : [0, \infty) \to [0, \infty)$ for every arc e. These *inflow functions* describe the rate of flow entering the arcs for every point in time $\theta \in [0, \infty)$. We set $f_e^+(\theta) := 0$ for $\theta < 0$.

For every arc e there is a bottleneck given by its capacity ν_e at the head of the arc.[1] When flow enters e it immediately starts to traverse this arc, which takes τ_e time. If the rate of flow trying to leave e exceeds the capacity ν_e, the flow builds up a queue in front of the bottleneck which is described by a function $z_e \colon [0, \infty) \to [0, \infty)$. Note that the queue does not have any physical dimension in the network, and is therefore called *point queue*. Whenever there is a positive queue the outflow rate operates at capacity rate ν_e. This leads to the following evolution of the queue starting with $z(0) = 0$,

$$z_e'(\theta) := \begin{cases} f_e^+(\theta - \tau_e) - \nu_e & \text{if } z_e(\theta) > 0 \\ \max \{ f_e^+(\theta - \tau_e) - \nu_e, 0 \} & \text{if } z_e(\theta) = 0. \end{cases} \tag{1}$$

This determines a unique queue function z_e [2], which is characterized later on. The outflow rate function $f_e^- \colon [0, \infty) \to [0, \infty)$ is defined by

$$f_e^-(\theta) := \begin{cases} \nu_e & \text{if } z_e(\theta) > 0, \\ \min \{ f_e^+(\theta - \tau_e), \nu_e \} & \text{if } z_e(\theta) = 0. \end{cases} \tag{2}$$

A *flow over time* is given by a family of inflow functions $(f_e^+)_{e \in E}$ that *conserve flow* at every $v \in V \backslash S^-$, which means that the following equation holds for almost all $\theta \in [0, \infty)$:

$$\sum_{e \in \delta^+(v)} f_e^+(\theta) - \sum_{e \in \delta^-(v)} f_e^-(\theta) = \begin{cases} 0 & \text{if } v \in V \setminus S^+, \\ r_i & \text{if } v = s_i \in S^+. \end{cases} \tag{3}$$

[1] The dynamics are exactly the same if the bottleneck is located at the tail of the arc or anywhere between tail and head.

This ensures that the network does not leak at intermediate vertices and that the amount of flow entering through source s_i matches the inflow rate r_i.

The *cumulative in- and outflow* of an arc e is the total amount of flow that has entered or left e up to some point in time θ and is defined by $F_e^+(\theta) := \int_0^\theta f_e^+(\xi)\,\mathrm{d}\xi$ and $F_e^-(\theta) := \int_0^\theta f_e^-(\xi)\,\mathrm{d}\xi$. The amount of flow in the queue of an arc e at time θ equals the difference between the amount of flow that has entered the queue before time θ and the flow that has left the queue up to this point in time. The former can be described by the amount of flow that has entered arc e at time $\theta - \tau_e$. In short, $z_e(\theta) = F_e^+(\theta - \tau_e) - F_e^-(\theta)$; see Lemma 14 in A.1. Since f_e^+ and f_e^- are bounded, the functions F_e^+, F_e^-, and z_e are Lipschitz continuous, and therefore almost everywhere differentiable due to Rademacher's theorem [11]. Considering that $f_e^+(\theta)$ and $f_e^-(\theta)$ are non-negative and $z_e'(\theta) \geq -\nu_e$ for all θ it follows that F_e^+ and F_e^- are non-decreasing and z_e cannot decrease faster than with slope $-\nu_e$.

We identify the flow with the non-negative reals $\mathbb{R}_{\geq 0}$, that is, each $\phi \in \mathbb{R}_{\geq 0}$ corresponds to an infinitesimally small flow particle. The natural ordering $\leq$ corresponds to the priority among the flow particles when entering the network, i.e., particle ϕ has priority over all $\phi' > \phi$. Consequently, all flow that wants to enter the network through the same source does this in order of priority. Note that the flow represented by the non-negative reals has a width of 1. That is, there is exactly one unit of flow associated with every unit interval $[a, a+1] \subseteq \mathbb{R}_{\geq 0}$. To distinguish between flow and time we write $\mathbb{R}_{\geq 0}$ for the ordered set of flow particles, mostly denoted by ϕ or φ, and $[0,\infty)$ for the time whose elements are *points in time*, often denoted by θ or ϑ.

A family of locally integrable functions $f_i \colon \mathbb{R}_{\geq 0} \to [0,1]$, for $i = 1, \dots, n$, is called *inflow distribution* if $\sum_{i=1}^n f_i(\phi) = 1$ for almost all $\phi \in \mathbb{R}_{\geq 0}$ and if each *cumulative source inflow* $F_i(\phi) := \int_0^\phi f_i(\varphi)\,\mathrm{d}\varphi$ is unbounded for $\phi \to \infty$. The function $f_i(\phi)$ describes the fraction of particle ϕ that enters the network trough s_i. The cumulative source inflow functions have to be unbounded in order to guarantee that the inflow rates at the sources never run dry.

Current shortest paths network. Given a flow over time $(f_e^+)_{e \in E}$ together with an inflow distribution $(f_i)_{i=1}^n$, the *arc travel time* for arc e is the function $T_e \colon [0,\infty) \to [0,\infty)$ that maps the entrance time θ to the exit time $T_e(\theta)$. More precisely, if a particle enters e at time θ, it traverses the arc first, which takes τ_e time, and then queues up and has to wait in line for $z_e(\theta + \tau_e)/\nu_e$ time units. Hence, $T_e(\theta) := \theta + \tau_e + z_e(\theta + \tau_e)/\nu_e$. We require the flow to satisfy the *first in first out (FIFO) condition* on every arc, that is, no particle can overtake other flow on an arc or in a queue. Suppose flow particle ϕ enters e at time θ, then the amount of flow which has entered e before ϕ is exactly the amount of flow that leaves e before time $T_e(\theta)$ when ϕ leaves the arc. In short $F_e^+(\theta) = F_e^-(T_e(\theta))$; see Lemma 14 in A.1.

For every $i = 1, \dots, n$, the *source arrival time function* maps each particle $\phi \in \mathbb{R}_{\geq 0}$ to the time it arrives at s_i and is given by $T_i(\phi) := F_i(\phi)/r_i$. Given an s_i-v path $P = (e_1, e_2, \dots, e_k)$ the *arrival time function* $T_P \colon \mathbb{R}_{\geq 0} \to [0,\infty)$ maps the particle ϕ to the time at which ϕ arrives at v if it traverses the path P, i.e., $T_P(\phi) := T_{e_k} \circ T_{e_{k-1}} \circ \dots \circ T_{e_1} \circ T_i(\phi)$. Since the functions z_e and F_i are Lipschitz continuous, the same holds for T_e,T_i, and T_P. Note that the queue length z_e cannot decrease faster than with slope $-\nu_e$ and, therefore all these T-functions are nondecreasing. Furthermore, all T-functions go to infinity for $\phi \to \infty$ since the queue lengths z_e are non-negative and the F_i are unbounded.

The *earliest arrival time function* $\ell_v \colon \mathbb{R}_{\geq 0} \to [0,\infty)$ of node $v \in V$ maps each particle ϕ to the earliest time $\ell_v(\phi)$ it can possibly reach node v. We have $\ell_v(\phi) = \min_{P \in \mathcal{P}_v} T_P(\phi)$, where $\mathcal{P}_v$ is the set of all paths from some source $s \in S^+$ to v. Note that these node-labels are also Lipschitz continuous, nondecreasing, and unbounded and that they are the unique solutions to the following Bellman equations:

$$\begin{aligned} \ell_{s_i}(\phi) &= \min\left(\{\, T_i(\phi) \,\} \cup \{\, T_e(\ell_u(\phi)) \mid e = us_i \in E \,\}\right) && \text{for } i = 1, \dots, n, \\ \ell_v(\phi) &= \min_{e=uv\in E} T_e(\ell_u(\phi)) && \text{for } v \in V \backslash S^+. \end{aligned} \tag{4}$$

This is well defined since all cycles in G have by assumption positive travel times.

For a fixed particle ϕ we call an arc $e = uv$ *active for* ϕ if $\ell_v(\phi) = T_e(\ell_u(\phi))$ holds. With E'_ϕ we denote the set of all active arcs for particle ϕ and the subgraph $G'_\phi = (V, E'_\phi)$ is called the *current shortest paths network*. Note that the current shortest paths network is always acyclic since the sum of transit times of each directed cycle is positive.

4 Multi-source single-sink Nash flows over time

For this section we only consider fluid queuing networks with exactly one sink t. A flow over time together with an inflow distribution corresponds to a strategy profile, where the strategy of each particle consists of a convex combination of S^+-t-paths. The following definition characterizes a Nash equilibrium.

▶ **Definition 1** (Nash flow over time). A tuple $f = ((f_e^+)_{e\in E}, (f_i)_{i=1}^n)$ consisting of a flow over time and an inflow distribution is a *Nash flow over time*, also called *dynamic equilibrium*, if the following two *Nash flow conditions* hold:

$$\ell_{s_i}(\phi) = T_i(\phi) \qquad \text{for all } i = 1, \dots, n \text{ and almost all } \phi \in \mathbb{R}_{\geq 0}, \tag{N1}$$

$$f_e^+(\theta) > 0 \;\Rightarrow\; \theta \in \ell_u(\Phi_e) \qquad \text{for all arcs } e = uv \in E \text{ and almost all } \theta \in [0, \infty), \tag{N2}$$

where $\Phi_e := \{\, \phi \in \mathbb{R}_{\geq 0} \mid e \in E'_\phi \,\}$ is the set of flow particles for which arc e is active.

Figuratively speaking, these two conditions mean, that entering the network through a source s_i is always a fastest way to reach s_i (N1) and that a Nash flow over time uses only active arcs (N2), and therefore only shortest paths to t. More precisely, particle ϕ reaches t at time $\ell_t(\phi)$ by using active arcs only, and $\ell_t(\phi)$ is the earliest time ϕ can possibly reach t under the assumption that the routes of all previous particles $\varphi < \phi$ are fixed. Since this is true for all particles, a Nash flow over time is indeed a Nash equilibrium.

▶ **Lemma 2.** *A tuple $f = ((f_e^+)_{e\in E}, (f_i)_{i=1}^n)$ of a flow over time and an inflow distribution is a Nash flow over time if, and only if, we have $F_e^+(\ell_u(\phi)) = F_e^-(\ell_v(\phi))$ and $F_i(\phi) = \ell_{s_i}(\phi) \cdot r_i$ for all arcs $e = uv \in E$, every $i = 1, \dots, n$, and all particles $\phi \in \mathbb{R}_{\geq 0}$.*

Proof. "⇒": Let $\xi \in [0, \phi]$ be the particle of largest value with $F_e^+(\ell_u(\xi)) = F_e^-(\ell_v(\phi))$. This ξ exists because of the intermediate value theorem, together with the fact that $F_e^+ \circ \ell_u$ is continuous and the following inequality, which follows by the monotonicity of F_e^- and Lemma 14:

$$F_e^+(\ell_u(0)) = 0 \quad \leq \quad F_e^-(\ell_v(\phi)) \quad \leq \quad F_e^-(T_e(\ell_u(\phi))) = F_e^+(\ell_u(\phi)).$$

Note that the second inequality is true because of $\ell_v(\phi) \leq T_e(\ell_u(\phi))$. In the case of $\xi = \phi$ we are done since $F_e^-(\ell_v(\phi)) = F_e^+(\ell_u(\xi)) = F_e^+(\ell_u(\phi))$. Suppose $\xi < \phi$. For all particles $\varphi \in (\xi, \phi]$ we know that $T_e(\ell_u(\varphi)) \neq \ell_v(\phi)$ because, otherwise, we had with Lemma 14 (ii) that $F_e^+(\ell_u(\varphi)) = F_e^-(T_e(\ell_u(\varphi))) = F_e^-(\ell_v(\phi))$ which would contradict the maximality of ξ. Hence, e is not active for particles in $(\xi, \phi]$ which implies $f_e^+(\theta) = 0$ for almost all $\theta \in \ell_u((\xi, \phi]) = (\ell_u(\xi), \ell_u(\phi)]$ since f is a Nash flow over time. This leads to

$$F_e^+(\ell_u(\phi)) - F_e^-(\ell_v(\phi)) = F_e^+(\ell_u(\phi)) - F_e^+(\ell_u(\xi)) = \int_{\ell_u(\xi)}^{\ell_u(\phi)} f_e^+(\vartheta)\,\mathrm{d}\vartheta = 0,$$

which finishes the first part. The second part follows directly from $\ell_{s_i}(\phi) = T_i(\phi) = F_i(\phi)/r_i$ for all $i = 1, \ldots, n$.

"$\Leftarrow$": Given a particle ϕ and an arc $e = uv$ such that e is not active for ϕ, i.e., $\ell_v(\phi) < T_e(\ell_u(\phi))$. The continuity of ℓ_v and $T_e \circ \ell_u$ implies that there is an $\varepsilon > 0$ with $\ell_v(\phi + \varepsilon) < T_e(\ell_u(\phi - \varepsilon))$ and e is not active for all particles in $[\phi - \varepsilon, \phi + \varepsilon]$. This, the fact that f_e^+ and f_e^- are non-negative, and Lemma 14 gives us

$$\begin{aligned} 0 \quad &\leq \quad \int_{\ell_u(\phi-\varepsilon)}^{\ell_u(\phi+\varepsilon)} f_e^+(\vartheta)\,\mathrm{d}\vartheta = \int_{T_e(\ell_u(\phi-\varepsilon))}^{T_e(\ell_u(\phi+\varepsilon))} f_e^-(\vartheta)\,\mathrm{d}\vartheta \leq \int_{\ell_v(\phi+\varepsilon)}^{T_e(\ell_u(\phi+\varepsilon))} f_e^-(\vartheta)\,\mathrm{d}\vartheta \\ &= \quad F_e^-(T_e(\ell_u(\phi+\varepsilon))) - F_e^-(\ell_v(\phi+\varepsilon)) = F_e^+(\ell_u(\phi+\varepsilon)) - F_e^-(\ell_v(\phi+\varepsilon)) \overset{(ii)}{=} 0. \end{aligned}$$

Hence, $f_e^+(\theta) = 0$ for almost all $\theta \in [\ell_u(\phi - \varepsilon), \ell_u(\phi + \varepsilon)]$. In other words, for almost all $\theta \in [0, \infty)$ it holds that $\theta \notin \ell_u(\Phi_e) \Rightarrow f_e^+(\theta) = 0$. This is true because for $\theta \geq \ell_u(0)$ we find a particle ϕ with $\ell_u(\phi) = \theta$, due to the fact, that ℓ_u is continuous and unbounded, and for all $\theta < \ell_u(0)$ we have $f_e^+(\theta) = 0$, since no flow can reach u faster than $\ell_u(0)$. Finally, we get $\ell_{s_i}(\phi) = T_i(\phi)$ since $\ell_{s_i}(\phi) \cdot r_i = F_i(\phi) = T_i(\phi) \cdot r_i$ for all $i = 1, \ldots, n$. This shows that f is a Nash flow over time, which finishes the proof. ◀

Lemma 2 motivates to consider the *underlying static flow* for every particle ϕ, which is defined by $x_e(\phi) \coloneqq F_e^+(\ell_u(\phi)) = F_e^-(\ell_v(\phi))$ and $x_i(\phi) \coloneqq F_i(\phi) = \ell_{s_i}(\phi) \cdot r_i$. For a fixed ϕ this is indeed a static S^+-t-flow since the integral of (3) over $[0, \ell_v(\phi)]$ yields

$$\sum_{e \in \delta^+(v)} x_e(\phi) - \sum_{e \in \delta^-(v)} x_e(\phi) = \begin{cases} 0 & \text{if } v \in V \backslash (S^+ \cup \{\, t \,\}), \\ \ell_{s_i}(\phi) \cdot r_i = x_i(\phi) & \text{if } v = s_i \in S^+. \end{cases} \tag{5}$$

Let x_e', x_i', and ℓ_v' denote the derivative functions, which exist almost everywhere, since the x- and ℓ-functions are Lipschitz continuous. It is possible to determine the inflow function of every arc $e = uv$ from these derivatives, since $x_e'(\phi) = f_e^+(\ell_u(\phi)) \cdot \ell_u'(\phi)$. Moreover, the inflow distribution is given by $f_i(\phi) = \ell_{s_i}'(\phi) \cdot r_i$. Consequently, a Nash flow over time is completely characterized by these derivatives. Note that differentiating (5) yields that $x'(\phi)$ also forms a static S^+-t-flow, which has very specific properties that are characterized in the following.

Thin flows with resetting for multiple sources and a single sink. A thin flow with resetting is a static flow defined on a subgraph of G characterizing the strategies of particles in a flow interval of a Nash flow over time. The definition of thin flows with resetting given in this article generalizes the thin flows with resetting introduced in [10] and the normalized thin flows with resetting from [2], in order to suit the multi-source setting.

Let $E' \subseteq E$ be a subset of arcs such that the subgraph $G' = (V, E')$ is acyclic and every node is reachable by a source within G'. Note that not every node needs to be able to reach sink t. Additionally, we consider a subset of arcs $E^* \subseteq E'$, called *resetting arcs*. Moreover, let $K(E', x_1', \ldots, x_n')$ be the set of all static S^+-t-flows in G' with inflow x_i' at source s_i for $x_i' \geq 0$ and $x_1' + \cdots + x_n' = 1$.

▶ **Definition 3** (Thin flow with resetting). A vector $(x'_i)_{i=1}^n$, with $x'_i \geq 0$ and $x'_1 + \dots + x'_n = 1$, and a static flow $(x'_e)_{e \in E} \in K(E', x'_1, \dots, x'_n)$ together with a node labeling $(\ell'_v)_{v \in V}$ is called *thin flow with resetting* on $E^* \subseteq E'$ if:

$$\ell'_{s_i} = x'_i / r_i \qquad \text{for all } i = 1, \dots, n, \tag{TF1}$$

$$\ell'_{s_i} \leq \min_{e = us_i \in E'} \rho_e(\ell'_u, x'_e) \qquad \text{for all } i = 1, \dots, n, \tag{TF2}$$

$$\ell'_v = \min_{e = uv \in E'} \rho_e(\ell'_u, x'_e) \qquad \text{for all } v \in V \backslash S^+, \tag{TF3}$$

$$\ell'_v = \rho_e(\ell'_u, x'_e) \qquad \text{for all } e = uv \in E' \text{ with } x'_e > 0, \tag{TF4}$$

$$\text{where} \qquad \rho_e(\ell'_u, x'_e) := \begin{cases} x'_e/\nu_e & \text{if } e = uv \in E^*, \\ \max\{\, \ell'_u, x'_e/\nu_e \,\} & \text{if } e = uv \in E' \backslash E^*. \end{cases}$$

The next theorem states that the derivatives of a Nash flow over time f form almost everywhere a thin flow with resetting on the arcs with positive queues. Recall that E'_ϕ is the subset of arcs that are active for ϕ, and let $E^*_\phi := \{\, e = uv \in E \mid z_e(\ell_u(\phi) + \tau_e) > 0 \,\}$ be the set of arcs where the particle ϕ would experience a queue.

▶ **Theorem 4.** *For a Nash flow over time $((f^+_e)_{e \in E}, (f_i)_{i=1}^n)$, the derivative labels $(x'_i(\phi))_{i=1}^n$ and $(x'_e(\phi))_{e \in E'_\phi}$ together with $(\ell'_v(\phi))_{v \in V}$ form a thin flow with resetting on E^*_ϕ in the current shortest paths network $G'_\phi = (V, E'_\phi)$, for almost all $\phi \in \mathbb{R}_{\geq 0}$.*

The intuitive idea is that x'_e/ν_e describes the *congestion* of arc e and $\rho_e(\ell'_u, x'_e)$ is the *congestion* of all paths to v using e. The higher this congestion is, the longer it will take for following particles to reach v, which is captured by a high derivative of the earliest arrival time ℓ'_v. If we have $\ell'_v < \rho_e(\ell'_u, x'_e)$ this means that e leaves the current shortest paths network, and therefore it cannot be used by following particles, i.e., $x'_e = 0$. A detailed proof is given in A.2.

The reverse of Theorem 4 is also true in the sense that we can use thin flows with resetting to construct a Nash flow over time. For this we first show that there always exists a thin flow with resetting for any acyclic graph and any subset of resetting arcs.

▶ **Theorem 5.** *Consider an acyclic graph $G' = (V, E')$ with sources S^+, sink t, capacities ν_e, and a subset of arcs $E^* \subseteq E'$ and suppose every node is reachable by a source. Then there exists a thin flow $((x'_i)_{i=1}^n, (x'_e)_{e \in E}, (\ell'_v)_{v \in V})$ with resetting on E^*.*

The proof is essentially given in [2] and is only slightly modified to fit the new definition of a thin flow with resetting. The key idea is to use a set-valued function in order to apply the Kakutan'i fixed-point theorem.

Constructing Nash flows. Note that in a dynamic equilibrium no particle can overtake any other particle, and therefore the choice of strategy for ϕ only depends on the strategies of the particles in $[0, \phi)$. So we may assume that the particles decide in order of priority. More precisely, given a Nash flow over time up to some $\phi \in \mathbb{R}_{\geq 0}$, it is possible to extend it by using a thin flow on the G'_ϕ with resetting on E^*_ϕ.

A *restricted Nash flow over time* on $[0, \phi]$ is a Nash flow over time where only the particles in $[0, \phi]$ are considered, i.e., for $i = 1, \dots, n$ we have $f_i(\varphi) = 0$ for all $\varphi > \phi$ and for each arc $e = uv \in E$ we have $f_e(\ell_u(\theta)) = 0$ for all $\theta > \ell_u(\phi)$. But the Nash flow conditions (N1) and (N2) are satisfied for almost all particles in $[0, \phi]$ and almost all times in $[0, \ell_u(\phi)]$.

Since all previous results carry over to restricted Nash flows over time, the earliest arrival times $(\ell_v)_{v \in V}$ are well-defined for particles in $[0, \phi]$, and therefore it is possible to determine $G'_\phi = (V, E'_\phi)$ and E^*_ϕ; see Lemma 15 in A.3. To extend a restricted Nash flow over time, we first compute a thin flow on G'_ϕ with resetting on E^*_ϕ, and then extend the labels linearly as follows. For some $\alpha > 0$ we get for all $v \in V$, $e \in E$, $i = 1, \dots, n$, and $\varphi \in (\phi, \phi{+}\alpha]$ that

$$\ell_v(\varphi) \coloneqq \ell_v(\phi) + (\varphi - \phi) \cdot \ell'_v \quad \text{and} \quad \begin{aligned} x_e(\varphi) &\coloneqq x_e(\phi) + (\varphi - \phi) \cdot x'_e, \\ x_i(\varphi) &\coloneqq x_i(\phi) + (\varphi - \phi) \cdot x'_i. \end{aligned}$$

Based on this we can extend the inflow function and the inflow distribution, which gives us

$$f^+_e(\theta) \coloneqq \frac{x'_e}{\ell'_u} \quad \text{for } \theta \in (\ell_u(\phi), \ell_u(\phi{+}\alpha)] \qquad \text{and} \qquad f_i(\varphi) \coloneqq x'_i \;=\; \ell'_{s_i} {\cdot} r_i \quad \text{for } \varphi \in (\phi, \phi{+}\alpha]$$

for all $e = uv \in E$ and all $i = 1, \dots, n$. Note that in the case of $\ell'_u = 0$ the time interval is empty. Furthermore, it turns out that $f^-_e(\theta) = x'_e/\ell'_v$ for all $\theta \in (\ell_v(\phi), \ell_v(\phi + \alpha)]$, which is formally shown in Lemma 16 in A.4. This extended flow over time together with the extended inflow distribution is called *α-extension* and it extends the Nash flow over time as long as the α stays within the the following bounds:

$$\ell_v(\phi) - \ell_u(\phi) + \alpha(\ell'_v - \ell'_u) \geq \tau_e \qquad \text{for all } e = uv \in E^* \tag{6}$$
$$\ell_v(\phi) - \ell_u(\phi) + \alpha(\ell'_v - \ell'_u) \leq \tau_e \qquad \text{for all } e = uv \in E \backslash E'. \tag{7}$$

The first inequality ensures that no flow can traverse an arc faster than its transit time. It holds with equality when the queue of e vanishes at time $\ell_u(\phi + \alpha)$. The second inequality makes sure that all non-active arcs are unattractive for all particles in $[\phi, \phi + \alpha)$. When it holds with equality the arc e becomes active for $\phi + \alpha$. When such an event occurs we must compute a new thin flow with resetting because either a resetting arc has become non-resetting or a non-active arc has become active. It is easy to see that there exists an $\alpha > 0$ that satisfies these inequalities since $\ell_v(\phi) > \ell_u(\phi) + \tau_e$ for arcs $e \in E^*_\phi$ and $\ell_v(\phi) < \ell_u(\phi) + \tau_e$ for arcs $e \notin E'_\phi$, as it is stated in Lemma 15 in A.3.

▶ **Lemma 6.** *The α-extension forms a flow over time and the extended ℓ-labels coincide with the earliest arrival times, i.e., satisfy the Bellman equations* (4) *for all $\varphi \in (\phi, \phi + \alpha]$.*

The flow conservation follows immediately from the flow conservation of x' and the Bellman equations are shown by distinguishing three cases. If the arc is non-active it stays non-active during the extended interval. For active, but non-resetting arcs that do not build up a queue, we obtain $\ell_v(\phi + \xi) \leq T_e(\ell_u(\phi + \xi))$ from (TF3) with equality if $\ell'_v = \rho_e(\ell'_u, x'_e)$. The same is true for resetting arcs or arcs that build up a queue, even though, the proof is a bit more technical. For a detailed proof we refer to A.5.

▶ **Theorem 7.** *Given a restricted Nash flow over time $((f^+_e)_{e \in E}, (f_i)^n_{i=1})$ on $[0, \phi]$ and $\alpha > 0$ satisfying* (6) *and* (7)*, the α-extension is a restricted Nash flow over time on $[0, \phi + \alpha]$.*

Proof. We have $\sum^n_{i=1} f_i(\theta) = \sum^n_{i=1} x'_i = 1$ for all $\theta \in (\phi, \phi{+}\alpha]$, which shows that $(f_i)^n_{i=1}$ is a restricted inflow distribution. Lemma 2 yields $F^+_e(\ell_u(\varphi)) = F^-_e(\ell_v(\varphi))$ and $F_i(\varphi) = \ell_{s_i}(\varphi) {\cdot} r_i$ for all $\varphi \in [0, \phi]$, so for $\xi \in (0, \alpha]$ it holds that

$$F^+_e(\ell_u(\phi + \xi)) = F^+_e(\ell_u(\phi)) + x'_e/\ell'_u \cdot \xi \cdot \ell'_u = F^-_e(\ell_v(\phi)) + x'_e/\ell'_v \cdot \xi \cdot \ell'_v = F^-_e(\ell_v(\phi + \xi)),$$
$$F_i(\phi + \xi) = F_i(\phi) + \xi \cdot x'_i = \ell_{s_i}(\phi) \cdot r_i + \xi \cdot \ell'_{s_i} \cdot r_i = \ell_{s_i}(\phi + \xi) \cdot r_i.$$

Again with Lemma 2 we have that the α-extension is a restricted Nash flow on $[0, \phi + \alpha]$. ◀

Finally, we show that this construction leads to a Nash flow over time.

▶ **Theorem 8.** *There exists a Nash flow over time with multiple sources and a single sink.*

Proof. In the first part we show that these α-extensions lead to a restricted Nash flow on $[0, \infty)$. In the second part we prove, that all cumulative source inflow functions are unbounded, which shows that we have, indeed, a Nash flow over time.

The process starts with the empty flow over time and the zero flow distribution, i.e., a restricted Nash flow over time for $[0, 0]$. By applying Theorem 7 iteratively and choosing α maximal according to (6) and (7), we obtain a sequence of restricted Nash flows over time f^i for $[0, \phi_i]$ for $i = 1, 2, \dots$, with strictly increasing $(\phi_i)_{i=1}^{\infty}$. In the case that this sequence has a finite limit, say ϕ_∞, we define a restricted Nash flow over time f^∞ for $[0, \phi_\infty]$ by using the point-wise limit of the x- and ℓ-labels, which exists due to monotonicity and Lipschitz continuity of these functions. Then the process can be restarted from this limit point.

Let $\mathcal{P}_G$ be the set of all particles $\phi \in \mathbb{R}_{\geq 0}$ for which there exists a restricted Nash flow over time on $[0, \phi]$ constructed as described above. The set $\mathcal{P}_G$ cannot have a maximal element because this could be extended by using Theorem 7. But it also cannot have an upper bound since the limit of any convergent sequence would be contained in this set. Therefore, there exists an unbounded increasing sequence $(\phi_i)_{i=1}^{\infty} \in \mathcal{P}_G$. From the corresponding restricted Nash flows over time we can construct the restricted Nash flow over time f on $[0, \infty)$ by taking the point-wise limit of the x- and ℓ-labels.

It remains to show that the inflow distribution of this restricted Nash flow over time is unbounded. For this we first show that the earliest arrival time ℓ_t is unbounded. There cannot be an upper bound B on ℓ_t since the flow rate into t is bounded by $N := \sum_{e \in \delta^-(t)} \nu_e$ and with the FIFO principle we obtain that no particle $\phi > N \cdot B$ reaches t before time $\phi / N > B$. Next, we show that all ℓ-labels are unbounded. Suppose this is not true. Since every node can reach t there would be an arc $e = uv$, where ℓ_u is bounded and ℓ_v is not. Since T_e is Lipschitz continuous $T_e \circ \ell_u$ would be bounded as well. But this contradicts that $\ell_v(\phi) \leq T_e(\ell_u(\phi))$ goes to infinity for $\phi \to \infty$. Hence, $F_i(\phi) = \ell_{s_i}(\phi) \cdot r_i$ is unbounded for every $i = 1, \dots, n$, which completes the proof. ◀

5 Multiple sinks with demands

In this section we consider a graph $G = (V, E)$ as before except that it can have multiple sinks $S^- := \{ t_1, \dots, t_m \}$ and demands $d_1, \dots, d_m > 0$ with $d_1 + \dots + d_m = 1$. We show how to construct a Nash flow over time in G where a share of d_j of the flow has t_j as destination.

Sub-flow over time decomposition. In the following we define a sub-flow over time, which is, intuitively, a colored proportion of a flow over time satisfying flow conservation. Given a flow over time $f = (f_e^+)_{e \in E}$ with queue functions $(z_e)_{e \in E}$, we consider a family of locally integrable and bounded inflow functions $g = (g_e^+)_{e \in E}$ with $g_e^+(\theta) \leq f_e^+(\theta)$ for almost all $\theta \in [0, \infty)$. The corresponding outflow functions are obtained by the following consideration. For a point in time $\vartheta \in [0, \infty)$ let $T_e^{-1}(\vartheta)$ be all times at which a particle could enter e in order to leave it at time ϑ. Whenever $T_e^{-1}(\vartheta)$ is not a singleton it is a proper interval and by (11) we have that $f_e^+(\theta) = 0$ for almost all $\theta \in T_e^{-1}(\vartheta)$. The *sub-outflow function* for arc $e \in E$ is defined as

$$g_e^-(\vartheta) := \begin{cases} f_e^-(\vartheta) \cdot \frac{g_e^+(\theta)}{f_e^+(\theta)} & \text{if } f_e^+(\theta) > 0 \text{ and } T_e^{-1}(\vartheta) = \{ \theta \}, \\ 0 & \text{else.} \end{cases} \tag{8}$$

In other words, if $g_e^+(\theta)/f_e^+(\theta) \in [0,1]$ is the inflow share of g at time θ, then the outflow share of g has the same value at time $T_e(\theta)$. We call $g = (g_e^+)_{e \in E}$ a *sub-flow over time of f* if for every $v \in V \backslash S^+$ and almost all $\theta \in [0,\infty)$ we have

$$\sum_{e\in\delta^-(v)} g_e^-(\theta) - \sum_{e\in\delta^+(v)} g_e^+(\theta) \leq \sum_{e\in\delta^-(v)} f_e^-(\theta) - \sum_{e\in\delta^+(v)} f_e^+(\theta). \tag{9}$$

Intuitively, this means that at every non-source node v the sub-flow over time g can at most "lose" as much flow as f does. Furthermore, we say g *conserves flow at node $v \in V\backslash S^+$* if $\sum_{e\in\delta^-(v)} g_e^-(\theta) - \sum_{e\in\delta^+(v)} g_e^+(\theta) = 0$ holds for almost all $\theta \in [0,\infty)$. Note that if f conserves flow at some node v, then g does so as well. We say g is an *S^+-t_j-sub-flow over time* if it conserves flow at all nodes in $V \backslash \{ t_j \}$.

Given an inflow distribution $(f_i)_{i=1}^n$ and a number $\gamma \in [0,1]$, a family of locally integrable functions $(g_i)_{i=1}^n$ with $g_i(\phi) \leq f_i(\phi)$ is called *sub-inflow distribution of value* γ if we have $\sum_{i=1}^n g_i(\phi) = \gamma$ for almost all $\phi \in \mathbb{R}_{\geq 0}$. To ensure that sub-flow is conserved at the sources we require the net flow leaving a source s_i at time $T_i(\phi)$ to be equal to the amount of flow distributed to s_i at time $T_i(\phi)$, which is $r_i \cdot g_i(\phi)/f_i(\phi) = g_i(\phi)/T_i'(\phi)$, whenever $f_i(\phi) > 0$ and 0 otherwise. More precisely, we say a sub-inflow distribution *matches* a sub-flow over time if for almost all $\phi \in \mathbb{R}_{\geq 0}$ and all $i = 1,\dots,n$ we have

$$T_i'(\phi) \cdot \left(\sum_{e\in\delta^+(s_i)} g_e^+(T_i(\phi)) - \sum_{e\in\delta^-(s_i)} g_e^-(T_i(\phi)) \right) = g_i(\phi).$$

In this case we also say that the sub-flow over time *conserves flow* at s_i and that the sub-flow over time g *has value* γ.

▶ **Definition 9** (Sub-flow over time decomposition). A family of sub-flows over time $(g_e^{j+})_{e\in E}$ and matching sub-inflow distributions $(g_i^j)_{i=1}^n$ of value γ_j, for $j = 1,\dots,m$, is called a *sub-flow over time decomposition of f with values $\gamma_1,\dots,\gamma_m$* if $\sum_{j=1}^m \gamma_j = 1$ and

$$g_e^{1+}(\theta) + \dots + g_e^{m+}(\theta) = f_e^+(\theta) \qquad \text{for all } e \in E \text{ and almost all } \theta \in [0,\infty).$$

Note that (8) implies $\sum_{j=1}^m g_e^{j-}(\xi) = f_e^-(\xi)$ for all $e \in E$ and almost all $\xi \in [0,\infty)$.

Nash flows over time with multiple sinks and demands. These sub-flow over time decompositions allow us to formalize Nash flows over time in the setting of multiple sinks with demands. Note that for the sake of clarity we omit the + and simply write f_e and g_e^j for the inflow functions for the remaining of this paper.

▶ **Definition 10** (Nash flow over time with demands). A tuple $f = ((f_e)_{e\in E}, (f_i)_{i=1}^n)$ consisting of a flow over time and an inflow distribution is a *Nash flow over time with demands* $d_1,\dots,d_m$ if it satisfies the Nash flow conditions (N1) and (N2) from Definition 1 and, furthermore, has a sub-flow over time decomposition $((g_e^j)_{e\in E}, (g_i^j)_{i=1}^n)_{j=1}^m$, such that $(g_e^j)_{e\in E}$ is an S^+-t_j-sub-flow over time of value d_j for all $j = 1,\dots,m$.

To construct a Nash flow over time with demands we add a super sink t to the graph and use a single-sink Nash flow over time as constructed in Section 4. For this let $\nu_{\min} := \min_{e\in E} \nu_e$ and $r_{\min} := \min_{i=1,\dots,n} r_i$ be the minimal capacity/inflow rate and $\sigma := \min\{ \nu_{\min}, r_{\min} \}$. For all $j = 1,\dots,m$ we define $\delta_j := \min_{s\in S^+} d(s,t_j)$, where $d(s,t_j)$ is the length of a shortest s-t_j-path according to the transit times. Furthermore, let $\delta_{\max} := \max_{j=1,\dots,m} \delta_j$ be the maximal distance to some sink t_j from its nearest source. We extend G by a super sink t and m new arcs $e_j := (t_j, t)$ with

$$\tau_{e_j} := \delta_{\max} - \delta_j \quad \text{and} \quad \nu_{e_j} := 1/2 \cdot d_j \cdot \sigma. \tag{10}$$

We denote the *extended graph* by $\bar{G} := (\bar{V}, \bar{E})$ with $\bar{V} := V \cup \{ t \}$ and $\bar{E} := E \cup \{ e_1, \dots, e_m \}$.

Note, that the new capacities are strictly smaller than all original capacities and all inflow rates and that they are proportional to the demands. Furthermore, we choose the transit times such that all new arcs are in the current shortest paths network for particle $\phi = 0$. The reason for the choice of σ is that for every thin flow with resetting (x', ℓ') in G we have $\ell'_v \leq 1/\sigma$ for all $v \in V$, which is shown in Lemma 17 in A.6.

We obtain a Nash flow over time with demands f by using a single-sink Nash flow over time $\bar{f}$ in $\bar{G}$, which exists due to Theorem 8. To prove this we first show that if all new arcs are active for some particle ϕ then there is a static flow decomposition of the thin flow with resetting x' with $x'_{e_j} = d_j$. This is formalized in the following lemma, where we write $x'\big|_E$ for the restriction of x' to the original graph G and $|\cdot|$ for the flow value of a static flow.

▶ **Lemma 11.** *Consider a thin flow with resetting (x', ℓ') in $\bar{G}$ where $\{ e_1, \dots, e_m \} \subseteq E'$, then there exists a static flow decomposition $x'\big|_E = x'^1 + \dots + x'^m$ such that each static flow x'^j conserves flow on all $v \in V \backslash (S^+ \cup \{ t_j \})$ and $\left|x'^j\right| = d_j$ for $j = 1, \dots, m$.*

Proof. Let $\mathcal{P}$ be the set of all S^+-t-paths in the current shortest paths network $G' = (V, E')$. Note, that G' is always acyclic and x' can, therefore, be described by the path vector $(x'_P)_{P \in \mathcal{P}}$ due to the well-known flow decomposition theorem. For $j = 1, \dots, m$ let $\mathcal{P}_j$ be the set of all S^+-t-paths that contain e_j. These sets form a partition of $\mathcal{P}$ since every path has to use exactly one of the new arcs. By setting $x'^j := \sum_{P \in \mathcal{P}_j} x'_P\big|_E$ we obtain the desired decomposition of x', because $x'_P\big|_E$ for $P \in \mathcal{P}_j$ conserves flow on all nodes except the ones in $S^+ \cup \{ t_j \}$ and the same is true for their sums.

Since x'^j sends $\left|x'^j\right|$ flow units from S^+ over e_j to t_j we have $\left|x'^j\right| = x'_{e_j}$. It remains to show that $x'_{e_j} = d_j$ for all $j = 1, \dots, m$. Suppose this is not true. Since x' sends exactly $d_1 + \dots + d_m = 1$ flow units from S^+ to t, there has to be an index $a \in \{ 1, \dots, m \}$ with $x'_{e_a} > d_a$ and an index $b \in \{ 1, \dots, m \}$ with $x'_{e_b} < d_b$.

With Lemma 17 it follows that

$$\ell'_{t_b} \leq \frac{1}{\sigma} \overset{(10)}{<} \frac{d_a}{\nu_{e_a}} < \frac{x'_{e_a}}{\nu_{e_a}} \overset{\text{(TF4)}}{\leq} \ell'_t \quad \text{as well as} \quad \frac{x'_{e_b}}{\nu_{e_b}} \overset{(10)}{=} \underbrace{\frac{x'_{e_b}}{d_b}}_{<1} \cdot \frac{2}{\sigma} < \underbrace{\frac{x'_{e_a}}{d_a}}_{>1} \cdot \frac{2}{\sigma} \overset{(10)}{=} \frac{x'_{e_a}}{\nu_{e_a}} \overset{\text{(TF4)}}{\leq} \ell'_t.$$

But this is a contradiction, because (TF3) yields that $\ell'_t = \min_{j=1,\dots,m} \rho_{e_j}(\ell'_{t_j}, x'_{e_j})$ and the last two equations show $\rho_{e_b}(\ell'_{t_b}, x'_{e_b}) < \ell'_t$. Hence, we have $\left|x'^j\right| = d_j$ for all $j = 1, \dots, m$, which finishes the proof. ◀

▶ **Lemma 12.** *In a Nash flow over time $\bar{f}$ in $\bar{G}$ the new arcs $e_1, \dots, e_m$ are active for all particles $\phi \in \mathbb{R}_{\geq 0}$.*

Proof. For particle $\phi = 0$ there are no queues yet, and therefore the exit time for each arc e is $T_e(\theta) = \theta + \tau_e$. Hence, $\ell_{t_j}(0) = \delta_j$ for all $j = 1, \dots, m$ and by construction we have $\ell_t(0) = \ell_{t_j}(0) + \tau_{e_j} = T_{e_j}(\ell_{t_j}(0))$ for $j = 1, \dots, m$. Therefore, all arcs e_j are active in the beginning and also during the first thin flow phase because by Lemma 11 we have $x'_{e_j} > 0$ for the first thin flow with resetting which implies that e_j stays active.

Suppose now for contradiction that there are particles for which not all new arcs are active. Let ϕ_0 be the infimum of these particles. By the consideration above we have $\phi_0 > 0$ and Lemmas 17 and 11 imply

$$f^+_{e_j}(\ell_{t_j}(\phi)) = \frac{x'_{e_j}}{\ell'_{t_j}} \geq x'_{e_j} \cdot \sigma = d_j \cdot \sigma \overset{(10)}{>} \nu_{e_j}$$

for almost all $\phi \in [0, \phi_0)$ and all $j = 1, \dots, m$. Hence, (1) yields $z'_{e_j}(\ell_{t_j}(\phi) + \tau_{e_j}) = f^+_{e_j}(\ell_{t_j}(\phi)) - \nu_{e_j} > 0$ and, together with the fact that $\ell'_{t_j} > 0$ (due to the positive throughput of x' at t_j), we obtain

$$\frac{\mathrm{d}}{\mathrm{d}\phi} z_{e_j}(\ell_{t_j}(\phi) + \tau_{e_j}) = z'_{e_j}(\ell_{t_j}(\phi) + \tau_{e_j}) \cdot \ell'_{t_j} > 0.$$

In other words, a queue is building up within $[0, \phi_0)$, and therefore $z_{e_j}(\ell_{t_j}(\phi_0) + \tau_{e_j}) > 0$ for all $j = 1, \dots, m$. But the continuity of z_{e_j} implies that there will be positive queues for all $\phi \in [\phi_0, \phi_0 + \varepsilon]$ for sufficiently small $\varepsilon > 0$. By Lemma 15 this implies that all new arcs are active during this interval contradicting that ϕ_0 is an infimum. ◀

By means of the previous lemmas we can finally prove that the Nash flow over time in $\bar{G}$ induces a Nash flow over time with demands in G.

▶ **Theorem 13.** *Let $\bar{f}$ be a S^+-t-Nash flow over time in $\bar{G}$. The flow over time $f := \bar{f}\big|_E$ on the original network together with the inflow distribution of $\bar{f}$ is a Nash flow over time with demands $d_1, \dots, d_m$.*

Proof. We have to show that the thin flow decompositions of the particles in $\mathbb{R}_{\geq 0}$ correspond to a sub-flow over time decomposition of the Nash flow over time. Throughout this proof we denote $\delta^-(v)$ and $\delta^+(v)$ for the in- and out-going arcs of v within the original network G. Let $I := [a, b)$ be an interval such that the thin flow with resetting is constant (x', ℓ') for all particles in I. For every node v we denote by $I_v := [\ell_v(a), \ell_v(b))$ the interval of local times of particles in I. By Lemma 12 all new arcs $e_1, \dots, e_m$ are active. Let $x'^1, \dots, x'^m$ be the thin flow decomposition given by Lemma 11. The corresponding decomposition for the Nash flow over time with demands is constructed by setting

$$\begin{aligned} g^j_e(\theta) &:= \frac{x'^j_e}{\ell'_u} && \text{for } \theta \in I_u \\ g^j_i(\phi) &:= \sum_{e \in \delta^+(s_i)} x'^j_e - \sum_{e \in \delta^-(s_i)} x'^j_e && \text{for } \varphi \in I \end{aligned}$$

for all $j = 1, \dots, m$, every $e = uv \in E$, and all $i = 1, \dots, n$. Note that if $\ell'_u = 0$ we have $\ell_u(a) = \ell_u(b)$, and therefore I_u is empty. By setting $g^j_e(\theta) := 0$ for all $\theta < \ell_u(0)$ we obtain well-defined functions g^j_e.

First, we show that g^j satisfies the sub-flow over time properties and conserves flow at all nodes except $S^+ \cup \{ t_j \}$ for all $\theta \in I_u$.

Given an arc $e = uv$ we obviously have for all $\theta \in I_u$ that

$$g^j_e(\theta) = x'^j_e / \ell'_u \leq x'_e / \ell'_u = f_e(\theta).$$

If $x'_e > 0$ we have $f_e(\theta) = x'_e / \ell'_u > 0$ for almost all $\theta \in I_u$ and by the definition of g^{j-} we get for almost all $\xi \in I_v = T_e(I_u)$ and $\theta \in I_u$, the unique value with $\xi = T_e(\ell_u(\phi))$, that

$$g^{j-}_e(\xi) = f^-_e(\xi) \cdot \frac{g^j_e(\theta)}{f_e(\theta)} = \frac{x'_e}{\ell'_v} \cdot \frac{x'^j_e}{\ell'_u} \cdot \frac{\ell'_u}{x'_e} = \frac{x'^j_e}{\ell'_v}.$$

But this equality also holds if $x'_e = 0$ because in this case it holds that $f_e(\theta) = 0$ for almost all $\theta \in I_u$, and therefore we have by definition that $g_e^{j-}(\xi) = 0$. The following equation shows that g^j conserves flow at all nodes $v \in V \backslash S^+ \cup \{ t_j \}$ for almost all $\theta \in I_v$

$$\sum_{e\in\delta^-(v)} g_e^{j-}(\theta) - \sum_{e\in\delta^+(v)} g_e^j(\theta) = \sum_{e\in\delta^-(v)} \frac{x_e'^j}{\ell_v'} - \sum_{e\in\delta^+(v)} \frac{x_e'^j}{\ell_v'} = \frac{1}{\ell_v'} \cdot \left(\sum_{e\in\delta^-(v)} x_e'^j - \sum_{e\in\delta^+(v)} x_e'^j \right) = 0,$$

where the last equality holds because of the flow conservation of x'^j at v. To show (9) it remains to prove it for t_j, which is true because for all $\theta \in I_{t_j}$ we have

$$\sum_{e\in\delta^-(t_j)} g_e^{j-}(\theta) - \sum_{e\in\delta^+(t_j)} g_e^j(\theta) = \frac{x_{e_j}}{\ell_{t_j}'} = \bar{f}_{e_j}(\theta) = \sum_{e\in\delta^-(t_j)} f_e^-(\theta) - \sum_{e\in\delta^+(t_j)} f_e(\theta).$$

Next, we show that $(g_i^j)_{i=1}^n$ is a matching sub-inflow distribution for all $j = 1, \dots, m$ with values d_j for all $\phi \in I$. In the case of $T'(\phi) \overset{\text{(N1)}}{=} \ell_{s_i}' > 0$ it holds that

$$\left(\sum_{e\in\delta^+(s_i)} g_e^j(\ell_{s_i}(\phi)) - \sum_{e\in\delta^-(s_i)} g_e^{j-}(\ell_{s_i}(\phi)) \right) T_i'(\phi) = \left(\sum_{e\in\delta^+(s_i)} \frac{x_e'^j}{\ell_{s_i}'} - \sum_{e\in\delta^-(s_i)} \frac{x_e'^j}{\ell_{s_i}'} \right) \ell_{s_i}' = g_i^j(\phi).$$

In the case of $\ell_{s_i}' = 0$ this is also true since both sides are equal to 0. By Lemma 11 we obtain for all $\phi \in I$ that

$$\sum_{i=1}^n g_i^j(\phi) = \sum_{i=1}^n \left(\sum_{e\in\delta^+(s_i)} x_e'^j - \sum_{e\in\delta^-(s_i)} x_e'^j \right) = |x'^j| = d_j.$$

Finally, we show that the family $(g^j)_{j=1}^m$ together with the matching sub-inflow distributions fulfills the sub-flow over time decomposition conditions for all $\theta \in I_u$. Clearly, $\sum_{j=1}^m d_j = 1$ and for all $e = uv \in E$ we have

$$\sum_{j=1}^m g_e^j(\theta) = \sum_{j=1}^m \frac{x_e'^j}{\ell_u'} = \frac{x_e'}{\ell_u'} = f_e(\theta).$$

Note that all these previous conditions hold for all $\phi \in \mathbb{R}_{\geq 0}$ and all $\theta \in [0, \infty)$ because either $\theta < \ell_v(0)$, where all in and out flow at v is 0, or θ is element of the local times I_u of some particle interval I. Hence, $(g^j)_{j=1^m}$ is a sub-flow over time decomposition of f with values $d_1, \dots, d_m$, where g^j is an S^+-t_j-sub-flow over time. Since $\bar{f}$ is a Nash flow over time f satisfies the Nash flow conditions (N1) and (N2) as well, and therefore f is a Nash flow over time with demands $d_1, \dots, d_m$. ◀

6 Conclusion and outlook

We showed that the Nash flow over time introduced in [10] can be extended to our multi-terminal setting, for which we uncoupled the flow particles from their entering times and introduced inflow distributions instead. Furthermore, the proper definition of a sub-flow-structure and a super-sink-construction allowed us to have Nash flows over time with multiple sinks and demands. Nonetheless the much more challenging question about the structure of a dynamic equilibrium in a setting with multiple origin-destination-pairs remains open. There are also further interesting aspects that are unsolved in the original setting, such as the computational complexity of thin flows with resetting or the question if the number of thin flow phases is finite within a Nash flow over time. Last but not least the very interesting question if the price of anarchy is bounded or not remains open, despite some promising progress in recent time.

References

1 R. Cominetti, J. Correa, and O. Larré. Existence and uniqueness of equilibria for flows over time. In L. Aceto, M. Henzinger, and J. Sgall, editors, *Automata, Languages and Programming*, volume 6756 of *Lecture Notes in Computer Science*, pages 552–563. Springer Berlin Heidelberg, 2011. `doi:10.1007/978-3-642-22012-8_44`.

2 R. Cominetti, J. Correa, and O. Larré. Dynamic equilibria in fluid queueing networks. *Operations Research*, 63:21–34, 2015. `doi:10.1287/opre.2015.1348`.

3 R. Cominetti, J. Correa, and N. Olver. Long term behavior of dynamic equilibria in fluid queuing networks. In F. Eisenbrand and J. Könemann, editors, *Integer Programming and Combinatorial Optimization*, volume 10328 of *Lecture Notes in Computer Science*, pages 161–172. Springer, 2017. `doi:10.1007/978-3-319-59250-3_14`.

4 L. R. Ford and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958. `doi:10.1287/opre.6.3.419`.

5 L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

6 A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Theoretical Computer Science*, 379:387–404, 2007. `doi:10.1016/j.tcs.2007.02.046`.

7 B. Hoppe. *Efficient dynamic network flow algorithms*. PhD thesis, Cornell University, 1995.

8 B. Hoppe and É. Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25:36–62, 2000. `doi:10.1287/moor.25.1.36.15211`.

9 A. Horni, K. Nagel, and K. Axhausen, editors. *Multi-Agent Transport Simulation MATSim*. Ubiquity Press, London, 2016. `doi:10.5334/baw`.

10 R. Koch and M. Skutella. Nash equilibria and the price of anarchy for flows over time. *Theory of Computing Systems*, 49:323–334, 2009. `doi:10.1007/978-3-642-04645-2_29`.

11 H. Rademacher. Über partielle und totale Differenzierbarkeit von Funktionen mehrerer Variabeln und über die Transformation der Doppelintegrale. *Mathematische Annalen*, 79(4):340–359, 1919. `doi:10.1007/BF01498415`.

12 B. Ran and D. E. Boyce. *Modelling Dynamic Transportation Networks*. Springer, Berlin, 1996. `doi:10.1007/978-3-642-80230-0`.

13 T. Roughgarden. *Selfish Routing and the Price of Anarchy*. MIT Press, 2005.

14 M. Schlöter and M. Skutella. Fast and memory-efficient algorithms for evacuation problems. In P. N. Klein, editor, *Proceedings of the 28th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 821–840. SIAM, 2017. `doi:10.1137/1.9781611974782.52`.

15 M. Skutella. An introduction to network flows over time. In *Research Trends in Combinatorial Optimization*, pages 451–482. Springer, 2009. `doi:10.1007/978-3-540-76796-1_21`.

A Appendix

A.1 Cumulative flows and queues

▶ **Lemma 14.** *For a given arc $e = uv \in E$ the following is true for all times $\theta \geq 0$:*

(i) $z_e(\theta) = F_e^+(\theta - \tau_e) - F_e^-(\theta)$

(ii) $F_e^+(\theta) = F_e^-(T_e(\theta))$

A proof can be found in [2, Section 2.2] and [10, Proposition 2].

A.2 Proof of Theorem 4

▶ **Theorem 4.** *For a Nash flow over time $((f_e^+)_{e\in E}, (f_i)_{i=1}^n)$, the derivative labels $(x_i'(\phi))_{i=1}^n$ and $(x_e'(\phi))_{e\in E_\phi'}$ together with $(\ell_v'(\phi))_{v\in V}$ form a thin flow with resetting on E_ϕ^* in the current shortest paths network $G_\phi' = (V, E_\phi')$, for almost all $\phi \in \mathbb{R}_{\geq 0}$.*

Proof. In Lemma 15 we showed that G'_ϕ and E^*_ϕ satisfy the preconditions. Furthermore, we have $x'_i(\phi) = f_i(\phi) \geq 0$ for all $i = 1, \dots, n$ and $\sum_{i=1}^n x'_i(\phi) = \sum_{i=1}^n f_i(\phi) = 1$ for almost all $\phi \in \mathbb{R}_{\geq 0}$. It remains to show that the equations (TF1) to (TF4) are satisfied for almost all particles. For this let ϕ be a particle such that for all $e = uv$ the derivatives of x_e, ℓ_v, and $T_e \circ \ell_u$ exist and $x'_e(\phi) = f_e^+(\ell_u(\phi)) \cdot \ell'_u(\phi) = f_e^-(\ell_v(\phi)) \cdot \ell'_v(\phi)$, which is almost everywhere. From Lemma 2 follows (TF1) directly.

For (TF2) and (TF3) first note that since z_e is Lipschitz continuous, so is T_e. We thus obtain from (1) that the derivative of $T_e(\theta)$ is almost everywhere

$$T'_e(\theta) = \begin{cases} \frac{f_e^+(\theta)}{\nu_e} & \text{if } z_e(\theta + \tau_e) > 0, \\ \max\left\{ \frac{f_e^+(\theta)}{\nu_e}, 1 \right\} & \text{if } z_e(\theta + \tau_e) = 0. \end{cases} \tag{11}$$

In the case of $z_e(\ell_u(\phi) + \tau_e) > 0$ we have

$$\frac{\mathrm{d}}{\mathrm{d}\phi} T_e(\ell_u(\phi)) = T'_e(\ell_u(\phi)) \cdot \ell'_u(\phi) \overset{(11)}{=} \left(\frac{f_e^+(\ell_u(\phi))}{\nu_e} \right) \cdot \ell'_u(\phi) = \frac{x'_e(\phi)}{\nu_e}$$

and if $z_e(\ell_u(\phi) + \tau_e) = 0$, it holds that

$$\frac{\mathrm{d}}{\mathrm{d}\phi} T_e(\ell_u(\phi)) \overset{(11)}{=} \max\left\{ \frac{f_e^+(\ell_u(\phi))}{\nu_e}, 1 \right\} \cdot \ell'_u(\phi) = \max\left\{ \frac{x'_e(\phi)}{\nu_e}, \ell'_u(\phi) \right\}.$$

Since the first case is equivalent to $e \in E^*_\phi$ and the second to $e \in E'_\phi \backslash E^*_\phi$ we obtain

$$\frac{\mathrm{d}}{\mathrm{d}\phi} T_e(\ell_u(\phi)) = \rho_e(\ell'_u(\phi), x'_e(\phi)).$$

This equality together with the Bellman equations (4) and the differentiation rule for a minimum, i.e., $\ell'_v(\phi) = \min_{uv \in E'_\phi} T'_{uv}(\phi)$, provides

$$\ell'_{s_i}(\phi) \leq \min_{e = us_i \in E'_\phi} \rho_e(\ell'_u(\phi), x'_e(\phi)) \quad \text{and} \quad \ell'_v(\phi) = \min_{e = uv \in E'_\phi} \rho_e(\ell'_u(\phi), x'_e(\phi)).$$

For (TF4) suppose $x'_e(\phi) = f_e^-(\ell_v(\phi)) \cdot \ell'_v(\phi) > 0$. With (2) we obtain

$$\begin{aligned} \ell'_v(\phi) = \frac{x'_e(\phi)}{f_e^-(\ell_v(\phi))} &= \begin{cases} \dfrac{x'_e(\phi)}{\min\left\{ f_e^+(\ell_u(\phi)), \nu_e \right\}} & \text{if } z_e(\ell_u + \tau_e) = 0, \\ \dfrac{x'_e(\phi)}{\nu_e} & \text{else,} \end{cases} \\ &= \begin{cases} \max\left\{ \ell'_u, \dfrac{x'_e(\phi)}{\nu_e} \right\} & \text{if } e \in E'_\phi \backslash E^*_\phi, \\ \dfrac{x'_e(\phi)}{\nu_e} & \text{if } e \in E^*_\phi, \end{cases} \\ &= \rho_e(\ell'_u(\phi), x'_e(\phi)). \end{aligned}$$

This shows that the derivatives $(x'_i(\phi))_{i=1}^n$, $(x'_e(\phi))_{e \in E'_\phi}$, and $(\ell'_v(\phi))_{v \in V}$ form a thin flow with resetting. ◀

A.3 Characterization of active and resetting arcs

This lemma shows, among other facts, that every arc with a positive queue has to be active.

▶ **Lemma 15.** *Consider a Nash flow over time f with earliest arrival times $(\ell_v)_{v\in V}$. For every particle $\phi \in \mathbb{R}_{\geq 0}$, the following statements are true:*
(i) $E^*_\phi \subseteq E'_\phi$
(ii) $E'_\phi = \{\, e = uv \in E \mid \ell_v(\phi) \geq \ell_u(\phi) + \tau_e \,\}$
(iii) $E^*_\phi = \{\, e = uv \in E \mid \ell_v(\phi) > \ell_u(\phi) + \tau_e \,\}$
(iv) *The graph $G'_\phi = (V, E'_\phi)$ is acyclic and every node is reachable by a source.*

For a proof we refer to [2, Proposition 2].

A.4 Extended outflow functions

▶ **Lemma 16.** *Let $((f^+_e)_{e\in E}, (f_i)^n_{i=1})$ be a restricted Nash flow over time on $[0,\phi]$ and let $\alpha > 0$ satisfy (6) and (7). Then the outflow functions of the α-extension satisfy $f^-_e(\theta) = \frac{x'_e}{\ell'_v}$ for all $e = uv \in E$ and almost all $\theta \in\ (\ell_v(\phi), \ell_v(\phi+\alpha)]$ and $f^-_e(\theta) = 0$ for $\theta > \ell_v(\phi+\alpha)$.*

Proof. Note that throughout this proof $\ell_v(\varphi)$ for $\varphi > \phi$ is not the earliest arrival time, but the linear extension $\ell_v(\varphi) := \ell_v(\phi) + (\varphi - \phi)\cdot \ell'_v$. Let $I := (\phi, \phi+\alpha]$ be the flow of interest and $I_v := (\ell_v(\phi, \ell_v(\phi+\alpha)]$ for all nodes v. The particles in $[0,\phi]$ do not interfere with the outflow function f^+_v within I_v, since otherwise the restricted Nash flow over time would not have chosen the fastest direction. We divide the proof into three cases.

Case 1: $x'_e = 0$.
Since $f^+_e(\theta) = x'_e/\ell'_u = 0$ for all $\theta \in I_u$ we have that $f^-_e(\theta) = 0 = x'_e/\ell'_v$ for all $\theta \in I_v$ and of course $f^-_e(\theta) = 0$ for $\theta > \ell_u(\phi+\alpha)$.

Case 2: $x'_e > 0$, $e \notin E^*$ and $x'_e/\nu_e \leq \ell'_u$.
We know that e is active during I and that $f^+_e(\theta) = x'_e/\ell'_u \leq \nu_e$ for $\theta \in I_u$. Furthermore, there is no queue at the beginning and no queue is building up. Therefore, we have $\ell_v(\phi) = \ell_u(\phi) + \tau_e$. The definition of thin flows with resetting provides $\ell'_u = \ell'_v$ and together with the definition of the extension we obtain

$$\ell_u(\phi+\alpha) + \tau_e = \ell_u(\phi) + \alpha\cdot\ell'_u + \tau_e = \ell_v(\phi) + \alpha\cdot\ell'_v = \ell_v(\phi+\alpha).$$

Hence, the last flow entering e at time $\ell_u(\phi+\alpha)$ leaves the edge at time $\ell_v(\phi+\alpha)$ and since the outflow rate at time $\theta \in I_v$ equals the inflow rate at time $\theta - \tau_e \in I_u$ we get

$$f^-_e(\theta) = f^+_e(\theta - \tau_e) = \frac{x'_e}{\ell'_u} = \frac{x'_e}{\ell'_v}.$$

Furthermore, no flow enters e after $\ell_u(\phi+\alpha)$, and therefore the outflow function is zero after $\ell_v(\phi+\alpha)$.

Case 3: $x'_e > 0$ and ($e \in E^*$ or $x'_e/\nu_e > \ell'_u$).
This means there is either a queue at the beginning and throughout the phase or there is no queue at the beginning but immediately after ϕ a queue will build up. In either case, e is active for all particles in I and $\ell'_v = x'_e/\nu_e$. The inflow rate is $f^+_e(\theta) = x'_e/\ell'_u$ for all I_u, and therefore the amount of flow entering e during this interval is

$$A := x'_e/\ell'_u\cdot(\ell_u(\phi+\alpha) - \ell_u(\phi)) = x'_e/\ell'_u\cdot(\ell_u(\phi) + \alpha\cdot\ell'_u - \ell_u(\phi)) = x'_e\cdot\alpha.$$

Since ϕ leaves e at time $\ell_v(\phi)$ and f_e^- operates at capacity rate the last particle $\phi + \alpha$ leaves e at time

$$\ell_v(\phi) + A/\nu_e = \ell_v(\phi) + \alpha \cdot x'_e/\nu_e = \ell_v(\phi) + \alpha \cdot \ell'_v = \ell_v(\phi + \alpha).$$

Therefore, we have for all $\theta \in I_v$ that

$$f_e^-(\theta) = \nu_e = \frac{x'_e}{\ell'_v}.$$

Since no flow is entering after $\ell_u(\phi + \alpha)$ and particle $\phi + \alpha$ leaves e at time $\ell_v(\phi + \alpha)$ the outflow function is zero afterwards. This completes the proof. ◀

A.5 Proof of Lemma 6

▶ **Lemma 6.** *The α-extension forms a flow over time and the extended ℓ-labels coincide with the earliest arrival times, i.e., satisfy the Bellman equations (4) for all $\varphi \in (\phi, \phi + \alpha]$.*

Proof. In order to prove that the α-extension forms a flow over time we have to show that the flow conservation is fulfilled at every $v \in V \setminus \{t\}$, which is true because for all $\theta \in (\ell_v(\phi), \ell_v(\phi + \alpha)]$ it holds that

$$\begin{aligned}\sum_{e \in \delta^+(v)} f_e^+(\theta) - \sum_{e \in \delta^-(v)} f_e^-(\theta) &= \sum_{e \in \delta^+(v)} x'_e/\ell'_v - \sum_{e \in \delta^-(v)} x'_e/\ell'_v \\ &= \begin{cases} 0 & \text{if } v \in V \setminus (S^+ \cup \{t\}) \\ x'_i/\ell'_v = r_i & \text{if } v = s_i \in S^+. \end{cases}\end{aligned}$$

For $\theta > \ell_v(\phi + \alpha)$ all functions as well as the inflow rates are zero, and therefore the flow conservation holds as well.

For the second part we show that the Bellman equations (4) for the earliest arrival times hold. Given an arc $e = uv \in E$, we distinguish between three cases.

Case 1: $e \in E \setminus E'_\phi$.
Since α satisfies equation (7) it is satisfied for all $\xi \in (0, \alpha]$ and hence,

$$\ell_v(\phi + \xi) = \ell_v(\phi) + \xi \cdot \ell'_v \overset{(7)}{\leq} \ell_u(\phi) + \xi \cdot \ell'_u + \tau_e \leq T_e(\ell_u(\phi) + \xi \cdot \ell'_u) = T_e(\ell_u(\phi + \xi)).$$

Case 2: $e \in E'_\phi \setminus E^*_\phi$ and $\ell'_u \geq x'_e/\nu_e$.
Since e is active we have $\ell_v(\phi) = T_e(\ell_u(\phi)) = \ell_u(\phi) + \tau_e$ and (TF3) implies $\ell'_v \leq \ell'_u$. There is no queue building up, which means $z_e(\ell_u(\phi + \xi) + \tau_e) = 0$ for all $\xi \in (0, \alpha]$. Combining these yields

$$\ell_v(\phi + \xi) = \ell_v(\phi) + \xi \cdot \ell'_v \overset{\text{(TF3)}}{\leq} \ell_u(\phi) + \tau_e + \xi \cdot \ell'_u = \ell_u(\phi + \xi) + \tau_e = T_e(\ell_u(\phi + \xi)).$$

Case 3: $e \in E^*_\phi$ or ($e \in E'_\phi$ and $\ell'_u < x'_e/\nu_e$).
Again, e is active, which means $\ell_v(\phi) = T_e(\ell_u(\phi)) = \ell_u(\phi) + \tau_e + z_e(\ell_u(\phi) + \tau_e)/\nu_e$. Additionally, $e \in E^*_\phi$ or $x'_e/\ell'_u \leq \nu_e$ together with the thin flow condition (TF3) implies $\ell'_v \leq x'_e/\nu_e$. Since $f_e^+(\ell_u(\phi)) - \nu_e = x'_e/\ell'_u - \nu_e > 0$, equation (1) implies $z'_e(\ell_u(\phi) + \tau_e) =$

$f_e^+(\ell_u(\phi)) - \nu_e = x'_e/\ell'_u - \nu_e$. Rearranging gives us, $x'_e/\nu_e = z'_e(\ell_u(\phi)+\tau_e)\cdot\ell'_u/\nu_e + \ell'_u$. Hence, for all $\xi \in (0,\alpha]$ we obtain with (TF3) that

$$\begin{aligned}
\ell_v(\phi+\xi) &= \ell_v(\phi) + \xi\cdot\ell'_v \\
&\le \ell_v(\phi) + \xi\cdot x'_e/\nu_e \\
&= \ell_u(\phi) + \tau_e + z_e(\ell_u(\phi)+\tau_e)/\nu_e + \xi\cdot(z'(\ell_u(\phi)+\tau_e)\cdot\ell'_u/\nu_e + \ell'_u) \\
&= \ell_u(\phi+\xi) + \tau_e + z_e(\ell_u(\phi)+\tau_e+\xi\cdot\ell'_u)/\nu_e \\
&= T_e(\ell_u(\phi+\xi)).
\end{aligned}$$

This shows that there is no arc with an exit time earlier than the earliest arrival time, and therefore the left hand side of the Bellman equations is always smaller or equal to the right hand side. It remains to show that the equations hold with equality. For a source s_i we have $x'_i = f_i(\phi) = r_i\cdot T'_i(\phi)$, and therefore

$$\ell_{s_i}(\phi+\xi) = \ell_{s_i}(\phi) + \xi\cdot\ell'_{s_i} = T_i(\phi) + \xi\cdot x'_i/r_i = T_i(\phi) + \xi\cdot T'_i(\phi) = T_i(\phi+\xi)$$

for all $\xi \in (0,\alpha]$. Hence, entering the network at a specific source is always a fastest option to reach it. For every node $v \in V\backslash S^+$ there is at least one arc $e \in E'$ with $\ell'_v = \rho(\ell'_u, x'_e)$ in the thin flow due to (TF3). No matter if this arc belongs to Case 2 or Case 3 the corresponding equation holds with equality, which shows for all $\xi \in (0,\alpha]$ that

$$\ell_v(\phi+\xi) = \min_{e=uv\in E} T_e(\ell_u(\phi+\xi)).$$

This completes the proof. ◀

A.6 Bound on node labels of thin flows with resetting

▶ **Lemma 17.** *For every thin flow with resetting (x',ℓ') in G we have $\ell'_v \le 1/\sigma$ for all $v \in V$.*

Proof. It holds that $\ell'_{s_i} = x'_i/r_i$ and for $v \in V\backslash S^+$ the ℓ'_v labels are equal to ℓ'_u or x'_e/ν_e for some incoming arc $e = uv$. It follows that all ℓ' labels in the original graph are bounded from above by

$$\max\left\{\left(\max_{i=1,\dots,n} x'_i/r_i\right), \left(\max_{e\in E} x'_e/\nu_e\right)\right\} \le \max\{1/r_{\min}, 1/\nu_{\min}\} = 1/\sigma.$$

Note that all x'_i and x'_e are bounded by 1 from above since the flow value of x' is 1. ◀

Oligopolistic Competitive Packet Routing

Britta Peis
Department of Management Science, RWTH Aachen
Kackertstraße 7, 52072 Aachen, Germany
peis@oms.rwth-aachen.de

Bjoern Tauer
Department of Management Science, RWTH Aachen
Kackertstraße 7, 52072 Aachen, Germany
bjoern.tauer@oms.rwth-aachen.de

Veerle Timmermans
Department of Management Science, RWTH Aachen
Kackertstraße 7, 52072 Aachen, Germany
veerle.timmermans@oms.rwth-aachen.de

Laura Vargas Koch
Department of Management Science, RWTH Aachen
Kackertstraße 7, 52072 Aachen, Germany
laura.vargas@oms.rwth-aachen.de

Abstract

Oligopolistic competitive packet routing games model situations in which traffic is routed in discrete units through a network over time. We study a game-theoretic variant of packet routing, where in contrast to classical packet routing, we are lacking a central authority to decide on an oblivious routing protocol. Instead, selfish acting decision makers ("players") control a certain amount of traffic each, which needs to be sent as fast as possible from a player-specific origin to a player-specific destination through a commonly used network. The network is represented by a directed graph, each edge of which being endowed with a transit time, as well as a capacity bounding the number of traffic units entering an edge simultaneously. Additionally, a priority policy on the set of players is publicly known with respect to which conflicts at intersections are resolved. We prove the existence of a pure Nash equilibrium and show that it can be constructed by sequentially computing an integral earliest arrival flow for each player. Moreover, we derive several tight bounds on the price of anarchy and the price of stability in single source games.

2012 ACM Subject Classification Networks → Network algorithms

Keywords and phrases Competitive Packet Routing, Nash Equilibrium, Oligopoly, Efficiency of Equilibria, Priority Policy

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.13

1 Introduction

One of the fundamental problems in parallel and distributed systems lies in the transport of discrete traffic units ("packets") through a network over time. From a centralized optimization perspective, the design of routing protocols requires two kinds of decisions: first, an origin-destination path needs to be selected for each of the packets, and second, priority policies need to be defined to resolve conflicts whenever more packets than the link-capacity allows are going to traverse the same link simultaneously.

We model the network by a directed graph $G = (V, E)$ whose edges correspond to the links of the network. Each edge $e \in E$ is equipped with a certain bandwidth $u_e > 0$ denoting the maximal number of packets that are allowed to enter edge e simultaneously, and a certain transit time $\tau_e \geq 0$ denoting the time needed for a single packet to traverse e. Each packet must be sent through the network from its origin $s_i \in V$ to its destination $t_i \in V$ along a single path P_i selected from the collection $\mathcal{P}_i \subseteq 2^{|E|}$ of all simple s_i-t_i-paths. We assume that time is discrete and that all packets take their steps simultaneously. Thus, it suffices to consider integral capacities and travel times. The corresponding packet routing problem is to minimize the makespan of such a routing protocol, which is the latest point in time when a packet reaches its destination vertex. This problem is also known under the name *quickest integral multi-commodity flow over time* (see e.g. [4]).

When considering packet routing problems, like routing traffic in a road network, it is natural to view these problems from a game-theoretical perspective. In particular, as it might well be the case that there is no central authority which predescribes a routing protocol. Instead, packets are routed through the network by selfish acting decision makers ("players") each of which aiming at sending the packets under her control as fast as possible from the player-specific origin to the player-specific destination. Such situations can be modeled by *competitive packet routing games*, a special class of non-cooperative strategic games. In a competitive packet routing game, the network and the forwarding policy are publicly known. Each of the players $i \in N = \{1, \ldots, n\}$ decides on the routes along which the k_i packets under her control are to be routed from origin s_i to destination t_i.

Packet routing games usually restrict to the setting where each player controls exactly one packet. In this paper, we consider the more general setting where each player $i \in N$ controls an arbitrary integral amount of k_i packets which all need to be routed along paths in $\mathcal{P}_i$. We call these games *oligopolistic competitive packet routing games* to distinguish between our model and the model of competitive packet routing games investigated in [6]. The individual goal for each player is to minimize the average arrival time of the packets under her control, which corresponds to the computation of an earliest arrival schedule [9]. As a forwarding policy, we assume in our model that a global priority list π on the players is given according to which conflicts at intersections are resolved. That is, when more packets seek to enter an edge than the capacity allows for, packets belonging to players higher on the priority list go first. In these games, we study the drawbacks of the absence of a central authority, and the benefits of coordination between players. This analysis is motivated by future road traffic scenarios where instead of individual cars, private companies own fleets with a large number of autonomous vehicles. Similar to the development of the commercialization of the internet (and the possible abolition of net neutrality), one can think of a system where higher paying fleet owners gain benefits (priority) over non-paying fleet owners. As a city you are interested in the performance of such a prioritized system.

Contributions

A strategy where no player can unilaterally deviate to decrease her cost is called a *pure Nash equilibrium* (*equilibrium*, for short). In Section 3, we show that an equilibrium exists and that it can be constructed within pseudo-polynomial time by sequentially computing an earliest arrival flow for each player. In Section 4 and Section 5, we measure the efficiency of equilibria by comparing the best and worst total cost under an equilibrium state with the minimal total cost achievable by a central authority. The corresponding ratios are usually referred to as *Price of Stability (PoS)* [cf. [1]] and the *Price of Anarchy (PoA)* [cf. [11]], respectively.

In Section 4 we consider games in which all players share a common source and a common sink ("single commodity games"). We prove that the PoS in single commodity games is equal to 1, while the PoA is bounded from above by n. To show the tightness of the PoA, we provide an example in which the PoA converges to n with increasing number of packets. For the case where all players have identical demands, i.e., where $k_i = k_j$ for all $i, j \in N$, we prove that the PoA is bounded from above by $\frac{1}{2}(n+1)$ and give a matching lower bound example. Note that these bounds depend on the number of players, but are independent of the number of packets to be routed through the network. Thus, even for a very high number of packets we get a low price of anarchy if the number of players is small.

Lastly, in Section 5, we study games in which all players share a common source s, but might have player-specific sinks t_i ("single source games"). For single source games, we give an example where the PoS grows to 2 with increasing number of packets. The PoA might also be larger than for single commodity games. We even give an algorithm that computes, given the demands of all players, an example maximizing the PoA for the given set of demands.

Related Work

Packet routing has received a vast amount of attention in the past decades. A break-through result is due to Leighton, Maggs and Rao [13], who prove the existence of a routing protocol for fixed paths, whose makespan is a constant-factor approximation in terms of the natural lower bound $(C + D)/2$. Here, C denotes the *congestion*, i.e., the maximum number of packets traversing the same edge, and D denotes the *dilation*, i.e., the length of the longest path along which a packet is routed. This result has been improved and simplified several times in the past (see, e.g., [19, 16, 7, 17]). For the more general problem where paths are not fixed, Srinivasan and Teo [21] show that a constant factor approximation is still possible. To prove this result they use the fact that it is sufficient to find paths which minimize the sum of congestion and dilation. Koch et al. [10] extend this result to a more general setting, where messages that consist of several packets need to be routed through a network. In contrast to our model, they require that all packets of a message wait at the head of each traversed link until the last packet of the message arrived.

A game-theoretic perspective on packet routing can be found in the pioneering work of Hoefer et al. [8]. Here, they start with network congestion games (see Roughgarden [18] for an introduction) and generalize this model to a variant over time. More details on this development can be found in [8]. Similar to our competitive packet routing model, the model in [8] considers players $i \in N$, and each player is associated with an origin vertex s_i, a destination vertex t_i, and a player-specific weight w_i. However, in contrast to our model, the capacity on each link does not bound the number of packets allowed to traverse the link simultaneously at each integral time step. Rather, it bounds the total load on an edge induced by packets traversing this edge at each point in time. The authors analyze four different forwarding policies (FIFO, equal time sharing, (non-) preemptive global ranking), they focus on the existence of Nash equilibria and the convergence of best responses. Kulkarni et al. [12] extend the model of Höfer et al. and bound the price of anarchy, using LP duality. Lastly, Harks et al. [6] investigates the special class of competitive packet routing in which each player controls exactly one packet. They study existence, efficiency, and computability of equilibria with respect to both local (i.e. edge-dependent) and global priority lists on the players. For both forwarding policies, they analyze the existence of equilibria and establish bounds on the price of anarchy and the price of stability using the techniques introduced by Kulkarni et al [12]. A more detailed comparison can be found in the respective chapters.

2 Preliminaries

The Model

A *multi commodity oligopolistic competitive packet routing game* $\mathcal{G}$ is represented by the tuple: $\mathcal{G} := (G, N, (s_i, t_i, k_i)_{i \in N}, \pi)$, where $G := (V, E, (\tau_e)_{e \in E}, (u_e)_{e \in E})$ is a directed graph that consists of a set of nodes V and edges E, where each edge $e \in E$ is endowed with an integral transit time $\tau_e \geq 0$ and an integral capacity $u_e > 0$. The transit time of an edge denotes the time it takes for each player to traverse this edge. The capacity is a limit on the number of packets that can enter an edge at each integral time step. We use N to denote the set of players, where each player $i \in N$ has a player-specific source and sink $s_i, t_i \in V$. Additionally, each player has a set of k_i identical packets she desires to send from s_i to t_i. We denote this set by K_i. Lastly, as a forwarding policy, we are given a priority list $\pi \in \Pi_n$, where Π_n is the set of all different orderings on n players. Whenever more packets desire to enter an edge than the capacity allows for, packets of players higher in the priority list can go first. Without loss of generality, we assume that players are numbered according to their position in the priority list π.

A feasible strategy x_i of a player $i \in N$ determines for every packet in K_i a simple s_i-t_i-path, together with a release time, i.e., the time the packet should start trying to traverse its assigned path. That is, player i decides on a path vector $P_i \in \mathcal{P}_i^{k_i}$, where $\mathcal{P}_i$ denotes the set of all simple s_i-t_i-paths. Additionally, player i decides on a release time for every packet by selecting a release-time vector $R_i \in \mathbb{N}_{\geq 0}^{k_i}$. Thus, the set of feasible strategies of player i can be described as $\mathcal{S}_i(k_i) := \left\{ x_i = (P_i, R_i) \mid P_i \in \mathcal{P}_i^{k_i}, R_i \in \mathbb{N}_{\geq 0}^{k_i} \right\}$.

The combined strategy space is denoted by $\mathcal{S} := \prod_{i \in N} \mathcal{S}_i(k_i)$ and additionally we denote by $x := (x_i)_{i \in N}$ the overall strategy profile. In a strategy profile x, each packet $\ell \in K_i$ travels over its assigned path to its destination. We let $C_{i,\ell}(x)$ denote its arrival time at sink t_i. The goal of each player is to minimize the sum of the arrival times of her packets $C_i(x) := \sum_{\ell \in K_i} C_{i,\ell}(x)$. The *social cost* of strategy profile $x \in \mathcal{S}$ is $C(x) = \sum_{i \in N} C_i(x)$, i.e., the total cost of all players. A profile $x \in \mathcal{S}$ minimizing the social cost is called *social optimum.*

Note that the arrival time of each packet is uniquely determined by embedding the strategies of the players in graph G. We embed the players one by one in order of their priority list and for every player we embed the packets in order of the strategy vector (assuming a decreasing priority) starting at their respective release time. In our model, packets are not allowed to wait at any intermediate node unless necessary. Thus, such an embedding is unique.

As usual in game theory, for every $i \in N$, we write $\mathcal{S}_{-i}(k_{-i}) := \prod_{j \neq i} \mathcal{S}_j(k_j)$ and $x = (x_i, x_{-i})$ meaning that $x_i \in \mathcal{S}_i(k_i)$ and $x_{-i} \in \mathcal{S}_{-i}(k_{-i})$. A strategy profile x is called a *Nash equilibrium* whenever no player can unilaterally deviate and decrease her own cost, i.e. $C_i(x_i, x_{-i}) \leq C_i(y_i, x_{-i})$ for all $y_i \in \mathcal{S}_i(k_i)$. A pair $(x, (y_i, x_{-i}))$ is called an *improving move* when $C_i(y_i, x_{-i}) < C_i(x)$. A strategy x_i of player i is called a *best response* to x_{-i} whenever $x_i \in \arg\min_{y_i \in \mathcal{S}_i(k_i)} \{C_i(y_i, x_{-i})\}$. Thus, a profile x is a Nash equilibrium if and only if there is no player that has an improving move, or equivalently, if each player plays a best response.

▶ **Example 1.** Consider the single commodity game $\mathcal{G}$ on directed graph G depicted in Figure 1, where the transit times are depicted in the picture, and the capacity of each edge is equal to one. We consider two players, each controlling exactly one packet that needs to be routed from the common source s to the common sink t. As stated before, we assume the

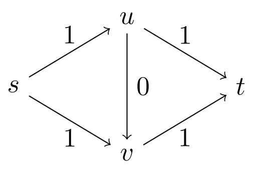


Figure 1 Graph G.

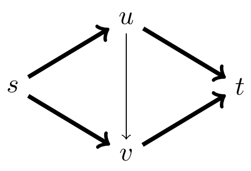


Figure 2 Social optimum.

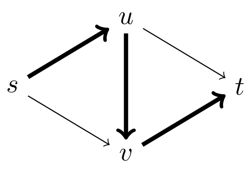


Figure 3 Strategy in a NE.

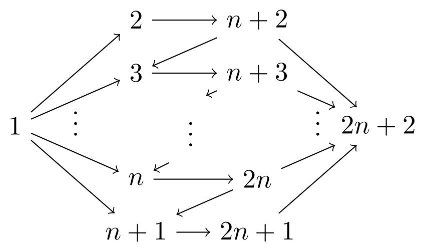


Figure 4 Graph $BG(n)$.

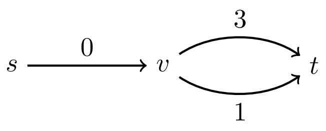


Figure 5 Release times as part of the strategy.

players to be numbered according to their spot in the priority list, hence, player 1 has priority over player 2. Note that the first player has three optimal strategies: she selects release time zero and either uses one of the parallel paths (Figure 2) or the path that intersects with both of these paths (Figure 3). If she chooses one of the parallel paths, the second player can take the other parallel path, resulting in a socially optimal equilibrium $x \in \mathcal{S}$ with $C_1(x) = C_2(x) = 2$. If she uses the the zig-zag path depicted in Figure 3 instead, she harms player 2 who cannot arrive at t before time step 3. If, for example, the second player selects release time zero, and travels along either path, we result in an equilibrium x' with social cost $C_1(x') + C_2(x') = 2 + 3 = 5$. Thus, $PoS = 1$ and $PoA \geq \frac{5}{4}$.

The graph G depicted in Figure 1 is a well-known graph, famous from the Braess-paradox, and is used several times to prove lower bounds on the price of anarchy, e.g. [12]. In the rest of this paper we use this graph, and an extension of it several times. Therefore, we define $BG(n)$ as a graph on $2n+2$ vertices with four types of edges $E_{BG(n)} = E_1(n) \cup E_2(n) \cup E_3(n) \cup E_4(n)$, where: $E_1(n) := \{(1, v) \mid v \in \{2, \dots, n+1\}$, $E_2(n) := \{(v, v+n) \mid v \in \{2, \dots, n+1\}$, $E_3(n) := \{(v, v-n+1) \mid v \in \{n+2, \dots, 2n\}$, $E_4(n) := \{(v, 2n+2) \mid v \in \{n+2, \dots, 2n+1\}$. Note that graph $BG(n)$ has n parallel paths from node 1 to $2n+2$, and one path that intersects all n parallel paths. A visualisation of graph $BG(n)$ can be found in Figure 4.

As stated in the model, players do not only choose a path in the network, but also a release time for each packet, i.e., the time at which a packet starts traversing its assigned path. This brings no advantage regarding the cost function of a player. Though, by allowing players to set a release time for each packet, friendly players have the option not to congest the network unnecessarily. Moreover, players might prefer to wait at the source instead of waiting at intermediate nodes. As is proven in Section 3, it also allows us to compute social optima in all single commodity games. We give an example that illustrates the use of setting release times for packets: Consider the graph depicted in Figure 5, with four players owning one packet each. The first edge has capacity two and the other edges have capacity one. The transit times of the edges are depicted in the network. We denote the path taking the lower edge (v, t) as p_1 and the path taking the upper edge as p_2. A possible equilibrium is $x_1 = (p_1, 0)$, $x_2 = (p_1, 1)$, $x_3 = (p_1, 2)$ and $x_4 = (p_2, 0)$ realizing arrival times $C_1 = 1$, $C_2 = 2$, $C_3 = 3$ and $C_4 = 3$. If players cannot choose release times, there is a unique equilibrium in which all players choose p_1, realizing arrival times $C_1 = 1$, $C_2 = 2$, $C_3 = 3$ and $C_4 = 4$.

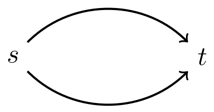


Figure 6 No PNE without global priority list.

Table 1 Three possible equilibria.

	Strategy 1	Strategy 2	Strategy 3
$(1,1)$	(top,0)	(top,0)	(top,0)
$(1,2)$	(top,0)	(bottom,0)	(bottom,0)
$(2,1)$	(bottom,0)	(top,0)	(bottom,0)
$(2,2)$	(bottom,0)	(bottom,0)	(top,0)

In Section 3 we prove that, whenever we are given a priority list on the players, Nash equilibria exist. In contrast, if the priority list is given on the set of packets instead of players, the existence of equilibria cannot be guaranteed.

▶ **Example 2.** Consider an oligopolistic packet routing game on the graph depicted in Figure 6. This graph has two edges: *top* and *bottom*, with both capacity and transit time equal to one. We assume there are two players that both want to route two packets from source s to sink t. Let (i, ℓ) denote packet ℓ of player i. Assume that the priority over the packets is $\pi = ((1,1),(2,1),(1,2),(2,2))$. Note that in this network no player can decrease her costs by increasing the release time from zero. Further, note that in any equilibrium each edge is used by exactly two packets. This implies that without loss of generality there are three candidates for an equilibrium, which are depicted in Table 1.

Strategy 1 is not an equilibrium, as player 1 is better of by switching packet one to the other edge. Strategy 2 is not an equilibrium, as player 2 is better of by interchanging packet one and two. Strategy 3 is also not an equilibrium, as player 1 would be better of by switching packet one and two around. Hence, this game does not have a Nash equilibrium.

Due to the simplicity of the example, it is sensible to restrict our research to priority lists on players instead of packets.

Flows over time and earliest arrival flows

In an oligopolistic packet routing game, a player sends a set of k_i packets from a source s_i to a sink t_i. Thus, every feasible strategy is an integral s_i-t_i-flow over time of flow value k_i. We shortly introduce flows over time, also known under the name *dynamic flows.* For a more detailed introduction on static and dynamic flows, we refer to Skutella [20].

▶ **Definition 3.** Given a graph $G = (V, E, (\tau_e)_{e\in E}, (u_e)_{e\in E})$ with transit times and capacities, an *integral s-t-flow over time* is a set of functions $f_e : \mathbb{N}_{\geq 0} \to \mathbb{N}_{\geq 0}$ for all $e \in E$ satisfying the following two constraints:

$$f_e(\theta) \leq u_e \qquad \forall e \in E, \theta \in \mathbb{N}_{\geq 0}, \tag{1}$$

$$\sum_{e\in\delta^-(v)} \sum_{\theta=0}^{\xi-\tau_e} f_e(\theta) \geq \sum_{e\in\delta^+(v)} \sum_{\theta=0}^{\xi} f_e(\theta) \qquad \forall \xi \in \mathbb{Z}_{\geq 0}, v \in V \setminus \{s,t\}. \tag{2}$$

Here $\delta^+(v) := \{(v,u) \in E \mid u \in V\}$ and $\delta^-(v) := \{(u,v) \in E \mid u \in V\}$. The first inequality imposes the capacity constraint of the edges on the flow, and the second constraint represents the flow conservation property. If Equation (2) is fulfilled with equality, we say that strong flow conservation holds, implying that there is no waiting at intermediate nodes.

A special variant of flows over time are *earliest arrival flows.* Such an earliest arrival flow (EAF) maximizes the flow value arriving at the sink at each integral time step $\theta \in \mathbb{N}_{\geq 0}$. To be more precise, we define $A(f,T)$ to be the amount of flow that arrives at t on or before time T, e.g. $A(f,T) := \sum_{\theta=0}^{T} a(f,\theta)$, where $a(f,\theta)$ denotes the amount of flow arriving at the sink at time θ. We say that a feasible integral s-t-flow over time f fulfills the *earliest arrival property* whenever $A(f,\theta) \geq A(f',\theta)$ for all feasible integral s-t-flows f' and all $\theta \in \mathbb{N}_{\geq 0}$. An integral s-t-flow over time that satisfies strong flow conservation and fulfills the earliest arrival property is called an *integral earliest arrival s-t-flow* (s-t-EAF). Integral earliest arrival flows are guaranteed to exist in a single commodity network [5]. In such networks, an earliest arrival flow can be computed by Wilkinson's algorithm [23] when the capacities do not vary over time, and Tjandra's algorithm when capacities do vary over time [22]. In a multiple source, single sink setting, earliest arrival flows also exist, and can be computed when capacities do not change over time [14, 15]. In a multi commodity setting there are networks such that no earliest arrival flow exists [3].

3 Existence of Nash equilibria

Whenever each player has exactly one packet, Harks et al. [6] show that a pure Nash equilibrium exists and can be found using a sequence of shortest path computations. We prove the existence of pure Nash equilibria in multi commodity oligopolistic competitive packet routing games by exploiting the connection to earliest arrival flows. We start by showing how to compute a best response for player i by computing an s_i-t_i-EAF in a network with time-varying capacities. Afterwards, we prove that a pure Nash equilibrium can be obtained by sequentially computing such an earliest arrival flow for each player, in order of the priority list. Lastly, we show that in a single commodity game an earliest arrival flow minimizes the social cost function.

▶ **Theorem 4.** *In a multi commodity oligopolistic competitive packet routing game, a best response of a player $i \in N$ corresponds to an s_i-t_i-earliest arrival flow with time-varying capacities, and vice versa.*

Proof. Fix a player $i \in N$ and strategies $x_1, \dots, x_{i-1}$ of players higher in the priority list, arbitrarily. As mentioned before, we assume that players are ordered according to the priority list. Thus, players $j \in \{i+1, \dots, n\}$ cannot influence the travel time of packets controlled by player i. A best response of player i towards x_{-i} is therefore a strategy choice (or flow) x_i minimizing $\sum_{\ell \in K_i} C_{i,\ell}(x)$, i.e., the sum of arrival times of all packets in K_i. Obviously, this corresponds to minimizing the average arrival time $\frac{1}{k_i} \sum_{\ell \in K_i} C_{i,\ell}(x)$. In [9], it was shown that minimizing $\frac{1}{k_i} \sum_{\ell \in K_i} C_{i,\ell}(x)$ is equivalent to maximizing $\sum_{\theta \in \mathbb{N}_{\geq 0}} A_i(x,\theta)$, where $A_i(x,\theta) := |\{\ell \in K_i \mid C_{i,\ell}(x) \leq \theta\}|$ denotes the number of packets of player i arriving at sink t_i before time θ under strategy x. For sake of completeness, we present a proof for this fact in Appendix A.

It is well-known, that every s-t-network admits an s-t-earliest arrival flow even for the case of time-dependent capacities (cf. Tjandra [22]). By definition, an earliest arrival flow (EAF) is a flow maximizing $A_i(x,\theta)$ for every time θ. Thus, such an EAF maximizes the sum $\sum_{\theta \in \mathbb{N}_{\geq 0}} A_i(x,\theta)$ as well, and therefore corresponds to a best response of player i.

On the contrary, there can be no feasible flow other than an earliest arrival flow maximizing this sum, since the earliest arrival flow maximizes every single summand. As a consequence, every best response corresponds to an earliest arrival flow, and vice versa.

We can compute a best response x_i as follows: We embed the strategies $x_1, \dots x_{i-1}$ of players $\{1, \dots, i-1\}$ one by one in the network, in order of the priority list. Using the algorithm of Tjandra [22], we compute an s_i-t_i-EAF f in the resulting network with varying capacities. We decompose flow f in k_i paths $(p_\ell)_{\ell \in K_i}$, where w.l.o.g. we assume that the paths are ordered according to non-decreasing path lengths. Each packet $l \in K_i$ is assigned the release time r_ℓ according to its release time in the path decomposition of the earliest arrival flow. To show that the strategy $x_i = (p_\ell, r_\ell)_{\ell \in K_i}$ is a feasible one, we prove that packets never wait at intermediate nodes, unless the capacity of this edge is reduced due to a preceding player, and that all paths are cycle-free. For a proof of the cycle-freeness, we refer to the Appendix B. If all packets start according to their release dates, no packet of player i has to wait for another packet of player i, since f is an earliest arrival flow. Particularly, all packets take the same path and arrive at every intermediate node at the same point in time as their correspondent in the earliest arrival flow. So, the arrival pattern of the flow corresponding to x_i has the earliest arrival property and thus x_i is a best response for player i. The priority rules in the model are obeyed since the players are embedded one by one in order to the priority list. If a packet of a player i needs to wait due to a reduced capacity, this corresponds to a packet of a player $j < i$ using the edge. ◀

Thus, in order to compute a pure Nash equilibrium, we subsequently compute earliest arrival flows for the players in the order of the priority lists according to Theorem 4.

▶ **Corollary 5.** *Each multi commodity oligopolistic competitive packet routing game admits a pure Nash equilibrium. Moreover, a pure Nash equilibrium can be computed by calculating subsequently an earliest arrival flow for each player in the order of the priority list by using the algorithm of Tjandra [22]. The running time is within $O(|E| \cdot |V| \cdot \sum_{i \in N} (S'_i + k_i)^2 \cdot k_i)$, where S'_i is the length of a shortest s_i-t_i-path in the underlying network with capacities adapted according to the best responses of players in $\{1, \dots, i-1\}$.*

In order to achieve a social optimum, we assume there is one central authority who coordinates all packets. Note that this central authority still needs to take the priority rules into account. In single commodity games, we are able to compute a social optimum.

▶ **Theorem 6.** *In single commodity oligopolistic competitive packet routing games, a social optimum can be computed within pseudo-polynomial time.*

Proof. First, we assume that there is a central authority controlling all $K = \sum_{i \in N} k_i$ packets. According to Theorem 4, a strategy minimizing the social cost function for one player corresponds to an earliest arrival flow. As capacities are constant over time, we can compute an earliest arrival flow f by using Wilkinson's algorithm [23]. Note that this algorithm computes K shortest paths, and thus runs in pseudo-polynomial time. It is left to decompose this strategy into player specific strategies and check if the player specific strategies obey the priority rules. In order to do so, we find a path decomposition of flow f with a corresponding release time for each packet: $(p_q, r_q)_{1 \le q \le K}$, where the tuples are numbered according to the time the corresponding packet arrives at the sink. We define $F_i := \sum_{q=1}^{i-1} k_q$ and $x_i = (p_q, r_q)_{F_i < q \le F_i + k_i}$.

By the choice of the release times, we guarantee that a packet following the corresponding successive shortest path can traverse the network without being delayed by other packets. Hence, x realizes the arrival pattern of an earliest arrival flow while obeying the priority rules. The paths are cycle-free due to Appendix B. ◀

4 Efficiency in single commodity games

We discuss the price of stability (PoS) and the price of anarchy (PoA) in single commodity games in this section, and in single source multiple sink games in the subsequent section. First of all, similar as in the model of Harks et al. [6], it can easily be derived from Theorem 6 that the price of stability in single commodity games is equal to one.

▶ **Corollary 7.** *Each single commodity oligopolistic competitive packet routing game admits a socially optimal pure Nash equilibrium which can be computed via one earliest arrival flow computation.*

Proof. We compute a social optimum as described in the proof of Theorem 6, and prove that the resulting strategies form a Nash equilibrium. Observe that strategy x_i is a best response for player i, as she cannot decrease the arrival times of her packets due to the earliest arrival property of the total flow. Thus, strategy $(x_i)_{i \in N}$ is an equilibrium minimizing the social cost. ◀

We show that in the single commodity setting, the price of anarchy is bounded by n. Furthermore, we introduce an example such that, when the number of packets grows large, the price of anarchy in our example converges to n. We start by proving an upper bound on the price of anarchy. The proof is based on the following insight.

▶ **Lemma 8.** *Let NE be a Nash equilibrium for game $\mathcal{G}$ and let OPT be a socially optimal strategy profile constructed as described in the proof of Theorem 6. Then, for every player i and every packet $\ell \in K_i$, it holds that: $C_{i,\ell}(NE) \leq i \cdot C_{i,\ell}(OPT)$.*

A proof of Lemma 8 can be found in the Appendix C. Using Lemma 8 we prove an upper bound on the price of anarchy in single commodity oligopolistic competitive packet routing games.

▶ **Theorem 9.** *In single commodity oligopolistic competitive packet routing games, the price of anarchy is bounded from above by n.*

Proof. Let NE be the Nash equilibrium for single commodity game $\mathcal{G}$ that maximizes the social cost, and let OPT be a strategy profile that minimizes the social cost function. We prove that $\frac{C(NE)}{C(OPT)} \leq n$. We use Lemma 8 to obtain:

$$C(NE) \leq \sum_{i=1}^{n} \sum_{\ell \in K_i} i \cdot C_{i,\ell}(OPT) \leq n \cdot \sum_{i=1}^{n} \sum_{\ell \in K_i} C_{i,\ell}(OPT) = n \cdot C(OPT).$$

Thus, the price of anarchy has an upper bound of n. ◀

In Theorem 10 we state an example of a single commodity game where, if the total number of packets in the game grows large, the price of anarchy converges to n.

▶ **Example 10.** Consider a game with n players, where the first $n-1$ players have only one packet, and player n has k_n packets. All players need to route their packets from s to t in the Braess graph $BG(n + k_n - 1)$ depicted in Figure 7.

In an optimal solution, all packets traverse the $k_n + n - 1$ available parallel paths as depicted in Figure 8, incurring a social cost of $k_n + n - 1$. Note that it is also a viable option for the first $n - 1$ players to traverse the path as depicted in Figure 9. The k_n packets of

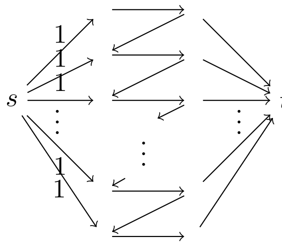


Figure 7 Graph BG.

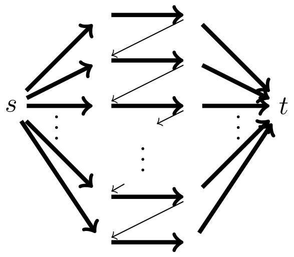


Figure 8 OPT.

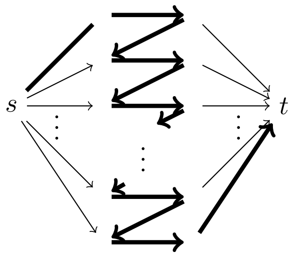


Figure 9 NE.

player n cannot arrive before time n, incurring a social cost of $\frac{1}{2}n(n-1)+nk_n$. Hence, in this example the price of anarchy is:

$$PoA = \frac{\frac{1}{2}n(n-1)+nk_n}{k_n+n-1} = n - \frac{\frac{1}{2}n(n-1)}{k_n+n-1}.$$

Note that when k_n grows large, this ratio converges to n. Also observe that this result generalizes the bound in [6], where all players own only one packet each ($k = 1$).

Hence, the bound we prove in Theorem 9 is tight. In this example, we exploit the fact that the last player has far more packets than the others. Hence, it is reasonable to consider the special case that all players have the same number of packets k, i.e. $k_i = k_j$ for all $i, j \in N$. We denote such a game as a symmetric game, since the strategy spaces of all the players are identical. In a symmetric game the price of anarchy decreases to $\frac{1}{2}(n+1)$. This is an extension of the result of Harks et al. [6], which would give a price of anarchy of $\frac{1}{2}(kn+1)$ and coincide for $k = 1$. In order to prove this statement, we first show that $\frac{1}{2}(n+1)$ is an upper bound on the price of anarchy.

▶ **Theorem 11.** *In symmetric oligopolistic competitive packet routing games the price of anarchy is bounded from above by $\frac{1}{2}(n+1)$.*

Proof. Let S be the length of the shortest s-t-path in the network. Assume that in an optimal strategy the first player has a_p packets arriving at time $S+p-1$, where p ranges from one up to some $q_1 \in \mathbb{N}_{>0}$, where $q_1 = \arg\min_{p\in\mathbb{N}_{>0}}\{a_{q'} = 0, \forall q' > p\}$. Thus $\sum_{p=1}^{q_1} a_p = k$, and within a time span of q_1 all packets of player 1 reach the sink. We say that q_1 is the *arrival spread* of player 1. Note that $1 \le a_1 \le a_2 \le \cdots \le a_{q_1-1}$, as at least one packet of player 1 can arrive at time S by taking the shortest path. Further, if a_p packets arrive at time $S+p-1$ at least so many packets can arrive at $S+p$ by choosing the same paths as the packets arriving at $S+p-1$ as long as there are enough packets left to fill up all paths.

As arrival times are increasing, the arrival times for all packets of the remaining $n-1$ players are at least $S+q_1-1$. Hence, we can find the following lower bound on the social cost: $C(OPT) \ge knS + \sum_{p=1}^{q_1} a_p(p-1) + (q_1-1)k(n-1)$.

In order to find an upper bound on the worst Nash equilibrium, we give an upper bound on the arrival times of the i'th player, in terms of the arrival times of the first player. We show that player i can always copy the strategy of player 1, but increase the release times by $(i-1)q_1$ time units. In general, we prove the following statement: if the first player has a_p packets arriving at $S+p-1$ as described above, then the i'th player can play the same strategy $(i-1)q_1$ time units later, with a_p packets arriving at $S+p-1+(i-1)q_1$.

We prove this statement by induction. In order to do so, we use the even stronger statement which says: player i can copy the strategy of the first player $(i-1)q_1$ time units later, without being delayed by any other player. Assume the induction hypothesis holds for the first $i-1$ players. Then we show that the i'th player can play the strategy of the first player, where the release times are increased by $(i-1)q_1$. Note that the arrival times of the first $i-1$ players are all strictly smaller than $S+(i-1)q_1$. On the contrary, we assume that there exists a packet ℓ of player i that needs to wait for a packet ℓ' by a previous player. This implies that, if packet ℓ' would not block packet ℓ, then packet ℓ could arrive on the original arrival time of packet ℓ', which is smaller than $S+(i-1)q_1$. As packet ℓ can only depart from s at release time $(i-1)q_1$, this would imply that the shortest s-t-path has a length smaller than S, which contradicts the fact that S is the length of shortest path in the network. Hence, player i can repeat the strategy of the first player $(i-1)q_1$ time units later without being delayed and thus with a_p packets arriving at $S+p-1+(i-1)q_1$. Furthermore, we know by Theorem 4 that a best response is equivalent to an earliest arrival flow. This guarantees that no packet of player i arrives later than $S+q_1-1+(i-1)q_1$. This gives us an upper bound on the total cost of any Nash equilibrium: $C(NE) \leq \sum_{i=1}^{n} \sum_{p=1}^{q_1} a_p(S+p-1+(i-1)q_1)$.

As we have a lower bound on the social cost of an optimal solution, and an upper bound of the cost in any Nash equilibrium, we can find an upper bound on the price of anarchy.

$$PoA \quad \leq \quad \frac{\sum_{i=1}^{n} \sum_{p=1}^{q_1} a_p(S+p-1+(i-1)q_1)}{knS + \sum_{p=1}^{q_1} a_p(p-1) + (q_1-1)k(n-1)}.$$

This fraction is maximized whenever $S=1$ and $q_1=1$, therefore the price of anarchy is bounded from above by $\frac{1}{2}(n+1)$. The technical argument for this claim can be found in the Appendix D. ◀

To observe that this result is tight, consider Braess graph $BG(n)$ (see Figure 7), where each edge has a capacity k. Assume that the edges leaving s have cost S. Since there are n disjoint paths with capacity k, all packets of all players reach the sink simultaneously at time S in a social optimal profile OPT (see Figure 8), resulting in a social cost $C(OPT) = nkS$. However, there is a profile in which all packets of each player take the path depicted in Figure 9, which turns out to be a Nash equilibrium NE. Here, the arrival time is $C_{i,\ell}(NE) = S+i-1$ for all players $i \in N$ and for all packets $\ell \in K_i$. Therefore $C(NE) = nk(S-1) + k\sum_{i=1}^{n} i$. If we choose $S=1$ we get the tight upper bound $PoA = \frac{1}{2}(n+1)$. Observe that this result generalizes the bound in [6], where all players own only one packet each ($k=1$).

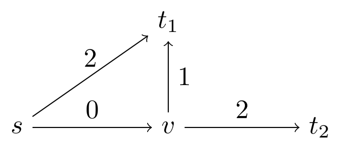
5 Efficiency in single source games

In this section, we consider games where players have a common source s, but player specific sinks t_i, $i \in N$. In general, earliest arrival flows do not necessarily exist in multi-commodity games, even if all commodities share a common source (see, e.g., [2]).

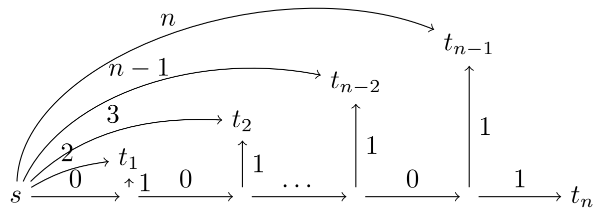
▶ **Example 12.** Consider the graph depicted in Figure 10 with unit capacities and travel times as shown in the picture. Assume one traffic unit needs to be send from s to t_1, and one unit from s to t_2. In order to maximize the amount of flow arriving after two units of time, we send one unit along edge (s,t_1) and one unit along (s,v,t_2) so that both units reach the respective sink after two time units. This flow does obviously not maximize the amount of flow reaching sink t_1 at time step $\theta = 1$. Thus, in this graph, no earliest arrival flow exists.

We extend this example to a single source competitive packet routing game on n players, and show that, in contrast to single commodity games, single source games do not necessarily admit socially optimal pure Nash equilibria.

▶ **Theorem 13.** *In oligopolistic competitive packet routing games with a global source s and player specific sinks $t_1, \dots, t_n$, the price of stability is bounded from below by* 2.



■ **Figure 10** Network without an EAF.



■ **Figure 11** Network with a PoS converging to 2.

Proof. We consider the graph depicted in Figure 11, where the capacity of each edge is equal to one. We assume there are n players, where the first $n-1$ players control one packet, and the last player controls n packets. Note that each of the first $n-1$ players has two feasible strategies. Either she takes her direct s-t_i-route, or the path using the zero-length edges. The last player has only one feasible strategy.

In the optimal solution OPT, the first $n-1$ players all take the direct s-t_i-route, incurring a total cost of $n(n+1)-1$ for all players. In the unique Nash equilibrium NE, all players use their indirect route, incurring a total cost of $n(2n-1)$. Hence, the price of stability is:

$$PoS = \frac{C(NE)}{C(OPT)} = \frac{n(2n-1)}{n(n+1)-1} \geq \frac{2n-1}{n+1} = 2 - \frac{3}{n+1}.$$

Note that the last term converges to 2 when the number of players grows to infinity. ◀

In the remaining part of this section we focus on the price of anarchy. We show that, in contrast to single commodity games, the bound for the setting with equal demands coincides with the general bound. The tight bound turns out to be a fraction that depends on the number of packets k_i of player i and the number of players n. We present an algorithm that constructs a matching lower bound example for every given n and $(k_i)_{i\in N}$.

Similar as in the previous section, we define S_i to be the length of a shortest s-t_i-path in (G,τ), i.e., $S_i := \min_{P\in\mathcal{P}_i}\sum_{e\in P}\tau_e$. For each player $i\in N$, let OPT_i be an optimal strategy under the assumption that no other player exists, i.e., OPT_i is an integral earliest arrival flow with source s and sink t_i. Clearly, under flow OPT_i, at least one packet reaches the sink at time S_i. We are interested in the *arrival spread* q_i of flow OPT_i which is the length of the time interval in which packets are arriving at the sink under flow OPT_i. Here, q_1 corresponds to the arrival spread q_1 we defined in the proof of Theorem 11. To be more precise, we let $M_i(OPT_i)$ denote the makespan of player i in OPT_i, i.e., the latest point in time when a packet of player i reaches the sink. Then, we define $q_i := M_i(OPT_i) - S_i + 1$ to be the *arrival spread* of flow OPT_i. Let $C_{i,l}(OPT_i)$ denote the arrival time of the packet $l\in K_i$ under flow OPT_i.

▶ **Theorem 14.** *Let NE be an arbitrary pure Nash equilibrium in a single source multiple sink oligopolistic competitive packet routing game $\mathcal{G}$. Then, if $C_{i,\ell}(NE)$ denotes the arrival time of packet $l\in K_i$ under NE,*

$$C_{i,\ell}(NE) \leq C_{i,\ell}(OPT_i) + \sum_{k=1}^{i-1} q_k,$$

Proof. Recall that, under an equilibrium, each player $i\in N$ plays a best response towards the strategy choices of the players $j\in\{1,\dots,i-1\}$ higher in the priority list. We prove this

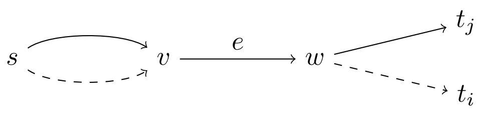


Figure 12 Two player with origin s interact at a common edge e in the network.

theorem by induction. For the first player, the statement trivially holds, since a best response corresponds to an s-t_1-EAF, so $C_{1,l}(NE) = C_{1,l}(OPT_1)$ for each packet $l \in K_1$ controlled by the first player. Assume that $C_{j,\ell}(NE) \leq C_{l,\ell}(OPT_j) + \sum_{k=1}^{j-1} q_k$ holds for each packet $l \in K_j$ controlled by a player $j \in \{1, \dots, i-1\}$. To show that $C_{i,\ell}(NE) \leq C_{i,\ell}(OPT_i) + \sum_{k=1}^{i-1} q_k$ is true, it suffices to convince ourselves that player i could release all of her packets at time $\sum_{k=1}^{i-1} q_k$ and follow the flow pattern of OPT_i without ever being delayed by a packet of players higher in the priority list.

For the sake of contradiction, we assume that a packet ℓ_i of player i has to wait for a packet ℓ_j of player $j < i$. If this is the case, there must exist an edge $e = (v, w)$ that is traversed by both packets ℓ_i and ℓ_j (see Figure 12). Hence, packet ℓ_j could have started at time $\sum_{k=1}^{i-1} q_k$ and arrive at node v at the same time as before, by taking the same s-v-path as packet ℓ_i. By the induction hypothesis, the original arrival time of packet ℓ_j is smaller or equal to $S_j - 1 + \sum_{k=1}^{i-1} q_k$. Note that if packet ℓ_j takes the same s-v-path as packet ℓ_i, and after that continues with its original v-t_j route, it leaves s after time $\sum_{k=1}^{i-1} q_k$ and arrives at t_j before time $\left(S_j - 1 + \sum_{k=1}^{i-1} q_k\right)$. Thus, the time that packet ℓ_j is in the network is bounded from above by:

$$\left(S_j - 1 + \sum_{k=1}^{i-1} q_k\right) - \sum_{k=1}^{i-1} q_k = S_j - 1.$$

This contradicts the fact that S_j is the length of a shortest s-t_j-path. Thus, all packets of player i can leave s at time $\sum_{j=1}^{i-1} q_j$, and arrive at t_i using their optimal strategy, without being delayed by previous players. Therefore:

$$C_{i,\ell}(NE) \leq C_{i,\ell}(OPT_i) + \sum_{k=1}^{i-1} q_k.$$

Further, no packet of player i arrives later than $S_i - 1 + \sum_{j=1}^{i-1} q_j$ since any best response of a player is an earliest arrival flow by Theorem 4. Thus, in no best response a packet falls behind a realizable time. ◀

▶ **Corollary 15.** *For a single source competitive packet routing game $\mathcal{G}$ with n players, demands $(k_i)_{i \in N}$ and arrival spreads $(q_i)_{i \in N}$ of the associated earliest arrival flows OPT_i for each $i \in N$, we have*

$$PoA(\mathcal{G}) \leq 1 + \frac{\sum_{i \in N} \sum_{j=i+1}^{n} q_i k_j}{\sum_{i \in N} C_i(OPT_i)}.$$

Proof. Assume that strategy OPT is a strategy that minimizes the social cost function. Using Theorem 14 we obtain that for any Nash equilibrium NE, we have that:

$$\frac{C(NE)}{C(OPT)} \leq \frac{\sum_{i \in N} C_i(OPT_i) + \sum_{i \in N} \sum_{j=i+1}^{n} q_i k_j}{\sum_{i \in N} C_i(OPT_i)} = 1 + \frac{\sum_{i \in N} \sum_{j=i+1}^{n} q_i k_j}{\sum_{i \in N} C_i(OPT_i)}.$$ ◀

We prove that this bound is actually tight.

▶ **Theorem 16.** *Let N be a set of n players and let $(q_i)_{i\in N}$ and $(k_i)_{i\in N}$ be arbitrary, but fixed, sequences of non-negative integers such that $q_i \le k_i$ for all $i \in N$. Then, there exists a single source competitive packet routing game $\tilde{\mathcal{G}}$ on n players with*

$$PoA(\tilde{G}) = 1 + \frac{\sum_{i\in N}\sum_{j=i+1}^{n} q_i k_j}{\sum_{i\in N} C_i(OPT_i)} = 1 + \frac{\sum_{i\in N}\sum_{j=i+1}^{n} q_i k_j}{\sum_{i\in N}\sum_{j=1}^{q_i} a_{i,j}(S_i + j - 1)}.$$

The proof of this theorem can be found in Appendix E. In the rest of this paper we create an algorithm that, for any set of players N with demands $(k_i)_{i\in N}$, can find arrival patterns for each player that maximizes the price of anarchy. First note that our goal is to maximize q_i while minimizing $C_i(OPT_i) = \sum_{j=1}^{q_i} a_{i,j}(S_i + j - 1)$.

▶ **Lemma 17.** *For any player with k_i packets and arrival spread of $q_i \le k_i$, her cost $C_i(OPT_i) = \sum_{j=1}^{q_i} a_{i,j}(S_i + j - 1)$ is minimized by $Q_i(S_i, q_i)$, where $Q_i(S_i, 1) := k_i S_i$ and*

$$Q_i(S_i, q_i) := k_i(S_i - 1) + \left\lfloor \frac{k_i - 1}{q_i - 1} \right\rfloor \cdot \frac{1}{2} q_i(q_i - 1) + \sum_{j = q_i - (k_i - 1) \mod (q_i - 1)}^{q_i} j.$$

Proof. Given a number of packets k_i and an arrival spread q_i, we determine the arrival pattern $(a_{i,p})_{p\le q_i}$ such that $\sum_{j=1}^{q_i} a_{i,j}(S_i + j - 1)$ is minimized. In order to get a feasible arrival pattern we are restricted to arrival patterns where $a_{i,1} \le \cdots \le a_{i,q_i-1}$. We choose $a_{q_i} = 1$, and divide the $k_i - 1$ leftover packets evenly over the $q_i - 1$ leftover arrival times such that $a_{i,1} \le \cdots \le a_{i,q_i-1}$. Thus:

$$a_{i,1} = \cdots = a_{i,p} = \left\lfloor \frac{k_i - 1}{q_i - 1} \right\rfloor, \quad a_{i,p+1} = \cdots = a_{i,q_i-1} = \left\lfloor \frac{k_i - 1}{q_i - 1} \right\rfloor + 1, \quad a_{i,q_i} = 1,$$

where $p = q_i - 1 - ((k_i - 1) \mod (q_i - 1))$. The total cost that corresponds to this arrival pattern is the $Q_i(S_i, q_i)$ described in the lemma. ◀

In order to find an example the expression mentioned in Theorem 16, we pick $S_i = 1$ and define $Q_i(q_i) := Q_i(1, q_i)$ for each $i \in N$. Then, we use Lemma 17 and it is left to maximize

$$P((q_i)_{i\in N}) := \frac{\sum_{i\in N}\left(Q_i(q_i) + q_i \sum_{j=i+1}^{n} k_j\right)}{\sum_{i\in N} Q_i(q_i)}. \tag{3}$$

Thus, in order to find an example that maximizes the price of anarchy, we only need to decide on a q_i for each player. In order to do so, we define $\mu_{i,OPT}(p) := Q_i(p+1) - Q_i(p)$ and $\mu_{i,NE}(p) := Q_i(p+1) - Q_i(p) + \sum_{j=i+1}^{n} k_j$. Intuitively, if a player decides to increase q_i from p to $p+1$, it would add a cost of $\mu_{i,OPT}(p)$ to the social optimum and a cost of $\mu_{i,NE}(p)$ to the worst equilibrium. We state Algorithm 1 using $\mu_{i,OPT}(p)$ and $\mu_{i,NE}(p)$.

▶ **Theorem 18.** *Given a set of players N, where each player has a demand k_i. Then, Algorithm 1 returns an sequence $q := (q_i)_{i\in N}$ that maximizes $P(q)$ as defined in* (3).

Proof. First note, that by definition

$$\frac{\sum_{i\in N}\sum_{p=0}^{q_i-1} \mu_{i,NE}(p)}{\sum_{i\in N}\sum_{p=0}^{q_i-1} \mu_{i,OPT}(p)} = \frac{\sum_{i\in N}\left(Q_i(q_i) + q_i \sum_{j=i+1}^{n} k_j\right)}{\sum_{i\in N} Q_i(q_i)}.$$

Let $q' \in \arg\max P(q)$ and let $(q_i)_{i\in N}$ be the output of Algorithm 1. Assume $(q_i)_{i\in N}$ is not optimal, thus $P(q') > P(q)$. We first prove that $q'_i \le q_i$ for all $i \in N$.

We define q_{-i} to be the vector $(q_j)_{j\in N\setminus\{i\}}$. For sake of contradiction, assume that there exists an $i \in N$ such that $q'_i > q_i$. We distinguish two cases:

Algorithm 1: Creating an example with maximized price of anarchy.

```
    Input: A set $N$ consisting of $n$ players with $k_i$ packets.
    Output: A vector $(q_i)_{i \in N}$.
 1  $q_i \leftarrow 1$ for all $i \in N$;
 2  for $i \in N$ do
 3      $p_i \leftarrow \arg\max_{q_i \le p < k_i} \left\{ \frac{\sum_{q=q_i}^{p} \mu_{i,NE}(q)}{\sum_{q=q_i}^{p} \mu_{i,OPT}(q)} \right\}$;
 4      $P_i \leftarrow \max_{q_i \le p < k_i} \left\{ \frac{\sum_{q=q_i}^{p} \mu_{i,NE}(q)}{\sum_{q=q_i}^{p} \mu_{i,OPT}(q)} \right\}$;
 5  end
 6  $j \leftarrow \arg\max_{i \in N}\{P_i\}$;
 7  while $P_j > \frac{\sum_{i \in N} \left( Q_i(q_i) + q_i \sum_{j=i+1}^{n} k_j \right)}{\sum_{i \in N} Q_i(q_i)}$ do
 8      $q_j \leftarrow p_j + 1$;
 9      $p_j \leftarrow \arg\max_{q_j \le p < k_j} \left\{ \frac{\sum_{q=q_j}^{p} \mu_{j,NE}(q)}{\sum_{q=q_j}^{p} \mu_{j,OPT}(q)} \right\}$;
10      $P_j \leftarrow \max_{q_j \le p < k_j} \left\{ \frac{\sum_{q=q_j}^{p} \mu_{j,NE}(q)}{\sum_{q=q_j}^{p} \mu_{j,OPT}(q)} \right\}$;
11      $j \leftarrow \arg\max_{i \in N}\{P_i\}$;
12  end
13  return $(q_i)_{i \in N}$
```

1. $P(q_{-i}, q'_i) > P(q)$. In this case the algorithm would not terminate. From the assumption $P(q_{-i}, q'_i) > P(q)$ we get that
$$\frac{\sum_{i \in N} \sum_{p=0}^{q_i-1} \mu_{i,NE}(p) + \sum_{p=q_i}^{q'_i-1} \mu_{i,NE}(p)}{\sum_{i \in N} \sum_{p=0}^{q_i-1} \mu_{i,OPT}(p) + \sum_{p=q_i}^{q'_i-1} \mu_{i,OPT}(p)} > \frac{\sum_{i \in N} \sum_{p=0}^{q_i-1} \mu_{i,NE}(p)}{\sum_{i \in N} \sum_{p=0}^{q_i-1} \mu_{i,OPT}(p)}.$$
Thus,
$$\frac{\sum_{p=q_i}^{q'_i-1} \mu_{i,NE}(p)}{\sum_{p=q_i}^{q'_i-1} \mu_{i,OPT}(p)} > P(q),$$
is one candidate for P_j determined in line 10 of the algorithm. This candidate is already larger than $P(q)$, thus the algorithm would not terminate with q respectively $P(q)$. Thus, this contradicts the fact that Algorithm 1 outputs q.
2. $P(q_{-i}, q'_i) \le P(q)$. In this case, $(\sum_{p=q_i}^{q'_i-1} \mu_{i,NE}(p))/(\sum_{p=q_i}^{q'_i-1} \mu_{i,OPT}(p)) \le P(q) < P(q')$. Decreasing q'_i to q_i would increase the quotient of $P(q')$, which means $P(q'_{-i}, q_i) > P(q')$. This is a contradiction to $q' \in \arg\max P(q)$.

Thus, we have shown that if q is not optimal, $q'_i \le q_i$ for all $i \in N$. It remains to show that $q'_i < q_i$ leads to a contradiction. Due to the initialization of $q_i = 1$ which is minimal, we know that q_i is less or equal to q'_i at the start of Algorithm 1. Assume that during the execution of Algorithm 1, we obtain the following vectors for $(q_i)_{i \in N}$: $\vec{1}, q^1, \dots, q^k, q$.

If there is a $i \in N$ such that $q'_i < q_i$, during Algorithm 1 there needs to be a vector q^b such that $q^b_j \le q'_j$ for all $j \in N$ and there is an $i \in N$ such that $q^{b+1}_i > q'_i$ and $q^{b+1}_j \le q'_j$ for all $j \in N \backslash \{i\}$. This means q^{b+1} is the vector where for the first time in Algorithm 1 a value of q is increased over a value of q'. By definition of Algorithm 1, we know that the chosen value q^{b+1}_i maximizes the quotient of marginal cost increase of Nash equilibrium over optimal solution among all alternative vectors. This means:

$$\frac{\sum_{p=q_i^b}^{q_i^{b+1}-1} \mu_{i,NE}(p)}{\sum_{p=q_i^b}^{q_i^{b+1}-1} \mu_{i,OPT}(p)} \geq \max_{j\in N} \frac{\sum_{p=q_j^b}^{q'_j-1} \mu_{j,NE}(p)}{\sum_{p=q_j^b}^{q'_j-1} \mu_{j,OPT}(p)}. \tag{4}$$

Furthermore, by the choice of q' we know that:

$$P(q') = \frac{\sum_{i\in N} \left(\sum_{p=0}^{q_i^b-1} \mu_{i,NE}(p) + \sum_{p=q_i^b}^{q'_i-1} \mu_{i,NE}(p)\right)}{\sum_{i\in N} \left(\sum_{p=0}^{q_i^b-1} \mu_{i,OPT}(p) + \sum_{p=q_i^b}^{q'_i-1} \mu_{i,OPT}(p)\right)}.$$

By definition of Algorithm 1:

$$\frac{\sum_{p=q_i^b}^{q_i^{b+1}-1} \mu_{i,NE}(p)}{\sum_{p=q_i^b}^{q_i^{b+1}-1} \mu_{i,OPT}(p)} > P(q^b) = \frac{\sum_{i\in N}\sum_{p=0}^{q_i^b-1} \mu_{i,NE}(p)}{\sum_{i\in N}\sum_{p=0}^{q_i^b-1} \mu_{i,OPT}(p)}. \tag{5}$$

and by (4):

$$\frac{\sum_{p=q_i^b}^{q_i^{b+1}-1} \mu_{i,NE}(p)}{\sum_{p=q_i^b}^{q_i^{b+1}-1} \mu_{i,OPT}(p)} \geq \frac{\sum_{i\in N}\sum_{p=q_i^b}^{q'_i-1} \mu_{i,NE}(p)}{\sum_{i\in N}\sum_{p=q_i^b}^{q'_i-1} \mu_{i,OPT}(p)}. \tag{6}$$

Given $\frac{a_1}{a_2}, \frac{b_1}{b_2}, \frac{c_1}{c_2} \in \mathbb{Q}$, then, whenever $\frac{a_1}{a_2} > \frac{b_1}{b_2}$ and $\frac{a_1}{a_2} \geq \frac{c_1}{c_2}$, it holds that $\frac{a_1}{a_2} > \frac{b_1+c_1}{b_2+c_2}$. We use this type of argumentation on (5) and (6) to obtain:

$$\frac{\sum_{p=q_i^b}^{q_i^{b+1}-1} \mu_{i,NE}(p)}{\sum_{p=q_i^b}^{q_i^{b+1}-1} \mu_{i,OPT}(p)} > P(q').$$

Hence, one could increase $P(q')$ by increasing q'_i to q_i^{b+1}. This contradicts the fact that q' maximizes $P(\cdot)$. ◀

▶ Remark. The running time of the algorithm is polynomial in $k_1, \dots, k_n$ and n.

Proof. In the initial phase we compute n times a maximum value which takes at most k_i^3 time for every $i \in N$. In the while loop, we do the same computation. Note that in every execution of the while loop one q_i for $i \in N$ is increased by at least one. Since the q_i's are initialized as one and bounded from above by k_i, the while loop takes at most $\sum_{i\in N} k_i$ iterations. Hence, the running time of the algorithm is polynomial in $(k_i)_{i\in N}$ and n. ◀

In Appendix F we apply this algorithm to a small example.

References

1 Elliot Anshelevich, Anirban Dasgupta, Jon Kleinberg, Eva Tardos, Tom Wexler, and Tim Roughgarden. The price of stability for network design with fair cost allocation. *SIAM Journal on Computing*, 38(4):1602–1623, 2008.

2 Nadine Baumann and Martin Skutella. Earliest arrival flows with multiple sources. *Mathematics of Operations Research*, 34(2):499–512, 2009.

3 Lisa K Fleischer. Faster algorithms for the quickest transshipment problem. *SIAM journal on Optimization*, 12(1):18–35, 2001.

4 Lisa K Fleischer and Martin Skutella. Quickest flows over time. *SIAM Journal on Computing*, 36(6):1600–1630, 2007.

5 David Gale. Transient flows in networks. *Michigan Math. J.*, 6(1):59–63, 1959.

6 Tobias Harks, Britta Peis, Daniel Schmand, Bjoern Tauer, and Laura Vargas Koch. Competitive packet routing with priority lists. *ACM Trans. Econ. Comput.*, 6(1):4:1–4:26, March 2018.

7 David G. Harris and Aravind Srinivasan. Constraint satisfaction, packet routing, and the lovasz local lemma. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 685–694, New York, NY, USA, 2013. ACM.

8 Martin Hoefer, Vahab S Mirrokni, Heiko Röglin, and Shang-Hua Teng. Competitive routing over time. *Theoretical Computer Science*, 412(39):5420–5432, 2011.

9 John J Jarvis and H Donald Ratliff. Note - some equivalent objectives for dynamic network flow problems. *Management Science*, 28(1):106–109, 1982.

10 Ronald Koch, Britta Peis, Martin Skutella, and Andreas Wiese. Real-time message routing and scheduling. In *Approx., Rand., and Comb. Opt. Algorithms and Techniques*, pages 217–230. Springer Berlin Heidelberg, 2009.

11 Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *Stacs*, volume 99, pages 404–413. Springer, 1999.

12 Janardhan Kulkarni and Vahab Mirrokni. Robust price of anarchy bounds via lp and fenchel duality. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1030–1049. SIAM, 2014.

13 Frank Thomson Leighton, Bruce M Maggs, and Satish B Rao. Packet routing and job-shop scheduling in $\mathcal{O}$(congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

14 Nimrod Megiddo. Optimal flows in networks with multiple sources and sinks. *Mathematical Programming*, 7(1):97–107, 1974.

15 Edward Minieka. Maximal, lexicographic, and dynamic network flows. *Operations Research*, 21(2):517–527, 1973.

16 Britta Peis and Andreas Wiese. Universal packet routing with arbitrary bandwidths and transit times. In *Proceedings of the 15th International Conference on Integer Programming and Combinatoral Optimization*, IPCO'11, pages 362–375, Berlin, Heidelberg, 2011. Springer-Verlag.

17 Thomas Rothvoß. A simpler proof for o(congestion + dilation) packet routing. *CoRR*, abs/1206.3718, 2012.

18 Tim Roughgarden. Routing games. In *Algorithmic Game Theory*, pages 461–486. Cambridge Univ. Press, 2007.

19 Christian Scheideler and Berthold Vöcking. From static to dynamic routing: Efficient transformations of store-and-forward protocols. *SIAM Journal on Computing*, 30(4):1126–1155, 2000.

20 Martin Skutella. An introduction to network flows over time. In *Research trends in combinatorial optimization*, pages 451–482. Springer, 2009.

21 Aravind Srinivasan and Chung-Piaw Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. *SIAM Journal on Computing*, 30(6):2051–2068, 2001.

22 Stevanus Adrianto Tjandra. *Dynamic network optimization with application to the evacuation problem.* Shaker, 2003.

23 William L Wilkinson. An algorithm for universal maximal dynamic flows in a network. *Operations Research*, 19(7):1602–1612, 1971.

A Technical details of the Proof of Theorem 4

We use ideas of [6] to prove that:

$$\min \sum_{\ell \in K_i} C_{i,\ell}(x) = \max \sum_{\theta \in \mathbb{N}_{>0}} A_i(x, \theta - 1).$$

Observe that:

$$\begin{aligned}
&\sum_{\ell \in K_i} C_{i,\ell}(x) \\
=\ & \sum_{\theta \in \mathbb{N}_{>0}} \left(A_i(x,\theta) - A_i(x,\theta-1)\right)\theta \\
=\ & \sum_{\theta \in \mathbb{N}_{>0}} A_i(x,\theta)\theta - \sum_{\theta \in \mathbb{N}_{>0}} A_i(x,\theta-1)\theta \\
=\ & \sum_{\theta \in \mathbb{N}_{>0}} A_i(x,\theta)\theta - \sum_{\theta \in \mathbb{N}_{>0}} A_i(x,\theta-1)(\theta-1) - \sum_{\theta \in \mathbb{N}_{>0}} A_i(x,\theta-1).
\end{aligned}$$

Note that $\sum_{\theta \in \mathbb{N}_{>0}} A_i(x,\theta)\theta = \sum_{\theta \in \mathbb{N}_{>0}} A_i(x,\theta-1)(\theta-1)$, as $A_i(x,0) = 0$. Thus:

$$\min \sum_{\ell \in K_i} C_{i,\ell}(x) = \min - \left(\sum_{\theta \in \mathbb{N}_{>0}} A_i(x,\theta-1) \right) = \max \sum_{\theta \in \mathbb{N}_{>0}} A_i(x,\theta-1).$$

B Cycle free path decomposition

▶ **Lemma 19.** *For any s-t-graph G, there exist an earliest arrival flow for varying capacities that has a path decomposition where no flow is send along cycles.*

Proof of Lemma 19. First we construct an earliest arrival flow by using the algorithm of Tjandra [22]. The algorithm is roughly speaking a successive shortest path algorithm in a network with varying capacities. We prove that there exists a sequence of shortest paths in the successive shortest path algorithm such that the resulting flow does not contain cycles. If no cycles occurs in the flow, then no cycles occur in any path decomposition of the earliest arrival flow.

During the successive shortest path algorithm, cycles can arise in two different ways.

1. During the course of the algorithm, we choose a shortest path that contains a cycle. As all transit times are non-negative, the length of the cycle is bounded from below by zero. Hence, we can delete this cycle and use the resulting (shortest) path.
2. During the course of the algorithm, we add a shortest path P that closes a directed cycle for the first time, by connecting some nodes u and v by forward edges. Thus, there needs to be a directed sequence of edges connecting the nodes v and u. Instead of closing the cycle, the path could also go along this forward edges as backwards edges. Since the cost of the sequence of forward edges is lower bounded by zero and the cost of the backwards edges is upper bounded by zero, this never increases the costs of the path. Thus, this is a feasible choice for a shortest path.

Hence, there exists a sequence of shortest paths such that the resulting flow does not contain any cycles. ◀

C Proof of Lemma 8

Proof of Lemma 8. We prove this lemma by induction. Note that, as player 1 is not affected by other players, $C_{1,\ell}(NE) = C_{1,\ell}(OPT)$. Hence, the lemma clearly holds for the first player.

Assume that the lemma holds for the first $i-1$ players (players with highest priority) then we prove that $C_{i,\ell}(NE) \leq i \cdot C_{i,\ell}(OPT)$. As player i comes after player j on the priority list for any player $j < i$, we have, by construction of OPT, that $C_{i,\ell}(OPT) \geq C_{j,\ell}(OPT)$.

Hence:

$$\frac{C_{j,\ell}(NE)}{C_{i,\ell}(OPT)} \leq \frac{C_{j,\ell}(NE)}{C_{j,\ell}(OPT)} \leq j,$$

where the last inequality holds as of our induction hypothesis. Therefore, we know that:

$$C_{j,\ell}(NE) \leq j \cdot C_{i,\ell}(OPT). \tag{7}$$

Observe that in the worst case, player i can play the same strategy as she did in the optimal solution, but only after all previous players $j < i$ have already left the network. Hence:

$$C_{i,\ell}(NE) \leq \max_{j<i,\ell\in K_j} \{C_{j,\ell}(NE)\} + C_{i,\ell}(OPT). \tag{8}$$

We combine inequalities (7) and (8) to obtain

$$C_{i,\ell}(NE) \leq \max_{j<i,\ell\in K_j} \{j \cdot C_{i,\ell}(OPT)\} + C_{i,\ell}(OPT).$$

Then, $j \cdot C_{i,\ell}(OPT)$ is clearly maximized whenever $j = i-1$. Hence, we obtain:

$$C_{i,\ell}(NE) \leq (i-1) \cdot C_{i,\ell}(OPT) + C_{i,\ell}(OPT) = i \cdot C_{i,\ell}(OPT),$$

which proves the lemma. ◀

D Details of Theorem 11

In this appendix, one can find the technical details of the proof of Theorem 11. We formally prove why:

$$PoA \leq \frac{\sum_{i=1}^{n}\sum_{p=1}^{q_1} a_p(S+p-1+(i-1)q_1)}{knS + \sum_{p=1}^{q_1} a_p(p-1) + (q_1-1)k(n-1)} \leq \frac{1}{2}(n+1).$$

Technical details. During the proof of Theorem 11 we established a lower bound on the cost of a socially optimal profile:

$$C(OPT) \geq knS + \sum_{p=1}^{q_1} a_p(p-1) + (q_1-1)k(n-1).$$

Furthermore, we bounded the cost of any equilibrium from above by:

$$\begin{aligned} C(NE) &\leq \sum_{i=1}^{n}\sum_{p=1}^{q_1} a_p(S+p-1+(i-1)q_1) \\ &= n\left(\sum_{p=1}^{q_1} a_p(S+p-1)\right) + \tfrac{1}{2}q_1kn(n-1) \\ &= knS + n\left(\sum_{p=1}^{q_1} a_p(p-1)\right) + \tfrac{1}{2}q_1kn(n-1). \end{aligned}$$

As we have a lower bound on the social cost, and an upper bound of the cost in any Nash equilibrium, we can find an upper bound on the price of anarchy.

$$PoA \leq \frac{knS + n\left(\sum_{p=1}^{q_1} a_p(p-1)\right) + \frac{1}{2}q_1kn(n-1)}{knS + \sum_{p=1}^{q_1} a_p(p-1) + (q_1-1)k(n-1)}$$

Note that for all $n \geq 1, k \geq 1$, we have that

$$n\left(\sum_{p=1}^{q_1} a_p(p-1)\right) + \tfrac{1}{2}q_1kn(n-1) \geq \left(\sum_{p=1}^{q_1} a_p(p-1)\right) + (q_1-1)k(n-1) \geq 0.$$

As $k, n, S \geq 1$, the PoA is maximized when knS is minimal, which is the case when $S = 1$. We obtain:

$$\begin{aligned} PoA &\leq \frac{kn + n\left(\sum_{p=1}^{q_1} a_p(p-1)\right) + \frac{1}{2}q_1kn(n-1)}{kn + \sum_{p=1}^{q_1} a_p(p-1) + (q_1-1)k(n-1)} \\ &= \frac{n\left(\sum_{p=1}^{q_1} pa_p\right) + \frac{1}{2}q_1kn(n-1)}{\sum_{p=1}^{q_1} pa_p + q_1k(n-1)} \\ &= \frac{n\left(\sum_{p=1}^{q_1} pa_p + q_1k(n-1)\right) - \frac{1}{2}kq_1n(n-1)}{\sum_{p=1}^{q_1} pa_p + q_1k(n-1)} \\ &= n - \frac{\frac{1}{2}q_1kn(n-1)}{\sum_{p=1}^{q_1} pa_p + q_1k(n-1)}. \end{aligned}$$

Thus, it is left to minimize $\left(\frac{1}{2}q_1kn(n-1)\right) / \left(\sum_{p=1}^{q_1} pa_p + q_1k(n-1)\right)$. Note that for any q_1, $\sum_{p=1}^{q_1} pa_p$ is maximized when $a_p = 1$ for $p \in \{1, \ldots, q_1 - 1\}$ and $a_{q_1} = k - q_1 + 1$. Hence, for any q_1,

$$\sum_{p=1}^{q_1} pa_p \leq \tfrac{1}{2}q_1(q_1-1) + q_1(k-q_1+1) = q_1(k - \tfrac{1}{2}(q_1-1)).$$

Thus,

$$\begin{aligned} n - \frac{\frac{1}{2}q_1kn(n-1)}{\sum_{p=1}^{q_1} pa_p + q_1k(n-1)} &\leq n - \frac{\frac{1}{2}q_1kn(n-1)}{q_1(k - \frac{1}{2}(q_1-1)) + q_1k(n-1)} \\ &= n - \frac{\frac{1}{2}kn(n-1)}{kn - \frac{1}{2}(q_1-1)}. \end{aligned}$$

As $q_1 \geq 1$, the PoA is clearly maximized when $q_1 = 1$. We obtain:

$$PoA \leq n - \frac{\frac{1}{2}kn(n-1)}{kn - \frac{1}{2}(1-1)} = n - \frac{\frac{1}{2}kn(n-1)}{kn} = \tfrac{1}{2}(n+1),$$

which proves the theorem. ◀

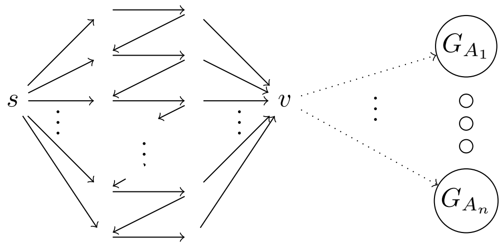


Figure 13 Braess graph BG with player specific paths to individual sink.

E Proof of Theorem 16

Proof. We denote the arrival pattern of a player i in an optimal solution where player i is the only player in the network by $A_i := (a_{i,1}, \dots, a_{i,q_i})$. Here $a_{i,p}$ denotes the number of packets that arrive at time $S_i + p - 1$. At time $S_i + q_i - 1$ the last packet of player i arrives, i.e. player i has an arrival spread of $q_i := \arg\min_{p \in \mathbb{N}_{>0}}\{a_{i,q'} = 0, \forall q' > 0\}$, thus, $\sum_{p=1}^{q_i} a_{i,p} = k_i$. Again note that $1 \leq a_{i,1} \leq a_{i,2} \leq \dots \leq a_{i,q_i-1}$. This holds true with a simple following argumentation. If p packets arrive at time θ, p further packets can arrive at time $\theta + 1$ by following the first p packets.

Given a set of players N with $(k_i)_{i \in N}$ and $(q_i)_{i \in N}$ with $k_i \geq q_i$, we can construct an arrival pattern $A_i := (a_{i,1}, \dots, a_{i,q_i})$ for every player i realizing her k_i and q_i in the following way:

$$a_{i,1} = \dots = a_{i,p} = \left\lfloor \frac{k_i - 1}{q_i - 1} \right\rfloor, \quad a_{i,p+1} = \dots = a_{i,q_i-1} = \left\lfloor \frac{k_i - 1}{q_i - 1} \right\rfloor + 1, \quad a_{i,q_i} = 1,$$

where $p = q_i - 1 - ((k_i - 1) \mod (q_i - 1))$.

For every player i with arrival pattern $A_i := (a_{i,p})_{1 \leq p \leq q_i}$, we construct a corresponding s_i-t_i-graph $G_{A_i} = (V_{A_i}, E_{A_i})$ consisting of only parallel s_i-t_i-edges, such that the arrival pattern of the earliest arrival flow of G_{A_i} matches A_i. In order to do so, we first define $a_{i,0} = 0$. Then, we add $\max\{a_{i,p} - a_{i,p-1}, 0\}$ parallel edges of length $S + p - 1$ and capacity one for all $1 \leq p \leq q_i$ to graph G_{A_i} for all $i \in N$.

We define $K := \sum_{i \in N} k_i$, and define $BG(K)$ as in Section 2. We connect B_K and G_{A_i} by setting $v \in V_{BG(K)}$ equal to $s_i \in V_{A_i}$ for all $i \in N$ as in Figure 13.

In a socially optimal solution, each player $i \in N$ can enter their graph G_{A_i} at time zero, and thus arrive at sink t_i according to arrival pattern A_i, resulting in a social cost $\sum_{i \in N} C_i(OPT_i)$. In the worst Nash equilibrium, player i blocks graph $BG(K)$ for q_i units of time, delaying all players $j > i$ by q_i time units. This results in a total cost of $\sum_{i \in N} \left(C_i(OPT_i) + \sum_{j=i+1}^{n} q_i k_j \right)$. As this holds for all players $i \in N$, this gives us the desired price of anarchy. ◀

F Example algorithm 1

▶ **Example 20.** For a better understanding of the algorithm we apply it to a small example. We are given two players with a demand of four each, and we return a game that maximizes the price of anarchy for the given demands. We start with $q_1 = q_2 = 1$. In the for loop of Algorithm 1 p_i and P_i are determined. We start with player 1: for increasing q_1 from 1 to 2, we get a quotient of $\frac{7}{3}$, for increasing it to 3 we get $\frac{12}{4} = 3$ and for increasing it from 1

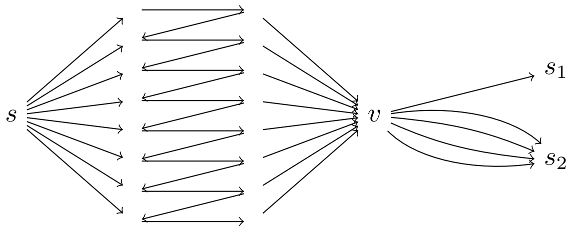


Figure 14 Example with price of anarchy of $\frac{30}{14}$.

to 4 we get $\frac{18}{6} = 3$. Thus, we obtain $(p_1, P_1) = (3, \frac{12}{4})$. Similarly, for player 2 we obtain $(p_2, P_2) = (2, 1)$. Therefore, after the first for loop we choose $j \leftarrow 1$.

Since $\frac{12}{4} > \frac{4+4+4}{4+4}$ we enter into the `while` loop. We increase q_1 from 1 to 3. and update the values of p_1 and P_1, $(p_1, P_1) \leftarrow (4, \frac{6}{2})$. Hence, again $j \leftarrow 1$.

Since $\frac{6}{2} > \frac{24}{12}$ we enter the `while` loop a second time. We set $q_1 = 4$ and update p_1 and P_1. Since q_1 cannot be increased, $P_1 = 0$. Hence, $j \leftarrow 2$.

Since $\frac{1}{1} > \frac{30}{14}$ is not correct, we do not enter the `while` loop again and return $q = (4, 1)$. This results in a price of anarchy of $\frac{30}{14}$. Note that this is larger than two and thus strictly worse than in the single commodity case, where we established an upper bound of n. The graph realizing this price of anarchy is depicted in Figure 14.

In the optimal solution, the arrival times of player 1 are $1, 2, 3, 4$, and for player 2 we obtain $1, 1, 1, 1$, resulting in a total cost of 14. In the the worst Nash equilibrium, the arrival times of player 1 are $1, 2, 3, 4$, and the arrival times of player 2 are $5, 5, 5, 5$, resulting in a total cost of 30. Hence, the price of anarchy is indeed $\frac{30}{14}$.

The Path&Cycle Formulation for the Hotspot Problem in Air Traffic Management

Carlo Mannino
SINTEF
Forskningsveien 1, Oslo, Norway
carlo.mannino@sintef.no

Giorgio Sartor
SINTEF
Forskningsveien 1, Oslo, Norway
giorgio.sartor@sintef.no

Abstract

The Hotspot Problem in Air Traffic Management consists of optimally rescheduling a set of airplanes that are forecast to occupy an overcrowded region of the airspace, should they follow their original schedule. We first provide a MILP model for the Hotspot Problem using a standard big-M formulation. Then, we present a novel MILP model that gets rid of the big-M coefficients. The new formulation contains only simple combinatorial constraints, corresponding to paths and cycles in an associated disjunctive graph. We report computational results on a set of randomly generated instances. In the experiments, the new formulation consistently outperforms the big-M formulation, both in terms of running times and number of branching nodes.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases Air Traffic Management, Hotspot Problem, Job-shop scheduling, Mixed Integer Linear Programming

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.14

Funding Project OPSTRA (Nr. 267554/080) founded by the Norwegian Research Council

1 Introduction

An important task in Air Traffic Management is the dynamic (re)scheduling of flights in order to preemptively avoid that regions of the airspace would become overcrowded at some point in time after the flights have departed (the frequency at which this scheduling happens is not important in this paper). This is necessary to avoid overburdened air traffic controllers. In fact, in order to guarantee the safety of air travel in large regions, the airspace is partitioned into small volumes called *control sectors*. At any time, each such sector is managed by one or more air traffic controllers. Due to safety reasons, each controller can only watch up to a certain number of airplanes. The maximum number of airplanes controllable in a given sector is called *capacity* (of the sector). If too many airplanes occupy a sector at a given time, then there is an *hotspot* (see, e.g. [1, 2]). Hotspots can be avoided by delaying some flights, holding airplanes on the ground. Our objective is to compute a hotspot-free schedule for a set of airplanes in a large region of the airspace while minimizing the total delay.

For our purposes, it suffices to describe the route of each airplane as an ordered sequence of sectors, starting at the departure airport and ending at the arrival airport. Even if airplanes can adjust their cruising speed to a certain extent, in this paper we assume that such speed is fixed, which implies that the time to traverse a given sector is also fixed. This is in accordance with the standard subdivision of roles in air traffic management. In fact, the speed of an airplane is monitored and possibly adjusted usually only by an air traffic controller within his/her control sector. Instead, the schedule of a flight is assessed and possibly recomputed many hours ahead its original departure time from a central authority (e.g. in Europe, Eurocontrol). This procedure takes into consideration the official timetable for all the flights traversing a region of the airspace and their associated routes. Already this timetable can contain one or more hotspots. More typically, hotspots may emerge because of some unpredicted event, such as a sudden delay in one or more aircraft ground operations, bad weather conditions, or even the reduction of the capacity of one or more sectors.

When a hotspot is predicted, the authorities are required to implement some actions to eliminate it. These actions generally consist of delaying the departure of some of the airplanes. So, a natural problem arises: which airplanes should be held at the departure airport, and for how long? Clearly we would like to minimize a measure of the overall delay that is introduced with these actions. We call this the *Hotspot Problem (HP)*.

A few recent papers address variants of the hotspot problem. In [6], the airspace is subdivided into micro-cells of unit capacity, and airplanes can be delayed at the departure, but only within the assigned time slot. A related problem, but on the side of the airlines rather than of the controlling bodies, is addressed in [7]. Here, the authors assume that, in order to mitigate congestion, the control authority issues a number of flight restrictions (FCA) within feasible time slots for the flights of some airlines. The airline is then confronted with the decision of how to modify flights trajectories in order to satisfy the FCAs. The feasible trajectories are chosen in a predefined, finite set. Finally, in [5] an overarching, time-indexed formulation is developed for a problem which includes, as subproblem, capacity requirements in certain points in space.

All the above mentioned papers focus on modeling issues, using either constraint (CP) or mixed integer (MIP) programming. The resulting formulations are then solved by invoking a state-of-the-art CP or MIP solver. However, in our experience, this approach typically does not suffice to tackle instances of practical size. Indeed, the standard formulations for this kind of problems are the big-M and the time-indexed formulations. The former usually provides weak bounds, and thus large search trees; the latter tends to grow to intractable dimensions very quickly. In this paper we instead develop a new MILP formulation for the Hotspot Problem that allows us to significantly improve over a standard big-M formulation.

2 A MILP big-M model for the Hotspot Problem

We start by introducing a standard big-M model for the Hotspot Problem. It extends the model for job-shop scheduling problem with blocking and no-wait constraints introduced in [4] and exploited in several papers for different transportation problems.

The Hotspot Problem is characterized by a set of sectors S (i.e., the airspace) and a set of flights F. Each sector $s \in S$ is associated with a maximum capacity c_s. A *route node* is a pair (f, s), where $f \in F$ is a flight and $s \in S$ is a sector. For each flight $f \in F$, we define its flight route as an ordered sequence of route nodes: $\big((f, s_1), (f, s_2), \ldots, (f, s_q)\big)$ where s_1, s_q are the sectors in which the departure and arrival airports are located, respectively, and s_{i-1}, s_i are adjacent sectors, for $i = 2, \ldots, q$. With some abuse of notation, we denote by $(f, s{+}1)$ the route node that immediately follows (f, s) in the flight route of f.

Let R be the set of all route nodes for all flights in F, D the set of all departure nodes, and A the set of all arrival nodes. With each route node $(f, s) \in R$ we associate the fixed time $\Lambda_{(f,s)}$, that is the time flight f takes to traverse sector s. This time can be obtained from the the official flight schedule and mainly depends on the entry and exit point of the airplane in that sector. Moreover, we indicate with Γ_f the minimum departure time of a flight f in respect to a certain reference time that is common to all $f \in F$.

We can now start building the MILP model by associating a scheduling variable $t_{(f,s)} \in \mathbb{R}$ to each route node $(f, s) \in R$, where $t_{(f,s)}$ represents the time flight f enters sector s. Note that the time a flight exits a particular sector is equal to the time the flight enters the subsequent sector in its route. We also introduce a fictitious variable $t_o \in \mathbb{R}$, which serves as a reference time for all airplanes. Thus, we have

$$t_{(f,s)} - t_o \geq \Gamma_f, \quad (f, s) \in D. \tag{1}$$

Now let $(f, s), (f, s{+}1) \in R$ be two consecutive route nodes in a particular flight route. Then the following *precedence constraints* must hold:

$$t_{(f,s+1)} - t_{(f,s)} = \Lambda_{(f,s)}. \tag{2}$$

In fact, we assume that each airplane travels at fixed speed throughout its route, but it is allowed to delay its departure.

Now, for each pair of distinct flights $f, g \in F$, we denote by $S(f, g)$ the shared sectors, and for each $s \in S(f, g)$ we introduce the binary quantity x^s_{fg}, which is 1 if and only if f and g meet in s. Consider now a set of distinct flights $\bar{F} \subseteq F$ traversing a sector s, and assume that $|\bar{F}| > c_s$. Then, the following *hotspot constraints* must hold:

$$\sum_{\{f,g\} \subseteq K} x^s_{fg} \leq \binom{c_s + 1}{2} - 1, \quad K \subseteq \bar{F}, |K| = c_s + 1, s \in S. \tag{3}$$

It is easy to see that these constraints are enough to guarantee that at most c_s flights meet in sector s. This is indeed a straightforward application of the well-known *Helly's Theorem* in one dimension, which states that a set of intervals in $\mathbb{R}$ (i.e., the time) has a nonempty intersection if and only if every pair intersects.

Observe that, for a pair of distinct flights f, g traversing a sector s, exactly one of the following three conditions must occur: a) flight f and g meet in sector s, or b) flight f traverses sector s before flight g, or c) flight g traverses sector s before flight f. For each ordered pair of flights $(f, g) \in \bar{F}$, we define y^s_{fg} to be equal to 1 if f exits s before g enters, and 0 otherwise. Then, we have that

$$y^s_{fg} + y^s_{gf} + x^s_{fg} = 1, \quad \{f, g\} \subseteq \bar{F}, s \in S. \tag{4}$$

So, precisely one of the above three variables will be 1 in any feasible solution. Accordingly, for every $\{f, g\} \subseteq \bar{F}, s \in S$, the schedule t will satisfy a family of *disjunctive constraints* that can be modeled by means of a conjunction of big-M constraints as follows:

$$\begin{aligned}
&(i) && t_{(g,s)} - t_{(f,s+1)} \geq -M(1 - y^s_{fg}),\\
&(ii) && t_{(f,s)} - t_{(g,s+1)} \geq -M(1 - y^s_{gf}),\\
&(iii) && t_{(g,s+1)} - t_{(f,s)} \geq -M(1 - x^s_{fg}),\\
&(iv) && t_{(f,s+1)} - t_{(g,s)} \geq -M(1 - x^s_{fg}),\\
& && y^s_{fg}, y^s_{gf}, x^s_{fg} \in \{0, 1\},
\end{aligned} \tag{5}$$

Algorithm 1 An algorithm for the big-M formulation.

```
$\mathcal{P} \leftarrow$ Set of precedence constraints
$\mathcal{H} \leftarrow \emptyset$                                            ▷ Set of hotspot constraints
$\mathcal{D} \leftarrow \emptyset$                                            ▷ Set of disjunctive constraints
$\mathcal{M} \leftarrow \min_{y,x,t} c(t)$, subject to $\mathcal{P}$,$\mathcal{H}$, and $\mathcal{D}$      ▷ MILP model for $BF$
$(y, x, t) \leftarrow$ incumbent solution of $\mathcal{M}$
while true do
    Solve $\mathcal{M}$
    if $y, x, t$ violates a disjunction constraint $D$ then
        $\mathcal{D} \leftarrow \mathcal{D} \cup D$                           ▷ Row generation
        continue
    else if $y, x, t$ violates a hotspot constraint $H$ then
        $\mathcal{H} \leftarrow \mathcal{H} \cup H$                           ▷ Row generation
        continue
    else
        break                                                ▷ Found optimal!
```

where M is a suitably large positive constant, and $t_{(h,s+1)}$ is the time the flight h enters the sector next to s in its route (i.e., the time h exits sector s). Indeed, if $y^s_{fg} = 1$ then (ii) and (iii) and (iv) become redundant, whereas constraint (i) reduces to $t_{(g,s)} - t_{(f,s+1)} \geq 0$, which implies that f exits s before g enters s. Similarly, when (ii) is active, g exits s before f enters. On the other hand, when $x^s_{fg} = 1$, then (i) and (ii) become redundant, whereas (iii) and (iv) are active, implying that both f and g exit the sector s after the other flight enters it (i.e., they meet in s).

In conclusion, a complete MILP formulation for the Hotspot Problem can be obtained by considering constraints (1) and (2) for all routes, and constraints (3), (4), and (5) for all sectors $s \in S$ and all sets $\bar{F} \subseteq F$ of flights exceeding the capacity c_s of s. We call this the big-M formulation (BF).

Let $P \subset \mathbb{R}^p$ be the set of points (y, x, t) satisfying all such inequalities, then our problem reduces to $\{\min c(t) : (y, x, t) \in P\}$.

The objective $c(t)$ may vary from instance to instance, but in this paper it will simply be the (weighted) delay at destination.

In principle, formulation BF could be solved by resorting to any off-the-shelf MILP solver. However, the families of constraints (3), (4) and (5) can grow very quickly[1], making the formulation impractical even for small-medium size realistic instances. A standard way to tackle this issue is to make use of row generation. Namely, constraints are generated dynamically and added to the model only if they are violated by the incumbent integer feasible solution. An algorithm to solve formulation BF is presented in Algorithm 1.

3 A non-compact reformulation

In the previous section we presented a compact formulation that fully characterizes the Hotspot Problem, and we presented an algorithm to solve it in a practical context. However, BF still contains one major sources of complexity. In fact, in order to make the constraints in (5) redundant for certain values of the binary variables, we made use in BF of the (in)famous

[1] The total number of constraints is $O(|S||F|^{c_s})$ in (3), and is $O(|S| \times |F|^2)$ in (4) and (5).

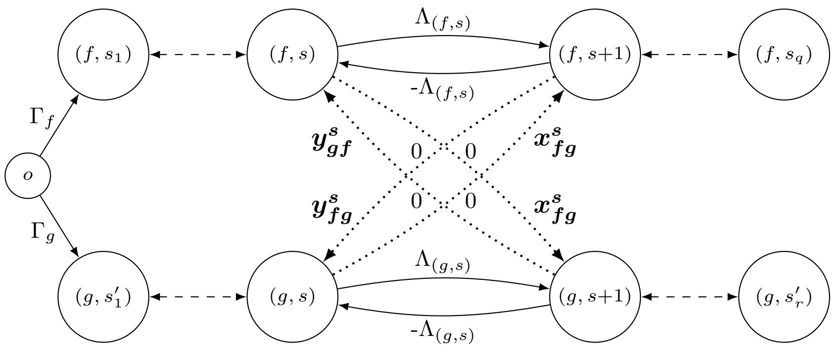


Figure 1 A disjunctive graph for a pair of flights f, g that meet in sector s. Note that, the sector associated with the node $(f, s{+}1)$ might be different from the sector associated with $(g, s{+}1)$. For each flight, the *precedence edges* enforce a fixed traversing time Λ in the corresponding sector. Instead, the zero-weighted *conflict edges* are each associated to a binary variable, and they become binding only if the corresponding binary variable is equal to 1.

big-M method. Unfortunately, including a large coefficient in the model usually makes the formulation weak and prone to return poor bounds in the search trees, often leading to slow solution times.

Our approach to tackle this problem and solve $\{\min c(t) : (y, x, t) \in P\}$ extends the methodology first developed in [3]. In particular, we exploit a Benders-like decomposition to obtain a (master) problem only in the binary variables, plus a few continuous variables to represent the objective function. The decomposition allows us to get rid of big-M coefficients (at the cost of an increased number of linear constraints). Moreover, the constraints of the reformulated master correspond to basic graph structures in the so called disjunctive graph, such as paths and cycles.

We sketch here how the reformulation is obtained. First, we consider the disjunctive graph associated with our big-M formulation BF. This is a directed graph $G = (V, E)$ obtained by considering a vertex for every route node $u \in R$, plus an extra node: the origin o. A directed edge (u, v) of length l_{uv} in the disjunctive graph represents an inequality $t_v - t_u \geq l_{uv}$, indicating that the minimum travel time from route node u to route node v is l_{uv}. Therefore, we can add edges to G to represent some of the constraints of BF.

In particular, the origin is connected with a direct edge (o, d_f) to the node $d_f \in D$, corresponding to the departure node of flight $f \in F$. The length of edge (o, d_f) equals the minimum departure time of flight $f \in F$, Γ_f. Then we add an edge (u, v) of weight Λ_u and an edge (v, u) of weight $-\Lambda_u$, for every constraint (2). These are called the *precedence edges*.

Consider now inequalities (5.i)-(5.iv). For every variable y^s_{fg}, we add the edge (u, v) with length zero, where u, v are the route nodes associated with $t_{(f,s+1)}, t_{(g,s)}$, respectively. In fact, if $y^s_{fg} = 1$ then $t_{(g,s)} - t_{(f,s+1)} \geq 0$. These edges are called *y-edges*, and the set of y-edges is denoted by K_y. Similarly, with every variable x^s_{fg} we associate two edges of length zero: these edges are called *x-edges*, and the set of x-edges is denoted by K_x. For $e \in K_y$ ($e \in K_x$), we let y_e (x_e) be the y-variable (x-variable) that generates e. The set $K_y \cup K_x$ is the set of *conflict edges*.

Figure 1 shows how a disjunctive graph would look like for a couple of flights that meet at least in one sector.

Consider now a feasible solution $(\bar{y}, \bar{x}, \bar{t})$ to (2), (3), (4) and (5). Let $G(\bar{y}, \bar{x})$ be the graph obtained from the disjunctive graph by removing all the edges $e \in K_y$ with $\bar{y}_e = 0$ and all the edges $e \in K_x$ with $\bar{x}_e = 0$. Note that the vector $(\bar{y}, \bar{x}) \in \{0,1\}^{K_y \cup K_x}$ is the incidence vector of the subset of conflict edges contained in $G(\bar{y}, \bar{x})$, and we say that such edges are *selected* by $(\bar{y}, \bar{x})$. Then the following lemma holds.

▶ **Lemma 1.** *i.) $G(\bar{y}, \bar{x})$ does not contain strictly positive directed cycles. ii.) If $(\bar{y}, \bar{x}, \bar{t})$ is an optimal solution, and $\bar{t}_{a_f}$ is the associated arrival time of flight $f \in F$, then $\bar{t}_{a_f}$ equals the length of the longest path from the origin o to route node $a_f \in A$ in $G(\bar{y}, \bar{x})$.*

Proof. When variables y, x are fixed, it is easy to see that the problem BF reduces to the dual of a max-cost flow problem. Then, the result follows immediately from well-known theorems of network theory. ◀

Note that our objective function is simply $c(t) = \sum_{a \in A} t_a$, but the following results can be immediately extended to any function non-decreasing with t.

The lemma has two straightforward consequences: any feasible solution corresponds to a selection $\bar{y}, \bar{x}$ of conflict edges such that $G(\bar{y}, \bar{x})$ does not contain a strictly positive directed cycle; and, for any feasible selection $\bar{y}, \bar{x}$, the best possible scheduling corresponds to the longest path tree in $G(\bar{y}, \bar{x})$.

In this context, the Hotspot Problem (HP) can be stated as follows: *find a feasible selection y, x of conflict edges such that $G(y, x)$ does not contain a strictly positive directed cycle, the sum of the lengths of the longest paths from the origin o to the arrival nodes $a \in A$ is minimum, and the resulting schedule is hotspot-free.*

Let us denote by $\mathcal{C}$ the set of strictly positive length di-cycles of G, and by $L^*(y, x, u)$ the length of the longest path from o to u in $G(y, x)$. Then a new formulation for the Hotspot Problem can be written as follows:

$$
\begin{array}{lll}
\min & \sum_{u \in A} L^*(y, x, u) & \\
s.t. & & \\
(i) & y^s_{fg} + y^s_{gf} + x^s_{fg} = 1, & \{f, g\} \in F, s \in S, \\
(ii) & \sum_{e \in C \cap K_y} y_e + \sum_{e \in C \cap K_x} x_e \leq |C \cap K| - 1, & C \in \mathcal{C}, \\
(iii) & \sum_{\{f,g\} \subseteq \bar{F}} x^s_{fg} \leq \binom{|\bar{F}|}{2} - 1, & s \in S, \bar{F} \subseteq F, |\bar{F}| = c_s + 1, \\
 & y \in \{0,1\}^{|K_y|}, x \in \{0,1\}^{|K_x|}. &
\end{array}
\tag{6}
$$

Constraint (6.ii) ensures that one does not select all the conflict edges contained in a strictly positive di-cycle. Equivalently we write

$$
\begin{array}{lll}
\min & \sum_{u \in A} \mu_u & \\
s.t. & & \\
(i) & y^s_{fg} + y^s_{gf} + x^s_{fg} = 1, & \{f, g\} \in F, s \in S, \\
(ii) & \sum_{e \in C \cap K_y} y_e + \sum_{e \in C \cap K_x} x_e \leq |C \cap K| - 1, & C \in \mathcal{C}, \\
(iii) & \sum_{\{f,g\} \subseteq \bar{F}} x^s_{fg} \leq \binom{|\bar{F}|}{2} - 1, & s \in S, \bar{F} \subseteq F, |\bar{F}| = c_s + 1, \\
(iv) & \mu_u \geq L^*(u, y, x), & u \in A, \\
 & y \in \{0,1\}^{|K_y|}, x \in \{0,1\}^{|K_x|}, \mu \in \mathbb{R}^{|A|}. &
\end{array}
\tag{7}
$$

We can finally rewrite constraints (7.*iv*) in a way that can be immediately exploited in a row generation algorithm. We denote by $\mathcal{H}$ the set of all $G(y,x)$ for (y,x) satisfying (7.*i*), (7.*ii*), (7.*iii*). If $H \in \mathcal{H}$, then we denote by $P_u(H)$ the (set of edges of a) longest path from o to u in H, and by $L_u(H)$ the length of $P_u(H)$. The final reformulation can now be written as follows:

$$
\begin{array}{lll}
\min & \sum_{u\in A} \mu_u & \\
s.t. & & \\
(i) & y^s_{fg} + y^s_{gf} + x^s_{fg} = 1, & \{f,g\} \in F, s \in S,\\
(ii) & \sum_{e\in C\cap K_y} y_e + \sum_{e\in C\cap K_x} x_e \le |C\cap K| - 1, & C \in \mathcal{C},\\
(iii) & \sum_{\{f,g\}\subseteq \bar F} x^s_{fg} \le \binom{|\bar F|}{2} - 1, & s\in S, \bar F \subseteq F, |\bar F| = c_s + 1,\\
(iv) & L_u(H)\Big(\sum_{e\in K_y\cap P_u(H)} y_e + \sum_{e\in K_x\cap P_u(H)} x_e & \\
 & \qquad -|K\cap P_u(H)| + 1\Big) \le \mu_u, & u\in A, H\in\mathcal{H},\\
 & y\in\{0,1\}^{|K_y|}, x\in\{0,1\}^{|K_x|}, \mu\in \mathbb{R}^{|A|}. &
\end{array}
\tag{8}
$$

Indeed, consider a feasible solution $(\bar y, \bar x, \bar\mu)$ to (8). Let $\bar H = G(\bar y,\bar x)$ and let $\bar P_u(\bar H)$ be a longest path from o to u in $\bar H$. Then all conflict edges on $\bar P_u(\bar H)$ are selected by $\bar y, \bar x$ and we have

$$\sum_{e\in K_y\cap \bar P_u(\bar H)} \bar y_e + \sum_{e\in K_x\cap \bar P_u(\bar H)} \bar x_e - |K\cap \bar P_u(\bar H)| + 1 = 1$$

which in turn implies

$$\bar\mu_u \ge L_u(\bar H) = L^*(\bar y, \bar x, u).$$

On the other hand, when one or more edges in a path are not selected, then the constraint is satisfied for any $\mu_u \ge 0$.

We call Problem (8) the *Path&Cycle* formulation (*PC*) of the Hotpot Problem and we solve it with the algorithm described in Algorithm 2. Constraints (8.i) are called the *selection constraints*, (8.ii) are called the *cycle constraints*, (8.iv) are called the *path constraints*, and (8.iii) are called the *hotspot constraints*.

In short, the algorithm works by generating combinations of the x, y variables such that $\sum_{u\in A}\mu_u$ is minimized. If a particular solution $(\bar y, \bar x)$ is such that $G(\bar y,\bar x)$ contains a positive cycle or (8.1) is not satisfied, then the corresponding constraint is added the problem. Otherwise, the longest paths $L_u(G(\bar y,\bar x)), u\in A$ are computed. If there exists a $u \in A$ such that $L_u(G(\bar y,\bar x)) > \mu_u$, then the corresponding path constraint is added to problem. Otherwise, the algorithm is able to use variables x, y, μ to produce a schedule for the t variables. If this schedule violates the capacity of any of the sectors, then the corresponding hotspot constraint is added to the problem. Finally, if none of these inequalities needs to be added, then we found the optimal solution.

4 Computational experiments

In this section we analyze the performance of the *BF* formulation versus the *PC* formulation on randomly generated instances. Each instance represents a snapshot (in time) of the

Algorithm 2 An algorithm for the *Path&Cycle* formulation.

```
$G \leftarrow$ disjunctive graph
$\mathcal{S} \leftarrow \emptyset, \mathcal{H} \leftarrow \emptyset$                                        ▷ Sets of selection and hotspot constraints
$\mathcal{C} \leftarrow \emptyset, \mathcal{P} \leftarrow \emptyset$                                        ▷ Sets of cycle and path constraints
$\mathcal{M} \leftarrow \min_{y,x,\mu} \mu^T \mathbb{1}$, subject to $\mathcal{S}$,$\mathcal{H}$,$\mathcal{C}$, and $\mathcal{P}$                    ▷ MILP model for $PC$
$(y, x, \mu) \leftarrow$ incumbent solution of $\mathcal{M}$
while true do
    while true do
        Solve $\mathcal{M}$
        if $y, x$ violates a selection constraint $S$ then
            $\mathcal{S} \leftarrow \mathcal{S} \cup S$                                        ▷ Row generation
            continue
        else if $y, x$ violates a cycle constraint $C$ then
            $\mathcal{C} \leftarrow \mathcal{C} \cup C$                                        ▷ Row generation
            continue
        else
            Find the longest paths in $G(y, x)$
            if $(y, x, \mu)$ violate a path constraint $P$ then
                $\mathcal{P} \leftarrow \mathcal{P} \cup P$                                    ▷ Row generation
                continue
            else
                break
    if $x, y$ violates a hotspot constraint $H$ then
        $\mathcal{H} \leftarrow \mathcal{H} \cup H$                                            ▷ Row generation
        continue
    else
        break                                                  ▷ Found optimal solution!
```

situation of an airspace made of 400 sectors where 20 airports are placed randomly and a certain number of flights is scheduled (with random departure times) between two randomly chosen airports (see Figure 2). We must say that real-life data is available, but we are not allowed (yet) to use it. However, exploiting the information obtained from the real-life data, we selected 30 "realistic" random instances[2].

The algorithm has been implemented in C$^\sharp$ and interfaced with CPLEX 12.8 using its default parameters except for the following: the number of available threads was set to 1, the advanced start switch was set to 0, and both dual reduction and dynamic search were disabled. The information used for the advanced start (sometimes also called warm start) is poorly exploited by CPLEX in our context, and led to inconsistent results. Instead, dual reduction and dynamic search are automatically disabled by CPLEX when row generation is implemented using callback functions.

The results of Table 1 show a consistent and sometimes dramatic improvement in the solution time of the *PC* formulation. The strength of this new formulation (compared to the *BF* formulation) is demonstrated particularly by the smaller number of branch and bound nodes visited before reaching the optimal solution. This is mostly due to the absence of

[2] The instances are available from the authors upon request.

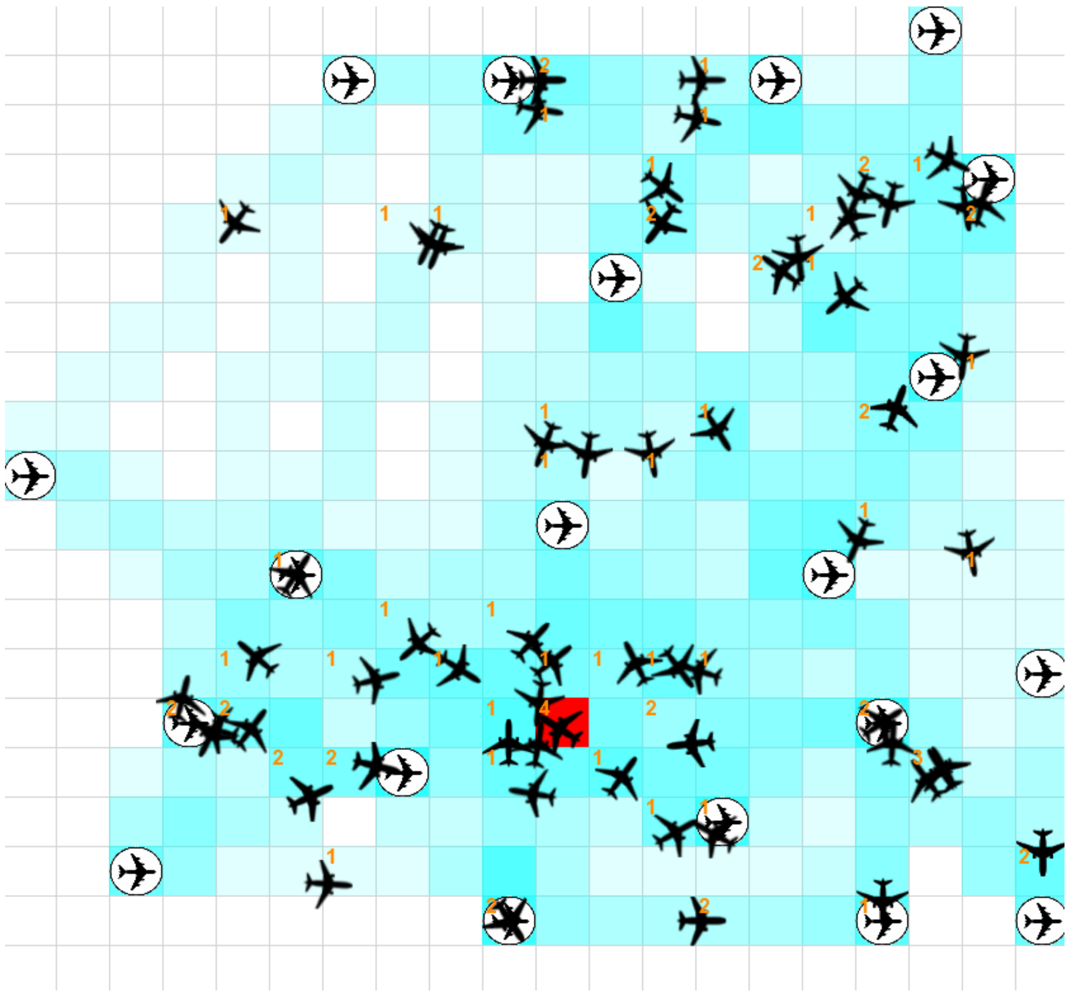


Figure 2 A snapshot of one of the instances where the sector capacity c_s is equal to 3, and a hotspot is highlighted in red. The orange number at the top left corner of each sector shows the current occupancy of the sector. The shades of turquoise simply indicate the number of flight routes traversing a particular sector, helping the visual analysis of an instance.

the large coefficient M in PC. In fact, the LP relaxation of BF at a particular node of the search tree is usually very poor, preventing an effective pruning of the branches of the search tree.

Overall, the *Path&Cycle* formulation for the Hotspot Problem proved to be very promising when compared to a more common formulation. Moreover, this formulation can be easily extended/modified to handle other job-shop scheduling problems.

Table 1 Results on 30 randomly generated instances. The table shows the number of scheduled *flights*, the capacity c_s of the sectors (all the sectors have the same capacity), and how many *hotspots* have been solved by each algorithm (curiously, for these instances they are all equal, although this is not always the case since the hotspot constraints are added dynamically and each algorithm may take a different path to reach the optimal solution). It also reports the total number of *nodes* visited by the branch and bound algorithm, and the total *time* required to find the optimal solution. The last column is the ratio between the computation time of BF and the computation time of PC, and indicates the speed up obtained by using PC instead of BF.

ID	$\|F\|$	c_s	Solved hotspots		Visited nodes		Time (s)		Speed up
			PC	BF	PC	BF	PC	BF	
ATM1	122	3	13	13	1016	19175	1.06	4.99	4.7x
ATM2	137	3	13	13	3062	36806	1.35	10.38	7.7x
ATM3	131	3	8	8	109	774	0.18	0.28	1.5x
ATM4	142	3	13	13	833	40482	0.73	6.43	8.8x
ATM5	110	3	12	12	795	31117	0.39	7.38	18.8x
ATM6	127	3	5	5	79	570	0.16	0.17	1.1x
ATM7	115	3	1	1	0	5	0.05	0.05	1.0x
ATM8	120	3	4	4	2	97	0.05	0.10	1.8x
ATM9	131	3	4	4	42	554	0.08	0.16	2.1x
ATM10	143	3	8	8	76	2313	0.18	0.48	2.6x
ATM11	136	3	15	15	371	39300	0.31	14.90	47.3x
ATM12	142	3	9	9	274	1974	0.22	0.57	2.6x
ATM13	139	3	11	11	118	2217	0.19	0.94	5.1x
ATM14	126	3	7	7	60	2182	0.13	0.59	4.6x
ATM15	139	3	9	9	3625	172950	0.88	50.61	57.4x
ATM16	288	5	4	4	47	1579	0.27	0.71	2.6x
ATM17	289	5	9	9	113	12503	0.38	6.05	16.0x
ATM18	278	5	6	6	183	2188	0.37	1.23	3.3x
ATM19	259	5	3	3	0	296	0.15	0.20	1.4x
ATM20	254	5	5	5	55	1977	0.23	1.01	4.3x
ATM21	279	5	6	6	255	4175	0.32	2.10	6.6x
ATM22	287	5	3	3	0	985	0.11	0.32	2.8x
ATM23	259	5	6	6	37	2452	0.25	1.00	4.0x
ATM24	281	5	4	4	161	1350	0.47	0.60	1.3x
ATM25	296	5	4	4	71	1518	0.22	0.60	2.7x
ATM26	275	5	7	7	50	2872	0.41	1.32	3.2x
ATM27	256	5	5	5	464	5042	0.46	1.58	3.5x
ATM28	273	5	6	6	298	1542	0.60	0.91	1.5x
ATM29	274	5	6	6	193	104627	0.53	35.09	66.7x
ATM30	287	5	7	7	1306	9129	0.75	3.10	4.1x

References

1 Cyril Allignol, Nicolas Barnier, Pierre Flener, and Justin Pearson. Constraint programming for air traffic management: a survey 1: In memory of pascal brisset. *The Knowledge Engineering Review*, 27(3):361–392, 2012.

2 Thomas Dubot, Judicaël Bedouet, and Stéphane Degrémont. Modelling, generating and evaluating sector configuration plans-methodology report of the sesar vp-755 exercise. In *30th Congress of the International Council of the Aeronautical Sciences (ICAS 2016)*, 2016.

3 Leonardo Lamorgese and Carlo Mannino. A non-compact formulation for job-shop scheduling problems in transportation (Dagstuhl Seminar 16171). *Dagstuhl Reports*, 6(4):151, 2016. submited to Operations Research, under revision. `doi:10.4230/DagRep.6.4.139`.

4 Alessandro Mascis and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002.

5 Tolebi Sailauov and ZW Zhong. An optimization model for large scale airspace. *International Journal of Modeling and Optimization*, 6(2):86, 2016.

6 Nina Schefers, Miquel Angel Piera, Juan José Ramos, and Jenaro Nosedal. Causal analysis of airline trajectory preferences to improve airspace capacity. *Procedia Computer Science*, 104:321–328, 2017.

7 Bo Vaaben and Jesper Larsen. Mitigation of airspace congestion impact on airline networks. *Journal of Air Transport Management*, 47:54–65, 2015.

Vehicle Scheduling Based on a Line Plan

Rolf N. van Lieshout
Erasmus University Rotterdam
Econometric Institute, Erasmus University Rotterdam, 3000 DR Rotterdam, The Netherlands
vanlieshout@ese.eur.nl

Paul C. Bouman
Erasmus University Rotterdam
Econometric Institute, Erasmus University Rotterdam, 3000 DR Rotterdam, The Netherlands
bouman@ese.eur.nl

Abstract

We consider the following problem: given a set of lines in a public transportation network with their round trip times and frequencies, a maximum number of vehicles and a maximum number of lines that can be combined into a vehicle circulation, does there exist a set of vehicle circulations that covers all lines given the constraints. Solving this problem provides an estimate of the costs of operating a certain line plan, without having to compute a timetable first. We show that this problem is NP-hard for any restriction on the number of lines that can be combined into a circulation which is equal to or greater than three. We pay special attention to the case where at most two lines can be combined into a circulation, which is NP-hard if a single line can be covered by multiple circulations. If this is not allowed, a matching algorithm can be used to find the optimal solutions, which we show to be a $\frac{16}{15}$-approximation for the case where it is allowed. We also provide an exact algorithm that is able to exploit low tree-width of the so-called circulation graph and small numbers of vehicles required to cover single circulations.

2012 ACM Subject Classification Applied computing → Transportation, Mathematics of computing → Graph algorithms

Keywords and phrases Vehicle scheduling, integrated railway planning, (fractional) matching, treewidth

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.15

1 Introduction

Traditionally, the planning of public transport services occurs in a number of steps. First, a *line plan* is constructed where service routes, usually referred to as lines, are selected such that high quality service is provided to the customers [5, 14]. In the second step, a *timetable* is constructed that specifies the departure and arrival times along the stops of all lines [6, 10]. In the final step, *vehicles* and possibly *human resources* are planned as they are necessary resources to execute the services [1, 8, 11]. As the individual scheduling steps are already quite challenging, the sequential planning approach is traditionally applied because an integrated approach is computationally not tractable. The disadvantage of the sequential approach is that the objectives of the subsequent steps are not taken into account when the prior steps are solved. In particular, the line plan and timetable are usually optimized based on passengers' convenience, while the vehicle schedule is optimized based on the operator costs. Therefore, the optimal solution for the combined problem is likely to be missed.

Recently, a number of authors have proposed ideas to integrate the separate planning steps. One example, the *eigenmodel* [15], replaces a fixed order with an iterative approach that takes a different route through the separate steps, controlling both the passengers'

18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018).
Editors: Ralf Borndörfer and Sabine Storandt; Article No. 15; pp. 15:1–15:14
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

convenience and the operator costs during the process. Another approach [12] incorporates penalties during the line planning phase for lines which can not be covered efficiently by a vehicle in a periodic timetable. That is, assuming the cycle time is 60 minutes, a line with frequency one for which a round trip takes 54 minutes (a downtime of 6 minutes) is given a low penalty, while a line with frequency one for which a round trip takes 65 minutes (a downtime of 55 minutes) is given a very high penalty.

In this paper, we consider the construction of vehicle schedules based on the line plan without the intermediate step of constructing a timetable. One goal of this is to quickly assess whether a line plan can be operated using a small number of vehicles. This allows public transport operators to detect potential inefficiencies early in the planning process, without having to compute a timetable first. The novel aspect of our approach is that we explicitly consider the possibility to combine lines into larger vehicle circulations. To illustrate, while a line that takes 65 minutes with a period of 60 minutes may seem inefficient by itself, it may be a good option if we can combine it with a line of 55 minutes. Although combinations of lines can help to reduce the number of vehicles required to operate a line plan, large and complex combinations of lines may result in greater dependencies between the operations of the different lines. Therefore, we provide a detailed examination of cases where at most two lines can be combined in a vehicle schedule.

The remainder of this paper is organized as follows. In Section 2 we introduce the vehicle circulation scheduling problem. In Section 3 we study the computational complexity of the general case. In Section 4 we study the special case where only two lines can be combined in a circulation. We conclude and discuss ideas for future research in Section 5.

2 Problem formulation

In this paper, we assume a *line plan* is already given and want to determine the minimum number of vehicles that are required to operate the line plan *without* the intermediate step of constructing a timetable. For the line plan, we have a fixed time period denoted by T and a set of lines $\mathcal{L}$ where a line $\{v, u\} \in \mathcal{L}$ is an unordered pair of terminal stations of the line. The *line graph* $L = (V, \mathcal{L})$ has terminal stations V as vertices and the lines as edges. In the line plan each line $l \in \mathcal{L}$ has a round trip time t_l and an integer frequency f_l assigned to it. The round trip times specified by the line plan should at least include the minimum driving and dwelling times required to execute the line, but can also include some slack to make operations more robust against disruptions. The frequency defines the number of times the line service must be executed by a vehicle within each time period of length T. If vehicles are only allowed to operate a single line, the number of vehicles required for line l equals $\lceil \frac{t_l}{T} f_l \rceil$. In order to reduce the number of vehicles required to operate the line plan, public transport operators may consider *circulations*, which are a combination of lines that can be executed by a single vehicle. Formally, a circulation $c \subseteq \mathcal{L}$ is a set of lines that can be operated by one or more vehicles. We allow lines to be contained more than once in the set, since this can be relevant if the line has a frequency greater than 1. We call the number of times a line l is contained in a circulation the *multiplicity* of l in c. Furthermore, we assume that a summation over the lines in the circulation includes a line multiple times if it has a multiplicity greater than 1. An example of a situation where we want to assign the same line to a circulation multiple times, is a line with a round trip time of $\frac{T}{2}$ and a frequency of 2. In that case, we want to consider a circulation where we execute the line twice during each period.

For this paper, we only consider circulations c such that the lines it contains form a connected subgraph of L. As a consequence of this, the time t_c needed to perform a single round trip of a circulation c can be expressed as $t_c = \sum_{l \in c} t_l$. Furthermore, a circulation c corresponds to a directed Eulerian tour in $\overleftrightarrow{L}$, where $\overleftrightarrow{L}$ is the *directed line graph*, which is a a symmetric directed graph derived from L where each edge of L is replaced by two arcs, one for each direction. Let us now consider the correspondence between a connected subset of L and the directed cycle in $\overleftrightarrow{L}$.

▶ **Lemma 1.** *A connected subset $c \subseteq \mathcal{L}$ of lines in the line graph L corresponds to a directed Eulerian sub-graph in the directed line graph $\overleftrightarrow{L}$. Thus, there always exists a directed cycle in $\overleftrightarrow{L}$ that visits all arcs that correspond to both directions of the lines in c a number of times that is exactly equal to the multiplicity of the lines in the circulation.*

Proof. A directed graph contains a directed Eulerian cycle if two conditions hold: (1) for every vertex the in-degree is equal to the out-degree, and (2) the graph is strongly connected. Since the graph $\overleftrightarrow{L}$ is symmetric, each vertex must have one outgoing arc for each incoming arc, and thus condition (1) always holds. As c is a connected subset of lines, the corresponding lines in $\overleftrightarrow{L}$ must be connected as well. Since the graph is symmetric, this implies that it is also strongly connected. ◀

Although more general concepts of circulations that do not enforce connectivity can be considered, these would require dead-heading of vehicles as part of a line plan. While public transport operators have to use dead-heading when operations start up, or frequencies of lines are changed throughout the day, it is usually avoided as much as possible by public transport operators during regular operations. As we focus on regular operations, we consider those generalized concepts of circulations beyond the scope of this paper.

In the problem we consider we do not only need to decide which circulations should be used, but we also have to decide how many vehicles must be assigned to each circulation to cover the constraints of the line plan. Since a circulation c can have a round trip time that is larger than T, we may need to assign multiple vehicles to it in order to enforce that every line in the circulation is covered during every period. We define the number of vehicles required to perform the circulation once in every period as $k_c = \lceil \frac{t_c}{T} \rceil$. If we would assign fewer vehicles than k_c to a circulation, the vehicle is not finished with its circulation when a new period starts and as a consequence no vehicle executes the circulation during some periods. Furthermore, an upper bound on the number of vehicles that can be assigned to a circulation c is given by the expression $\min_{l \in c} f_l k_c$, as assigning more vehicles to the circulation would imply that the line where the minimum was attained is executed with greater frequency than the line plan prescribes. In the special case that all frequencies are 1, k_c itself is an upper bound on the number of vehicles that can be assigned to c.

Note that we do not enforce that each line is covered by a single circulation. For example, suppose we have two lines a and b with round trip times $t_a = t_b = 30$, but different frequencies $f_a = 3$ and $f_b = 1$, with a period time of $T = 60$. The most efficient way to cover these lines is to have one vehicle circulation that executes line a two times each period, while a second vehicle alternates between line a and line b, executing both lines once each period. We investigate a strict version of the problem where a line can only be covered by a single circulation in Section 4.2.

The goal of the problem studied in this paper is to find a set of circulations and the number of vehicles assigned to each circulation such that all lines are covered. From a practical point of view it is desirable that the selected circulations do not contain too many lines, as this creates significant dependencies in the operations that make the operations extremely sensitive to minor disruptions. When a disruption occurs somewhere in a circulation, all subsequent lines in the circulation are affected. Furthermore, very large circulations can only be operated in practice if the timetables of all the lines in the circulation are synchronized. This may lead to problems in the timetabling phase, especially if you want to offer good transfer possibilities to passengers who want to follow different paths than the vehicles.

To avoid large circulations, we consider a restriction on the maximum number of lines than can be included in a circulation. We call a circulation c an α-circulation if the number of unique lines in c is α. A 1-circulation is also referred to as a *fixed circulation* and a 2-circulation is also referred to as a *combined circulation*. We introduce an input parameter κ that restricts the number of unique lines that can be combined in a single circulation. With this concept clearly defined, we can introduce the decision variant of our problem in a formal way:

VEHICLE CIRCULATION SCHEDULING PROBLEM (VCSP)
INSTANCE: A line graph $L = (V, \mathcal{L})$, a maximum number of of unique lines that are allowed to exist in a single circulation κ and a maximum number of vehicles z
QUESTION: Does there exist a set of circulations C, with for each circulation $c \in C$ a value assigned to the integer decision variable $\theta_c \in \mathbb{N}$ that indicates how many vehicles are assigned to circulation c, such that:
(1) the circulations cover all lines in every period, i.e. $\forall l \in \mathcal{L} : f_l = \sum_{c \in C : l \in c} \frac{\theta_c}{k_c}$,
(2) there are no α-circulations in C with $\alpha > \kappa$, and
(3) at most z vehicles are required to execute all circulations, i.e. $\sum_{c \in C} \theta_c \leq z$.

The optimization version of this problem seeks to find the smallest z for a given line graph and a given parameter κ, such that there exists a set of circulations C with integer vehicle assignments θ that satisfy the conditions.

3 Computational complexity

First, we consider a lower bound on the number of vehicles that is needed in a given line graph. Since we are not allowed to use dead-heading, we must consider the connected components of a line graph separately, as no vehicle will be able to move from one component to another. Thus without loss of generality we assume that a line graph is connected, since if it is not we can decompose the problem into independent sub-problems. A lower bound on the number of vehicles required can be computed by dividing the total running time by the cycle time. As no fractional vehicles can be used, we can round the number of vehicles up. This gives the following necessary condition to check if the instance can possibly be a YES-instance:

$$z \geq \left\lceil \frac{\sum_{l \in \mathcal{L}} t_l \cdot f_l}{T} \right\rceil \tag{1}$$

If we have an instance of the problem where $|\mathcal{L}|$-circulations are allowed, i.e. $\kappa \geq |\mathcal{L}|$, this lower bound can be obtained by a single circulation that contains all lines as many times as their frequency dictates. Thus all such instances are YES-instances.

If we are only allowed to used fixed circulations, i.e. $\kappa = 1$, the only way to cover all lines is to use a fixed circulation for each line. In this case an instance is a YES-instance if and only if z is sufficient for the sum over all fixed circulations:

$$z \geq \sum_{l \in \mathcal{L}} \left\lceil \frac{t_l \cdot f_l}{T} \right\rceil \tag{2}$$

▶ **Theorem 2.** *Any instance with $\kappa \geq |\mathcal{L}|$ for which the condition of Equation 1 holds is a YES-instance. Any instance with $\kappa = 1$ is a YES-instance if and only if the condition of Equation 2 holds.*

As we noted earlier, circulations that do not contain too many lines are preferred as they have many advantages. However, the number of vehicles required with fixed circulations can be significantly greater than when any circulation is allowed. We can see this via application of the following general identity for sums over ceiling functions. Suppose we have a sequence $a_1, \ldots a_n$ of n numbers with $n \geq 2$, then it is straightforward to show that

$$\sum_{i=1}^{n} \lceil a_i \rceil - \left\lceil \sum_{i=1}^{n} a_i \right\rceil \leq n-1 \tag{3}$$

Thus, we can see that the difference between the lower bound of Equation 2 and Equation 1 can become as large as $|\mathcal{L}| - 1$. If we allow 2-circulations, we are able to halve the number of terms in Equation 2 which halves the worst case gap. It thus can be very beneficial to consider restrictions on the circulation κ that are small but strictly greater than one. Unfortunately, for any case with a fixed $\kappa \geq 3$, we can show that the resulting problem becomes NP-hard.

▶ **Theorem 3.** *For any fixed $\kappa \geq 3$, the Vehicle Circulation Scheduling Problem is* NP*-hard.*

Proof. We show this by reduction from 3-partition. Let $S = \{s_1, s_2, \ldots, s_m\}$ be a set of integers such that $\sum_{i=1}^{m} s_i = \frac{m}{3}B$ and $\forall 1 \leq i \leq m : \frac{B}{4} < s_i < \frac{B}{2}$. A 3-partition instance is a YES-instance if it is possible to partition S into $n = \frac{m}{3}$ triplets $S_1, S_2, \ldots S_n$ such that each triplet sums to B.

For the case where $\kappa = 3$, we reduce the 3-partition instance to a line graph $L = (V, \mathcal{L})$ where we have a central hub station $v_0 \in V$ and m external stations $v_1, \ldots, v_m \in V$. This line plan must be periodically operated in a period of $T = B$ time units. Furthermore we have a set of m lines where $l_i \in \mathcal{L} = \{v_0, v_i\}$, with frequency $f_i = 1$ and a round trip and total time equal to s_i. As the sum of the round trip times is exactly mB, the only way to execute this line plan with m vehicles is to have every circulation take B time. Otherwise, at least one circulation will require two vehicles. Thus we set $z = m$. If the 3-partition instance is a YES-instance, we can use the triplets to create 3-circulations with this precise property. If the 3-partition instance is a NO-instance, we can not nicely divide the lines over 3-circulations and thus need at least one additional vehicle.

For cases where $\kappa > 3$, we scale the 3-partition instance by setting $s'_i = 4 \cdot \kappa \cdot s_i$ and $B' = (\kappa - 3) + 4 \cdot \kappa \cdot B$. This way we make sure that $\frac{B'}{4} > \kappa - 3$. We now introduce a set of $\kappa \cdot n$ lines where lines $l_1, \ldots, l_m$ have round trip times $s'_1, \ldots, s'_m$ and lines $l_{m+1}, \ldots, l_{\kappa \cdot n}$ all have round trip time of 1. We define $1 + \kappa$ terminal stations and let lines i connect terminal stations $\{v_0, v_i\}$ in the same structure as the $\kappa = 3$ case. By construction, only circulations that consist of $\kappa - 3$ lines with round trip time 1 and three other lines can sum up to B'. This way it is enforced that it is a YES-instance if and only if the 3-partition instance is a YES-instance. ◀

4 Fixed and combined circulations

Of special interest is the case of $\kappa = 2$ where we are only allowed to have fixed and combined circulations, as this gives us some flexibility to decrease the number of vehicles required to operate the line plan, while we still keep the number of lines in a circulation low. Since all finite cases with $\kappa > 2$ are NP-hard, the $\kappa = 2$ case is also of particular interest from a theoretical perspective.

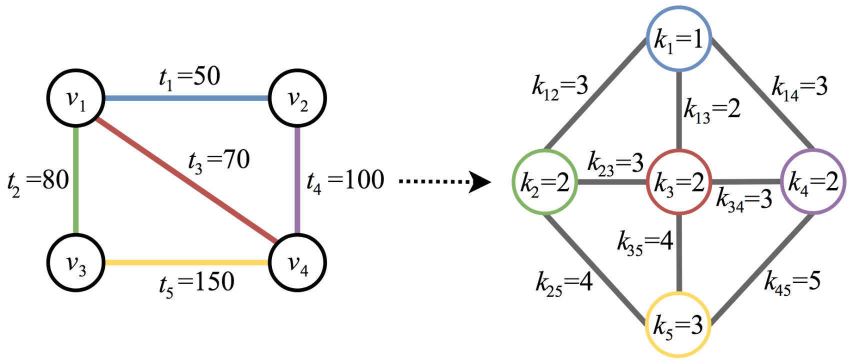


Figure 1 Example of a line graph with round trip times and the corresponding circulation graph. The cycle time T is 60.

For the remainder of this section, we restrict ourselves to instances of the vehicle circulation scheduling problem where the frequency of each line is 1. Although this restriction may seem unrealistic, we can approximate instances with higher frequencies either by splitting up a line l with a frequency higher than 1 into f_l lines with frequency 1 and the same characteristics or by increasing the round trip times as a function of the frequencies. The straightforward approach increases the round trip times based on the frequency, i.e. the new round trip time becomes $f_l \cdot t_l$. More sophisticated approaches can add slack to model the periodicity in more detail in order to increase the probability that a regular timetable exists, possibly at the cost of requiring more vehicles to execute the line plan.

In case $\kappa = 2$ and $f_l = 1$ for all lines, we can represent an instance of the VCSP by a *circulation graph* $G = (\mathcal{L}, E)$. In this graph, the lines are the vertices and the edges are the circulations. The set of edges consists of edges for the fixed circulations E_1 and of the 2-circulations E_2, thus $E = E_1 \cup E_2$. The set E_1 contains a self loop for every line $l_i \in \mathcal{L}$. The set E_2 contains an edge between two lines l_i and l_j if they have a common terminal station. To ease notation, we denote a circulation $\{l_i\} \in E_1$ simply as l_i.

An example of a VCSP instance represented by a circulation graph is given in Figure 1. On the circulation graph, we also depict the k_c value of each circulation c. For the self loops, these values are depicted inside the nodes to make the graph more clear. For example, line 3 has a round trip time of 70 minutes, so $k_3 = \left\lceil \frac{70}{60} \right\rceil = 2$. Line 4 also needs 2 vehicles if it is performed in a fixed circulation, but if lines 3 and 4 are combined only 3 vehicles are required to operate both lines since $k_{34} = \left\lceil \frac{70+100}{60} \right\rceil = 3$.

Using the circulation graph, we can formulate the following optimization problem for the VCSP:

$$\nu(\mathcal{L}) = \min_{\theta} \quad \sum_{c \in E} \theta_c \tag{4}$$

$$\text{s.t.} \quad \frac{1}{k_l}\theta_l + \sum_{c \in E_2 | l \in c} \frac{1}{k_c}\theta_c = 1 \qquad \forall l \in \mathcal{L}, \tag{5}$$

$$\theta_c \geq 0 \text{ and integer} \qquad \forall c \in E, \tag{6}$$

The objective is to minimize the number of used vehicles. We now consider how to rewrite this problem to a maximization problem where the goal is to maximize the number of *saving* circulations. First note that we can rewrite the summation in the objective of Equation 4 by splitting the summation over E into summations over E_1 and E_2, and that by definition a summation over E_1 is equal to a summation over $\mathcal{L}$. We rewrite the objective as follows:

$$\min_{\theta} \sum_{l\in\mathcal{L}} \theta_l + \sum_{c\in E_2} \theta_c \tag{7}$$

In the next step we make use of Constraints 5, which state that a line is included in a sufficient number of circulations. As these constraints imply that $\theta_l = k_l - \sum_{c\in E_2|l\in c} \frac{k_l}{k_c}\theta_c$, we can substitute the left term to obtain

$$\min_{\theta} \sum_{l\in\mathcal{L}} \left(k_l - \sum_{c\in E_2|l\in c} \frac{k_l}{k_c}\theta_c \right) + \sum_{c\in E_2} \theta_c \tag{8}$$

Note that the double summation over $l \in \mathcal{L}$ and $c \in E_2|l \in c$ can also be written as a double summation over $c \in E_2$ and $l \in c$. Rewriting the double summation and reshuffling some terms gives:

$$\sum_{l\in\mathcal{L}} k_l + \min_{\theta} \sum_{c\in E_2} \left(\theta_c - \sum_{l\in c} \frac{k_l}{k_c}\theta_c \right) \tag{9}$$

In the last step we factor out $\frac{\theta_c}{k_c}$ and Equations 4 – 6 are written as:

$$\nu(\mathcal{L}) = \sum_{l\in\mathcal{L}} k_l + \min \left\{ \sum_{c\in E_2} \left[k_c - \sum_{l\in c} k_l \right] \frac{\theta_c}{k_c}, \text{ s.t. } (5) - (6) \right\} \tag{10}$$

If we now apply Equation 3, it can be seen that $\left[k_c - \sum_{l\in c} k_l\right]$ equals either -1 or 0. If the term is -1, we call the circulation *saving*, otherwise we call it *non-saving*. For example, in Figure 1, circulation $\{3, 4\}$ is saving, while circulation $\{1, 2\}$ is non-saving. If we let the set of all saving circulations be denoted as E_2^S, we have that $\nu(\mathcal{L}) = \sum_{l\in\mathcal{L}} k_l - \sigma(\mathcal{L})$ where $\sigma(\mathcal{L})$ is the *savings problem* defined as follows:

$$\sigma(\mathcal{L}) = \max_{\theta} \left\{ \sum_{c\in E_2^S} \frac{\theta_c}{k_c} \text{ s.t. } (5) - (6) \right\} \tag{11}$$

As such, it can be observed that minimizing the number of vehicles is equivalent with maximizing the savings over a vehicle schedule that only uses fixed circulations. In the remainder of this section, we use this observation to give a proof of the NP-hardness of the VCSP with $\kappa = 2$ and $f_l = 1$ for all lines, and to develop an exact algorithm and an $\frac{16}{15}$-approximation algorithm.

4.1 NP-hardness

Our proof depends on the fact that we can construct an arbitrary circulation graph from the line graph if it is accompanied by auxiliary restrictions on which 2-circulations are allowed, which are not part of the formal input to the VCSP. This can be seen from the fact that a star-shaped line graph translates to a complete circulation graph, since we can combine all pairs of lines. If we can provide auxiliary restrictions on which combinations can be combined into circulations and which not, we have complete control over the structure of the circulation graph. In practice, such restrictions are realistic, since the possibility to combine lines into circulations does not only depend on the lines having a shared terminal station, but also on the precise layout of the infrastructure at the terminal station, and whether there exists a type of rolling stock that is able to operate both lines.

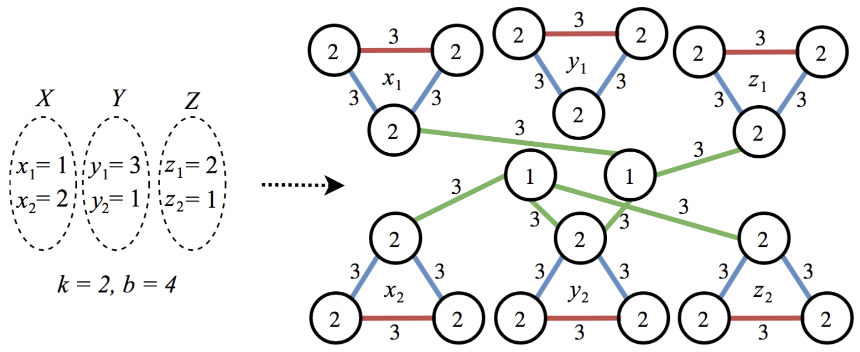


Figure 2 Transformation of a N3DM instance to a VCSP instance represented by a circulation graph.

▶ **Theorem 4.** *The vehicle circulation scheduling problem with* $\kappa = 2$ *and auxiliary restrictions on which* 2*-circulations are allowed is* NP*-hard.*

Proof. Our proof is based on a reduction from the NP-complete NUMERICAL 3-DIMENSIONAL MATCHING problem [9] to an instance of the vehicle circulation scheduling problem expressed by means of a circulation graph. Since each circulation graph can be generated based on a line graph with auxiliary restrictions, this is sufficient to prove the theorem.

The inputs to the N3DM are three multisets of integers X, Y, Z, each containing k elements, and a bound b. An N3DM instance is a YES-instance if there exist k disjoint triples (x, y, z) such that $x + y + z = b$ holds for every triple.

We transform this to the following instance of the VCSP. For every element in X, Y and Z we create three lines in $\mathcal{L}$, all with $k_l = 2$. The three lines can be combined in saving circulations ($k_c = 3$) such that they form a triangle. One of the three lines serves as the *connect line*, the other lines are referred to as *dummy lines*. For every triple (x, y, z) that sums up to b (all such triples can be found in polynomial time), a *triple line* is created with $k_l = 1$, which can be combined in non-saving circulations ($k_c = 3$) with the connect-lines corresponding to x, y and z. Letting μ denote the number of triples that sum up to b, the resulting VCSP instance has $9k + \mu$ lines. We call the generated instance a YES-instance, if the number of required vehicles is at most $14k + \mu$, equivalent to a saving $\sigma(\mathcal{L})$ of at least $4k$. In Figure 2, we visualized this transformation for a small N3DM instance.

We claim that the constructed VCSP instance is a YES-instance if and only if the N3DM instance is a YES-instance.

(if) Each disjoint triple (x, y, z) can be used to generate a saving of 4 by assigning 1 vehicle to each of the 3 circulations combining the triple line with the connect lines (green in Figure 2, 1 vehicle to each of the 6 circulations combining the triples lines with the dummy lines (blue) and 2 vehicles to each of the circulations combining dummy lines (red). In every triangle, this gives a saving of $\frac{4}{3}$, so the total saving generated by every disjoint triple equals 4. As such, if there are k disjoint triples, the total saving is $4k$ and the VCSP instance is indeed a YES-instance.

(only if) Since only the combined circulations in the triangle are saving, if the instance is a YES-instance, every triangle must generate a saving of $\frac{4}{3}$. This implies that in every triangle, the circulations combining the triple lines and dummy lines are assigned 1 vehicle (blue in Figure 2) and the circulations combining dummy lines are assigned 2 vehicles (red). As a consequence, for every connect line, one of the circulations connecting the line with a triple line must be assigned 1 vehicle (otherwise the connect line would not be covered entirely). Next, note that a triple line cannot be partially performed by combined circulations. Hence, if one of the circulations combining a certain triple line and a connect-line is assigned a vehicle, all three such circulations must be assigned a vehicle. So, as every connect line is included in exactly one circulation with a triple line, and as every triple line is included in either zero or three combined circulations, there must be k triple lines that are connected with 3 connect lines. Clearly, the associated triples in the N3DM instance must be disjoint, hence the N3DM instance must also be a YES-instance. ◀

4.2 The strict vehicle circulation scheduling problem

We define the *strict* version of the VCSP to state that each circulation c is either not executed at all, or executed by exactly k_c vehicles. We will now show that the strict version with $\kappa = 2$ and all frequencies equal to 1 can be solved exactly using an approach based on matching.

As in the non strict version, we have the relation that the minimum number of vehicles required under the strictness assumption, denoted as $\bar{\nu}(\mathcal{L})$, equals $\sum_{l \in \mathcal{L}} k_l - \bar{\sigma}(\mathcal{L})$, where $\bar{\sigma}(\mathcal{L})$ denotes the strict savings problem, obtained by replacing $\frac{\theta_c}{k_c}$ with the binary variable γ_c in the regular savings problem $\sigma(\mathcal{L})$:

$$\bar{\sigma}(\mathcal{L}) = \max_{\gamma} \sum_{c \in E_2^S} \gamma_c \tag{12}$$

$$\text{s.t. } \gamma_l + \sum_{c \in E_2 | l \in c} \gamma_c = 1 \quad \forall l \in \mathcal{L}, \tag{13}$$

$$\gamma_c \in \{0, 1\} \qquad \forall c \in E, \tag{14}$$

Constraints 13 now state that a line is either operated with a fixed circulation, or using one of the combined circulations. Since the objective does not contain the γ_c variables for the fixed circulations anymore, the γ-variables for these circulations can be viewed as slack variables for the Constraints 13. Furthermore, since the non-saving circulations have zero contribution to the objective, there always exists an optimal solution that does not contain any non-saving circulations. As a consequence, Constraints 13 can be rewritten as:

$$\sum_{c \in E_2^S | l \in c} \gamma_c \leq 1 \quad \forall l \in \mathcal{L} \tag{15}$$

Since the circulations in E_2^S contain precisely two lines, the resulting formulation is equivalent to a matching problem where we have to maximize the number of selected saving circulations. Thus, we can compute $\bar{\nu}(\mathcal{L})$ by computing the maximum matching in the graph that only contains the edges from E_2^S.

▶ **Theorem 5.** *The strict Vehicle Circulation Scheduling Problem with $\kappa = 2$, $f_l = 1$ for each line $l \in \mathcal{L}$ is solvable in polynomial time.*

4.3 The matching approximation

Since the strict vehicle circulation scheduling problem provides solutions that are also feasible for the regular problem, it can be applied as a heuristic. In this section we derive an approximation guarantee for this heuristic. Our approximation results are based on the observation that the savings problem is a maximization problem and that the linear programming relaxations of the savings problem and its strict version, denoted as $\sigma^{\text{LP}}(\mathcal{L})$

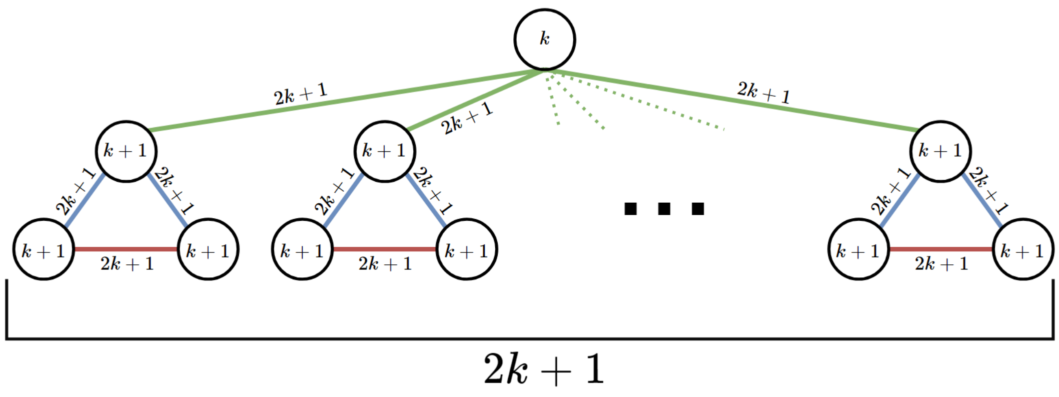


Figure 3 Circulation graph of the instance used to show that the bound of Theorem 6 is tight.

and $\bar{\sigma}^{\mathrm{LP}}(\mathcal{L})$ respectively, are equal. This easily follows from the fact that the strict version is obtained from the non strict version by performing a linear variable substitution, which does not influence the value of the linear relaxation.

▶ **Theorem 6.** *For any line plan it holds that $\bar{\nu}(\mathcal{L}) - \nu(\mathcal{L}) \leq \left\lfloor \frac{|\mathcal{L}|}{6} \right\rfloor$. Furthermore, there exists an instance that attains this bound.*

Proof. Note that we can consider the difference between the savings instead of the difference between the number of vehicles, as $\bar{\nu}(\mathcal{L}) - \nu(\mathcal{L}) = \sum_{l \in \mathcal{L}} k_l - \bar{\sigma}(\mathcal{L}) - \sum_{l \in \mathcal{L}} k_l + \sigma(\mathcal{L}) = \sigma(\mathcal{L}) - \bar{\sigma}(\mathcal{L})$. For any graph it holds that the difference between the value of the maximum fractional matching and the value of the maximum matching is at most $\frac{n}{6}$, with n the number of nodes [7]. This implies that $\bar{\sigma}^{\mathrm{LP}}(\mathcal{L}) - \bar{\sigma}(\mathcal{L}) \leq \frac{|\mathcal{L}|}{6}$. Since the linear programming relaxations of the savings problem and its strict version are equal, it follows that $\sigma(\mathcal{L}) - \bar{\sigma}(\mathcal{L}) \leq \sigma^{\mathrm{LP}}(\mathcal{L}) - \bar{\sigma}(\mathcal{L}) = \bar{\sigma}^{\mathrm{LP}}(\mathcal{L}) - \bar{\sigma}(\mathcal{L}) \leq \frac{|\mathcal{L}|}{6}$. Furthermore, the right hand side of this equation can be rounded down since the difference between savings must be integral.

To show that this bound is tight, consider the circulation graph depicted in Figure 3. The example contains $2k+1$ triangles, where k is a positive integer, and one central node connected to all triangles. The circulations between the lines in the triangles are saving. The value $\bar{\sigma}(\mathcal{L})$ is equal to the size of the maximum matching in the graph induced by all saving circulations, i.e. the graph with only the $2k+1$ triangles. Since we can pick only one circulation in every triangle, we have that $\bar{\sigma}(\mathcal{L}) = 2k+1$. The optimal unrestricted solution is as follows. We can assign 1 vehicle to all green circulations, k vehicles to all blue circulations and $k+1$ vehicles to all red circulations. The objective attained with this solution equals $\sigma(\mathcal{L}) = \sum_{c \in E_2^S} \frac{\theta_c}{k_c} = (2k+1)(\frac{k+k+k+1}{2k+1}) = 3k+1$.

Comparing the two objectives, we have that $\sigma(\mathcal{L}) - \bar{\sigma}(\mathcal{L}) = k$. As the bound equals $\left\lfloor \frac{|\mathcal{L}|}{6} \right\rfloor = \left\lfloor \frac{3(2k+1)+1}{6} \right\rfloor = \left\lfloor k + \frac{4}{6} \right\rfloor = k$, this circulation graph attains the bound for every k. ◀

▶ **Lemma 7.** *If $\sigma(\mathcal{L}) - \bar{\sigma}(\mathcal{L}) = k$, the circulation graph contains at least $2k+1$ disjoint odd cycles of saving circulations.*

Proof. Every vertex x of the fractional matching polytope is half-integral, i.e. $x_e \in \{0, \frac{1}{2}, 1\}$ [2]. Moreover, the edges with $x_e = 1$ form a matching and the set of edges with $x_e = \frac{1}{2}$ form a set of disjoint odd cycles. If the optimal solution to the fractional matching problem contains ω such odd cycles, the difference between the size of the fractional matching and the size of the matching equals $\frac{\omega}{2}$, as a fractional matching in a single odd cycle can improve the objective only by $\frac{1}{2}$ compared to an integer matching in the same cycle. As such, if $\sigma(\mathcal{L}) - \bar{\sigma}(\mathcal{L}) = k$, we certainly have that $\bar{\sigma}^{\mathrm{LP}}(\mathcal{L}) - \bar{\sigma}(\mathcal{L}) \geq k$, implying that the circulation graph contains at least $2k$ disjoint odd cycles of saving circulations.

We prove that the number of odd cycles of saving circulations should be one more than $2k$ by contradiction. As we have already established that the number of odd cycles is at least $2k$, we assume that $\sigma(\mathcal{L}) - \bar{\sigma}(\mathcal{L}) = k$ while there are exactly $2k$ odd cycles. Note that an odd cycle cannot contribute strictly more than $\frac{1}{2}$ to the difference between $\sigma(\mathcal{L})$ and $\bar{\sigma}(\mathcal{L})$, as this violates the fact that fractional matching is the relaxation of the savings problem. Hence, it must hold that every odd cycle contributes exactly $\frac{1}{2}$ to the difference between savings.

However, we will now show that for the VCSP, every odd cycle can increase the difference between $\sigma(\mathcal{L})$ and $\bar{\sigma}(\mathcal{L})$ with strictly less than $\frac{1}{2}$. This is the case since it is not possible to select all circulations in an odd cycle of saving circulations in the circulation graph with value $\frac{1}{2}$. To see this, note that if circulation $c = \{l, m\}$ contributes $\frac{1}{2}$ to the objective of the VCSP, this implies that k_c is even (e.g. $k_c = 4$ and $\theta_c = 2$). Furthermore, since $k_c = k_l + k_m - 1$ (the circulation is saving), it must hold that k_l and k_m have a different parity (one of them is odd, the other even). As such, if we do have an odd cycle in which every circulation is selected with value $1/2$, there must exists a 2-coloring of the vertices of the cycle. Since this is clearly not possible for an odd cycle, we reach a contradiction. ◀

▶ **Theorem 8.** *For any line plan it holds that* $\frac{\bar{\nu}(\mathcal{L}) - \nu(\mathcal{L})}{\nu(\mathcal{L})} \leq \frac{1}{15}$. *Furthermore, there exists an instance where this bound is attained. This implies that* $\bar{\nu}(\mathcal{L})$ *is a* $\frac{16}{15}$*-approximation algorithm for the VCSP with* $\kappa = 2$ *and all frequencies* 1*.*

Proof. First note that

$$\max \frac{\bar{\nu}(\mathcal{L}) - \nu(\mathcal{L})}{\nu(\mathcal{L})} = \max \frac{\sigma(\mathcal{L}) - \bar{\sigma}(\mathcal{L})}{\sum_{l \in \mathcal{L}} k_l - \sigma(\mathcal{L})}. \tag{16}$$

It follows from Lemma 7 that for a given value of $\bar{\nu}(\mathcal{L}) - \nu(\mathcal{L}) = k$, the worst case ratio must be attained by using $2k+1$ cycles of 3 vertices (more or larger cycles only lead to larger values in the denominator). Next to that, for a fixed numerator it is easily seen that the denominator of the ratio is minimized by letting a single node connect all the cycles. This implies that for a given value of $\bar{\nu}(\mathcal{L}) - \nu(\mathcal{L}) = k$, the instance in Figure 3 gives the worst case ration. Maximizing over k gives

$$\max_{k \in \mathbb{N}} \frac{k}{(2k+1)(3k+3) + k - (3k+1)} = \frac{1}{15}, \tag{17}$$

with the maximum being attained at $k = 1$. ◀

4.4 An exact algorithm for bounded treewidth

In this section we consider how to solve the VCSP exactly with $\kappa = 2$ where the circulation graph has a low *treewidth*. Treewidth is a graph property that was introduced by Robertson and Seymour [13] that, informally, indicates how "similar to a tree" the graph is. Many problems that are NP-hard on general graphs, such as independent and dominating set, are solvable in polynomial time if the treewidth of the input graph is bounded by a constant.

Formally, the treewidth of a graph G is the smallest *width* for which there a exists *tree-decomposition* of G with that width. A tree-decomposition of an undirected graph $G = (V, E)$ is a tree $\mathcal{T}$, where each node $n \in \mathcal{T}$ is associated with a bag $X_n \subseteq V$ and these two properties hold: (1) the endpoints of each edge should occur simultaneously in at least one bag, i.e. for each edge $\{v, w\} \in E$ there is a node $n \in \mathcal{T}$ such that both $v \in X_n$ and $w \in X_n$, and (2) for each vertex $v \in V$, all nodes n for which the associated bag contains v, i.e. $v \in X_n$, are a connected subtree of $\mathcal{T}$. The width of a tree decomposition $\mathcal{T}$ is equal to $\max_{n \in \mathcal{T}} |X_n| - 1$. Although finding a tree-decomposition of the treewidth of a graph is NP-hard, there is a linear time algorithm [3] for any fixed width. Furthermore, there are algorithms that are able to efficiently find good tree decompositions in practice, e.g. [16].

One versatile approach in the design of algorithms that exploit bounded treewidth is to perform *dynamic programming* on the tree-decomposition. Central to this idea is the interpretation of every bag X_n in the tree-decomposition as a *graph separator*, which means that if we remove the nodes in the bag from the graph, the graph splits up in different parts. By moving up the the tree of the tree-decomposition, the algorithm looks at the current bag of vertices of the graph, which separates the part of the graph that is already processed by the algorithm from the part still needs to be processed, with the invariant that all connections between the processed and unprocessed parts of the graph must go through the current bag. In each state of the algorithm a state table is constructed for (combinations of) vertices in the bag associated with the current node in the tree, under the assumption that optimal decisions were made for the processed part of the graph.

A helpful way to design a dynamic programming algorithm based on the tree-decomposition is to assume it is a *nice* tree-decomposition [4]. Such a tree-decomposition has a root and as a consequence the order in which the dynamic programming algorithm visits the nodes of the tree is fixed: we start at the leaves and moves up to the root. In our description of the algorithm, we say that the algorithm moves from *parent* nodes to *child* nodes. A nice tree-decomposition distinguishes four types of nodes: *create* nodes which corresponds to leaves in the tree that only have a single vertex in their bag, *introduce* nodes which introduce a single new vertex into the bag of their parent, *forget* nodes which remove a single vertex from the bag of their parents and *join* nodes which have the same bag as their two parents. Thus in a nice tree-decomposition join nodes have two parents, leaf nodes have no parents and the other nodes have a single parent. There exists a linear time algorithm that converts any tree-decomposition into a nice tree-decomposition with $O(|V|)$ nodes and the same width [4].

Our algorithm adopts this approach. For each bag a state table is constructed for all partial covers of the lines in the current bag, based on the possible combinations of values of the left hand sides of Constraints 5. In every step we are only allowed to increase the θ_c values and thus the coverage of each line. Since each circulation $c \in E$ can only be selected an integer number of times, there is a finite number of fractions $\sum_{c \in \delta(l)} \frac{\theta_c}{k_c}$ that lie in the range $[0, 1]$ for each line l. The total number of combinations of values of the left hand sides of Constraints 5 for a particular set of lines is at most the product of the possible number of values for the individual constraints. This gives us an upper bound on the number of states we need to maintain in a state table when we enumerate the optimal partial covers for that bag. An upper bound on the number of possible fractional values for the left hand side of the constraint of a line l is denoted by ρ_l. One (crude) upper bound for this can be computed as $1 + \prod_{c \in \delta(l)} k_c$. Note that if the circulations are short enough compared to T, which they often are in practice, this number will typically be small.

A single state in the state table for a bag X_n assigns a fraction q_l to each line $l \in X_n$ where we have $q_l \in \{0, \frac{1}{\rho_l}, \ldots, \frac{\rho_l - 1}{\rho_l}, 1\}$. The state table for a node $n \in \mathcal{T}$ maps each state to the minimum number of vehicles required to reach this state. If the algorithm generates the same state multiple times, it is sufficient to store only the state for which the minimum number of vehicles was required to reach the state. If we introduce ρ as the maximum ρ_l for all lines $l \in \mathcal{L}$, the size of the table is for a single bag is $O(\rho^{w+1})$.

To conclude the description of the algorithm, we describe how the state table for each of the four types of nodes in the tree-decomposition can be computed based on the state tables of the parent(s). In a *start* node, only a single line l is introduced and thus only a fixed circulation is considered, modeled by a self loop. This loop can be used at most ρ_l times and may not be used at all. These state tables can be generated in $O(\rho)$ time. In an *introduce* node, a new line l is introduced to the state table of the parent. Due to this introduction, we have to expand all states in the parent table with all possible multiplicities of the circulations in $\delta(l)$, including multiplicities of zero. As the state table of the parent contains $O(\rho^w)$ states and there are $O(w)$ circulations that connect the new line l to lines in the current bag, each of which can be used at most ρ times in the expansion, we can construct the state table for an introduce node in $O(w \cdot \rho^{w+1})$ time. In a *forget* node, a line l is removed from the state table of parent. This means that from this step onward, that particular line is in the set of lines that have been processed by the algorithm and thus we must make sure that it is fully covered. This can be achieved by removing all states from the parent table where the q_l of this line is not equal to 1, as the removal of line l implies that these states are infeasible. This can be done in $O(\rho^{w+1})$ time as we only have to filter the state table of the parent. Finally, in a *join* node we have two parent nodes with the same bag, but potentially different states. We construct the new state table by either taking the state and its associated number of vehicles from one of the two parents' state table or by taking a state from one table and combining it with a state from the second table, by adding up the q_l's of both states and adding up the number of vehicles of the two states. These combinations are only worth considering if none of the resulting q_l's exceeds 1. As both parent tables can be of size ρ^{w+1}, there are ρ^{2w+2}' combinations that can be explored in this step. This means the table for a join node can be computed in $O(\rho^{2w+2})$ time.

When the algorithm is done, we can find the optimal solution to the VCSP in the root node at the state where all q_l's are equal to one. Recall that the size of the tree-decomposition is $O(|\mathcal{L}|)$ and each node in this decomposition can be processed in $O(\rho^{2w+2})$ time.

▶ **Theorem 9.** *The VCSP with $\kappa = 2$ and auxiliary constraints on the allowed circulations can be solved in $O(n\rho^{2w+2})$ time where $n = |\mathcal{L}|$, w is the tree-width of the circulation graph and $\rho = \max_{l \in \mathcal{L}} \prod_{x \in \{k_c | c \in \delta(l)\}} x$.*

5 Conclusions and further research

We have shown that the Vehicle Circulation Scheduling Problem is NP-hard for any finite restriction on the number of lines that can be included in a circulation (κ) greater than two. For the $\kappa = 2$ case we need to make the (realistic) additional assumption that we have auxiliary restrictions on which lines can be combined in order to prove NP-hardness. For the $\kappa = 2$ case we show that if we can cover each line by at most one unique circulation, a matching algorithm yields the optimal solution. This solution provides a $\frac{16}{15}$-approximation in case multiple circulations can be used. We also provide an exact algorithm that can exploit low treewidth of the circulation graph, and a low number of vehicles required per circulation. For future research, it makes sense to combine these algorithms with the line planning process to see if it can help to make line plans that allow better vehicle schedules. Furthermore, it is interesting to consider whether algorithms exist that are useful for cases where κ is small, but greater than two. Finally, it is still an open question whether the $\kappa = 2$ case without auxiliary restrictions is NP-hard.

References

1 Erwin JW Abbink, Luis Albino, Twan Dollevoet, Dennis Huisman, Jorge Roussado, and Ricardo L Saldanha. Solving large scale crew scheduling problems in practice. *Public Transport*, 3(2):149–164, 2011.

2 Michel Louis Balinski. Integer programming: methods, uses, computations. *Management science*, 12(3):253–313, 1965.

3 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

4 Hans L Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996.

5 Ralf Borndörfer, Martin Grötschel, and Marc E Pfetsch. A column-generation approach to line planning in public transport. *Transportation Science*, 41(1):123–132, 2007.

6 Gabrio Caimi, Leo Kroon, and Christian Liebchen. Models for railway timetable optimization: Applicability and applications in practice. *Journal of Rail Transport Planning & Management*, 6(4):285–312, 2017.

7 Ilkyoo Choi, Jaehoon Kim, and O Suil. The difference and ratio of the fractional matching number and the matching number of graphs. *Discrete Mathematics*, 339(4):1382–1386, 2016.

8 Pieter-Jan Fioole, Leo Kroon, Gábor Maróti, and Alexander Schrijver. A rolling stock circulation model for combining and splitting of passenger trains. *European Journal of Operational Research*, 174(2):1281–1297, 2006.

9 Michael R Garey and David S Johnson. Computers and intractability. a guide to the theory of NP-completeness. a series of books in the mathematical sciences, 1979.

10 Omar J Ibarra-Rojas, Fernando López-Irarragorri, and Yasmin A Rios-Solis. Multiperiod bus timetabling. *Transportation Science*, 50(3):805–822, 2015.

11 Natalia Kliewer, Bastian Amberg, and Boris Amberg. Multiple depot vehicle and crew scheduling with time windows for scheduled trips. *Public Transport*, 3(3):213–244, 2012.

12 Julius Pätzold, Alexander Schiewe, Philine Schiewe, and Anita Schöbel. Look-Ahead Approaches for Integrated Planning in Public Transportation. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 17:1–17:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/OASIcs.ATMOS.2017.17`.

13 Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.

14 Anita Schöbel. Line planning in public transportation: models and methods. *OR spectrum*, 34(3):491–510, 2012.

15 Anita Schöbel. An eigenmodel for iterative line planning, timetabling and vehicle scheduling in public transportation. *Transportation Research Part C: Emerging Technologies*, 74:348–365, 2017.

16 Hisao Tamaki. Positive-Instance Driven Dynamic Programming for Treewidth. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ESA.2017.68`.

A Simple Way to Compute the Number of Vehicles That Are Required to Operate a Periodic Timetable*

Ralf Borndörfer
Zuse Institute Berlin (ZIB)
Takustr. 7, 14195 Berlin, Germany
borndoerfer@zib.de

Marika Karbstein
Zuse Institute Berlin (ZIB)
Takustr. 7, 14195 Berlin, Germany
karbstein@zib.de

Christian Liebchen
Technical University of Applied Sciences Wildau
Hochschulring 1, 15745 Wildau, Germany
liebchen@th-wildau.de

Niels Lindner
Zuse Institute Berlin (ZIB)
Takustr. 7, 14195 Berlin, Germany
lindner@zib.de

Abstract

We consider the following planning problem in public transportation: Given a periodic timetable, how many vehicles are required to operate it?

In [9], for this sequential approach, it is proposed to first expand the periodic timetable over time, and then answer the above question by solving a flow-based aperiodic optimization problem.

In this contribution we propose to keep the compact periodic representation of the timetable and simply solve a particular perfect matching problem. For practical networks, it is very much likely that the matching problem decomposes into several connected components. Our key observation is that there is no need to change any turnaround decision for the vehicles of a line during the day, as long as the timetable stays exactly the same.

2012 ACM Subject Classification Applied computing → Transportation, Mathematics of computing → Matchings and factors, Mathematics of computing → Network flows

Keywords and phrases Vehicle Scheduling, Periodic Timetabling, Bipartite Matching

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.16

Acknowledgements The authors want to thank Isabel Beckenbach for providing insights into matching theory.

* This research was carried out in the framework of MATHEON supported by Einstein Foundation Berlin.

18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018).
Editors: Ralf Borndörfer and Sabine Storandt; Article No. 16; pp. 16:1–16:15
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

During the last decades, public transport has become one of the classic fields for applied mathematical optimization [1]. Typically, the planning process is subdivided into line planning, timetabling, vehicle scheduling etc. Timetabling, in particular computing a periodic timetable for instance for bus networks, is still attracting several teams of researchers [2, 4, 5, 10].

The design of public transportation services is pursuing several objectives, of course. One is operating efficiency, where a key performance indicator is the number of vehicles that are required for operation.

In this paper, we restrain ourselves to the classical sequential approach of planning. In particular, having fixed the line plan as well as the timetable, the next task is to compute a vehicle schedule, in particular defining the number of vehicles required to operate the given timetable. This is essentially what in [9] is denoted "the traditional approach".

In more detail, we are considering the following setting, right as in [9]:

- We restrict ourselves to periodic timetables, where we denote the common period time of all lines as T.
- For a given line plan and periodic timetable, we want to compute the number of required vehicles, i.e., evaluate a so-called LTS-plan, according to [9].

We agree that in general a vehicle schedule is aperiodic. Hence, it makes most sense for software providers such as IVU or GIRO to develop and promote highly specialized algorithms on a commercial basis.

Yet, in our paper we show that the aperiodicity of optimum vehicle schedules is just a result of aperiodic timetables. In practice, this may be due to extra peak-hour trips and/or shorter trip durations during night hours. In contrast, as long as the underlying timetable is fully periodic, we prove that one can always find a vehicle schedule with a minimal number of vehicles, even when restricting the vehicle schedule to perform the very same turnaround activities of the vehicles over the entire day. In a sense, this turns out to be a consequence of the structure of bipartite matching polytopes. So, to compute the number of vehicles that are required to operate a given periodic timetable, in contrast to the procedure that is reported in [9], actually there is no need to expand (or, roll out) the periodic timetable for the number N of periods that are needed to cover a whole planning horizon (e.g., a day), and then perform a full vehicle schedule optimization from scratch, e.g., using a flow-based model. Rather, staying with the much more compact periodic representation turns out to be absolutely sufficient. Although we are aware that in several earlier contributions, minimization of operating cost had been done pretty much in this way (e.g. [6, 8]), we were not able to detect any justification in those papers that was equivalent to the one we are proposing here.

Notice that with respect to practice, this result is not only relevant, if a timetable stays the same over the entire day. Rather, if the peak in the number of vehicles was not induced by some single trips without any periodically recurring "copies", and if the trip lengths of the lines are relatively small (e.g., at most two hours) compared to the duration of the peak traffic time for which the periodic timetable is valid (say from 2 p.m. until 7 p.m.), already then our result applies.

The paper is organized as follows: At first, we shortly recall the setting of periodic timetabling. Second, we consider the task of periodic vehicle scheduling for a given fixed periodic timetable. Our goal is to prove in Theorem 12 that there is no advantage to compute the minimum number of vehicles on an expanded aperiodic network (as it is necessary for general vehicle scheduling), given that the underlying timetable is 100% periodic. To build the bridge from the periodic model to the expanded aperiodic model, we consider an expanded (or rolled-out) periodic version as an intermediate step, serving as a theoretical benchmark. Let us emphasize that the attribute "simple" in the title of this paper refers to the result itself rather than to the contents of its proof.

2 Periodic timetabling

The basis for our timetabling model is the *periodic event scheduling problem* (PESP) from [12]. Since we are focusing on computing the number of vehicles that are required to operate a periodic timetable, we are only considering activities that are associated with vehicles. The main player is an *event-activity network* $\mathcal{N} = (G, T, \ell)$, where G is a directed graph with node set V and arc set A satisfying the following properties:

- Each node $v \in V$ is either a *departure node* or an *arrival node*, so that the set of nodes of G decomposes as $V = V_{\text{dep}} \dot\cup V_{\text{arr}}$.
- The set A of arcs is the disjoint union of a set $A_d \subseteq V_{\text{dep}} \times V_{\text{arr}}$ of *driving arcs* and a set $A_t \subseteq V_{\text{arr}} \times V_{\text{dep}}$ of *turnaround arcs*. In particular, G is a bipartite graph.
- Each departure node has exactly one outgoing arc, and arrival nodes have exactly one ingoing arc, i.e., their respective driving arcs.

The event-activity network comes with a *period time* $T \in \mathbb{N}$. Moreover, we consider for each arc $a = (v, w) \in A$ its time duration $\ell_a \in [0, \infty)$. For a driving arc $vw \in A_d$, the quantity ℓ_{vw} denotes the time required to travel along vw. Similarly, if $vw \in A_t$ is a turnaround arc, then ℓ_{vw} measures the waiting time from the arrival at v until the departure at w. We assume that $\ell_{vw} > 0$ holds for driving arcs, later we will even motivate $\ell_{vw} \in (0, T]$.

A *periodic timetable* for an event-activity network $\mathcal{N} = (G, T, \ell)$ is a vector $\pi \in [0, T)^V$ such that

$$\pi_w - \pi_v \equiv \ell_{vw} \mod T \quad \text{for all } vw \in A.$$

In the case of technical minimum turnaround times (e.g., 3 min for subways), for a network with $T = 10$ an arrival at $\pi_v = 5$ and departure at $\pi_w = 6$ could yield a value $\ell_{vw} = 11$, because the train that arrives at minute five is not ready for departure at minute six, and thus has to wait until the next departure ten minutes later. This value is larger than the period time and does not equal the positive immediate difference

$$\pi_w - \pi_v = 1 \neq 11 = \ell_{vw} > T = 10.$$

We therefore define the *periodic offset* of an arc $vw \in A$ as

$$p_{vw} := \frac{\ell_{vw} - (\pi_w - \pi_v)}{T} \in \mathbb{Z}_{\geq 0}. \tag{1}$$

An example of an event-activity network with a periodic timetable is given in Figure 1. Notice that compared to periodic timetabling, where an optimal timetable is sought, here we are using a kind of simplified notation. Since in the setting that we are investigating the timetable is the input, and thus fixed, there is no need to elaborate on any minimum time durations serving as timetabling constraints. In fact, our values ℓ_{vw} are just the well-known periodic tensions [7].

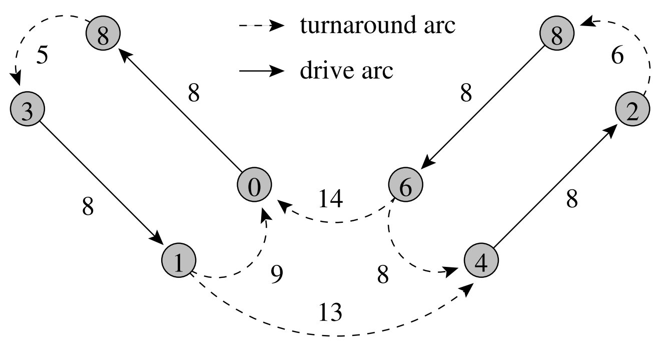


Figure 1 Example event-activity network ($T = 10$) with a periodic timetable.

3 Periodic vehicle scheduling

Let $\mathcal{N} = (G, T, \ell)$ be an event-activity network with a periodic timetable π. What is the minimal number of vehicles required to operate the timetable?

To answer this question, we define a *periodic vehicle schedule* as a collection S of directed cycles in G such that each driving arc $a \in A_d$ is contained in exactly one cycle in S. Moreover, we define the *length* resp. *periodic offset* of a directed cycle γ in G as

$$\ell(\gamma) := \sum_{a\in\gamma} \ell_a \quad \text{resp.} \quad p(\gamma) := \sum_{a\in\gamma} p_a.$$

▶ **Lemma 1.** *Let $\mathcal{N}$ be an event-activity network and let $\gamma = (v_1, \dots, v_k, v_1)$ be a directed cycle in G. If $\mathcal{N}$ admits a periodic timetable π, then $\ell(\gamma) = p(\gamma) \cdot T$ is a positive integer multiple of T.*

Proof. By definition of π and p,

$$\begin{aligned}\ell(\gamma) = \sum_{a\in\gamma} \ell_a = \ell_{v_1v_2} + \dots + \ell_{v_{k-1}v_k} + \ell_{v_kv_1}&\\ &= \pi_{v_2} - \pi_{v_1} + \dots + \pi_{v_k} - \pi_{v_{n-1}} + \pi_{v_1} - \pi_{v_k} + Tp_{v_1v_2} + \dots + Tp_{v_kv_1}\\ &= T \cdot \sum_{a\in\gamma} p_a\\ &= T \cdot p(\gamma).\end{aligned}$$ ◀

In fact, this is a special case of the well-known cycle periodicity constraints in periodic timetabling [7]. This means that a vehicle driving on a cycle γ of a periodic vehicle schedule S can periodically continue after a time of $\ell(\gamma)$. Since each driving arc has to be covered in every period, the cycle γ requires in total $\ell(\gamma)/T = p(\gamma)$ vehicles. The *number of vehicles* $n(S)$ associated to a periodic vehicle schedule S is thus

$$n(S) := \frac{1}{T}\sum_{\gamma\in S} \ell(\gamma) = \frac{1}{T}\sum_{\gamma\in S}\sum_{a\in\gamma} \ell_a = \sum_{\gamma\in S}\sum_{a\in A} p_a = \sum_{\gamma\in S} p(\gamma).$$

In other words, in any periodic schedule S we can obtain the number of required vehicles either by summing up all cycle lengths and dividing by the period time, or by counting for each cycle the "jumps" to the next period. Notice already, that later we will translate this optimal compact periodic solution to optimal solutions for both, the expanded aperiodic vehicle scheduling problem as well as the expanded periodic model as an intermediate step.

The goal is now to compute a periodic vehicle schedule S such that $n(S)$ is minimal. We call this the *minimal periodic vehicle schedule problem*. This problem has an easy reformulation as a minimum cost circulation problem, where the variables x_a indicate whether the arc a is used in the optimal vehicle schedule:

$$\begin{aligned} \text{Minimize} \quad & \sum_{a \in A} p_a x_a \\ \text{s.t.} \quad & \sum_{u\colon uv \in A} x_{uv} = \sum_{w\colon vw \in A} x_{vw}, && v \in V, \\ & x_a = 1, && a \in A_d, \\ & x_a \in \{0,1\}, && a \in A_t. \end{aligned} \tag{2}$$

▶ **Lemma 2.** *The integer program* (2) *solves the minimal periodic vehicle schedule problem.*

Proof. This follows directly from plugging in the definitions of periodic vehicle schedules and their minimal number of vehicles into the standard integer programming formulation for minimum cost circulations. ◀

A closer inspection of the IP (2) yields the following: Since all driving arcs are covered exactly once, their cost is fixed in the objective. As every driving arc $a \in A_d$ requires at least $\lfloor \frac{\ell_a}{T} \rfloor$ vehicles, we may assume w.l.o.g. that for any driving arc $a \in A_d$ holds $\ell_a \in [0, T)$, which by (1) implies $p_a \in \{0, 1\}$. In turn, we remember to add $\lfloor \frac{\ell_a}{T} \rfloor$ vehicles for each shortened driving arc a. Furthermore, as any departure (arrival) node has only one outgoing (ingoing) arc, which is a driving arc, the flow conservation conditions may be replaced by

$$\begin{aligned} & \sum_{u\colon uv \in A_t} x_{uv} = 1, && v \in V_{\text{dep}}, \\ & \sum_{w\colon vw \in A_t} x_{vw} = 1, && v \in V_{\text{arr}}. \end{aligned}$$

In the end, we arrive at the following minimum weight perfect matching problem:

$$\begin{aligned} \text{Minimize} \quad & \sum_{a \in A_t} p_a x_a + \sum_{a \in A_d} p_a \\ \text{s.t.} \quad & \sum_{a \in A_t\colon v \in a} x_a = 1, && v \in V, \\ & x_a \in \{0,1\}, && a \in A_t. \end{aligned} \tag{3}$$

In other words, we have established the following:

▶ **Lemma 3.** *Let $\mathcal{N} = (G, T, \ell)$ be an event-activity network with periodic timetable π. Let $G_t = (V, A_t)$ be the subgraph of G where all driving arcs are removed. There is a one-to-one correspondence*

$$\{\textit{perfect matchings in } G_t\} \leftrightarrow \left\{ \begin{array}{c} \textit{circulations in } G \textit{ covering} \\ \textit{all driving arcs exactly once} \end{array} \right\}.$$

Moreover, a minimum weight perfect matching w.r.t. ℓ (or p) in G_t corresponds to a minimum cost circulation w.r.t. ℓ (or p) in G.

Note that the matching formulation is of rather local nature: It suffices to compute a perfect matching for every weakly connected component of G_t. Since the turnaround arcs usually stem from turnarounds at certain stations, this means that we can compute a minimal periodic vehicle schedule by optimizing the transitions at every station. Of course, several stations might be connected by longer unloaded trips.

The following theorem summarizes the different ways to solve the minimal periodic vehicle schedule problem:

▶ **Theorem 4.** *For an event-activity network $\mathcal{N} = (G, T, \ell)$ with periodic timetable π, the number $n(S_{\min})$ of vehicles of a minimal periodic vehicle schedule is given by:*

(a) *The cost of a minimum cost circulation in G w.r.t. ℓ covering all driving arcs exactly once, divided by T.*

(b) *The sum of periodic offsets of the arcs occurring in a minimum cost circulation in G w.r.t. ℓ covering all driving arcs exactly once.*

(c) *The sum of the weights ℓ_a of a minimum weight perfect matching of the turnaround arcs in G w.r.t. ℓ plus the travel times of all driving arcs, divided by T.*

(d) *The sum of periodic offsets p_a occurring in a minimum weight perfect matching of the turnaround arcs in G w.r.t. ℓ plus the periodic offsets of all driving arcs.*

4 Periodic expansion

In this section, we describe a procedure to expand an event-activity network in a periodic way. This construction will be of use for the proof of our main result Theorem 12, the optimality proof for a periodic vehicle scheduling solution in an expanded aperiodic context.

At first, we define for any $x \in \mathbb{R}$ and $N \in \mathbb{N}$ the expression $[x]_N$ as the unique real number $y \in [0, N)$ with $x \equiv y \bmod N$. For example, $[-8]_{10} = 2$.

Let $\mathcal{N} = (G, T, \ell)$ be an event-activity network with periodic timetable π. For any positive integer N, we define another event-activity network, namely the *N-th periodic expansion* $\mathcal{N}^{(N)} = (G^{(N)}, T^{(N)}, \ell^{(N)})$ as follows:

- The node set of $G^{(N)}$ is $V^{(N)} := V \times \{0, 1, \dots, N-1\}$. A node (v, i) is called a departure (arrival) node iff v is a departure (arrival) node.
- For each driving arc $vw \in A_d$, add to the arc set $A^{(N)}$ of $G^{(N)}$ the driving arcs
$$((v, i), (w, [i + p_{vw}]_N)), \quad i = 0, \dots, N-1.$$
- For each turnaround arc $vw \in A_t$, add to $A^{(N)}$ turnaround arcs
$$((v, i), (w, j)), \quad i, j = 0, \dots, N-1.$$
- The duration of an arc $((v, i), (w, j)) \in A^{(N)}$ is set to
$$\ell^{(N)}_{(v,i),(w,j)} := \ell_{vw} + [j - i - p_{vw}]_N \cdot T.$$
- $T^{(N)} := N \cdot T$.

▶ Remark. Some observations:

(a) Up to notation, $\mathcal{N}^{(1)}$ is the same as $\mathcal{N}$.

(b) Each driving arc in $\mathcal{N}$ has N copies in $\mathcal{N}^{(N)}$, whereas each turnaround arc has N^2 copies. In fact, take a periodic turnaround arc $vw \in A_t$. For each of the N expanded occurrences of its periodic arrival event v, we keep the possibility to continue on *any* of the N copies of the respective expanded departure event w. At first sight, it could appear that some of these expanded arcs point backward in time. Yet, since $\mathcal{N}^{(N)}$ is still a periodic model, these arcs have positive durations, too, when considering their actual endpoints one period $N \cdot T$ later.

(c) Let vw be an arc in $\mathcal{N}$. Then the value of $\ell^{(N)}$ of any arc $((v,i),(w,j))$ is at least ℓ_{vw}, and the arcs $((v,i),(v,[i+p_{vw}]_N))$ for $i = 0,\dots,N-1$ are precisely the arcs whose duration is exactly ℓ_{vw}.

Periodic timetables extend in a natural way to the N-th periodic expansion:

▶ **Lemma 5.** *Let π be a periodic timetable for $\mathcal{N}$. Define $\pi^{(N)} \in [0, N\cdot T)^{V^{(N)}}$ via*

$$\pi^{(N)}_{(v,i)} := \pi_v + i \cdot T, \quad (v,i) \in V^{(N)}.$$

Then $\pi^{(N)}$ is a periodic timetable for $\mathcal{N}^{(N)}$ for the periodic tension values $\ell^{(N)}_{(v,i),(w,j)}$.

Proof. Let $((v,i),(w,j)) \in A^{(N)}$. We need to show that $\pi^{(N)}_{(w,j)} - \pi^{(N)}_{(v,i)} - \ell^{(N)}_{(v,i),(w,j)}$ is an integer multiple of $N \cdot T$. Plugging in the definitions,

$$\begin{aligned}\pi^{(N)}_{(w,j)} - \pi^{(N)}_{(v,i)} - \ell^{(N)}_{(v,i),(w,j)} &= \pi_w + j\cdot T - \pi_v - i\cdot T - \ell_{vw} - [j-i-p_{vw}]_N \cdot T \\ &= (j-i-p_{vw})\cdot T - [j-i-p_{vw}]_N \cdot T \\ &\equiv 0 \mod N\cdot T,\end{aligned}$$

as i, j, p_{vw} are all integers and π is a periodic timetable for $\mathcal{N}$. ◀

In the remainder of this section, we establish that $n(S_{\min}) = n(S^{(N)}_{\min})$, where $S_{\min}$ denotes a minimal vehicle schedule for $\mathcal{N}$, $S^{(N)}_{\min}$ a minimal vehicle schedule for the N-th periodic expansion $\mathcal{N}^{(N)}$ of $\mathcal{N}$, and $n(\cdot)$ the number of vehicles of the respective schedules. We first prove that $n(S^{(N)}_{\min}) \leq n(S_{\min})$.

▶ **Lemma 6.** *Let $\mathcal{N}$ be an event-activity network with a periodic timetable π and a periodic vehicle schedule S using $n(S)$ vehicles. For any positive integer N, the timetable $\pi^{(N)}$ on $\mathcal{N}^{(N)}$ can be operated with $n(S)$ vehicles.*

Proof. Let M be a perfect matching of the turnaround arcs in G, resulting in a periodic vehicle schedule S using $n(S)$ vehicles. Then

$$M^{(N)} := \{((v,i),(w,[i+p_{vw}]_N) \mid vw \in M,\ i = 0,\dots,N-1\}$$

is a perfect matching of the turnaround arcs in $G^{(N)}$. By the previous remark, the arcs of $M^{(N)}$ have the same turnaround time as their counterpart in M. Moreover, every driving arc in G has N copies with the same travel time in $G^{(N)}$. By Theorem 4, $M^{(N)}$ leads hence to a periodic vehicle schedule whose number of vehicles is

$$\frac{1}{N\cdot T}\left(\sum_{a\in M^{(N)}} \ell^{(N)}_a + \sum_{a\in A^{(N)}_d} \ell^{(N)}_a\right) = \frac{1}{N\cdot T}\left(N\cdot\sum_{a\in M}\ell_a + N\cdot\sum_{a\in A_d}\ell_a\right) = n(S). \quad ◀$$

▶ **Theorem 7.** *Let $\mathcal{N}$ be an event-activity network with a periodic timetable π. For any positive integer N, the number of vehicles of a minimal periodic vehicle schedule w.r.t. $\pi^{(N)}$ on $\mathcal{N}^{(N)}$ equals the number of vehicles of a minimal periodic vehicle schedule w.r.t. π on $\mathcal{N}$.*

Proof. By Lemma 6, here it remains to show that $n(S_{\min}) \leq n(S^{(N)}_{\min})$, where $S^{(N)_{\min}}$ denotes a minimal periodic vehicle schedule w.r.t. π on $\mathcal{N}^{(N)}$, and $S_{\min}$ for the initial unexpanded periodic network $\mathcal{N}$. By Theorem 4, $S^{(N)}_{\min}$ induces a perfect matching $M^{(N)}$ of the turnaround arcs, with corresponding binary variables $x^{(N)}_a$ for $a \in A^{(N)}_t$ set to 1 in the integer programming formulation (3).

We define a – possibly fractional – periodic vehicle schedule S_{frac} w.r.t. π on $\mathcal{N}$ as follows: For each turnaround arc $vw \in A_t$, set the value of its matching variable x_{vw} as

$$x_{vw} := \frac{1}{N} \cdot \sum_{i,j=0}^{N-1} x^{(N)}_{(v,i),(w,j)} \tag{4}$$

By the definition of $A^{(N)}$ and by the matching property of $M^{(N)}$, S_{frac} indeed constitutes a – possibly fractional – periodic vehicle schedule w.r.t. π on $\mathcal{N}$, i.e., a fractional perfect matching in the bipartite graph of the turnaround arcs A_t of G. By Remark 4, the travel time along any arc used by S_{frac} is at most the travel time of any of its counterparts in $S^{(N)}_{\min}$. This implies that the total cost of S^{frac} is at most $n(S^{(N)}_{\min})$:

$$\begin{aligned}
n(S_{\text{frac}}) &= \frac{1}{T}\left(\sum_{vw\in A_d} \ell_{vw} + \sum_{vw\in A_t} x_a \ell_{vw}\right)\\
&\overset{(4)}{=} \frac{1}{T}\left(\sum_{vw\in A_d} \frac{1}{N}\sum_{i=0}^{N-1} \ell_{vw} + \sum_{vw\in A_t} \frac{1}{N}\sum_{i,j=0}^{N-1} x^{(N)}_{(v,i),(w,j)}\ell_{vw}\right)\\
&\leq \frac{1}{T}\left(\frac{1}{N}\sum_{vw\in A_d}\sum_{i=0}^{N-1} \ell^{(N)}_{(v,i),(v,[i+p_{vw}]_N)} + \frac{1}{N}\sum_{vw\in A_t}\sum_{i,j=0}^{N-1} x^{(N)}_{(v,i),(w,j)}\ell^{(N)}_{(v,i)(w,j)}\right)\\
&= n(S^{(N)}_{\min}).
\end{aligned}$$

Recall several elementary results as they are collected, e.g., in the book of Schrijver [11]:

- As the subgraph (V, A_t) of G is bipartite, the constraints $x_a \geq 0$ and $\sum_{a\in\delta(v)} x_a = 1$ (i.e., the incidence matrix) already determine the perfect matching polytope [11, Theorem 18.1].
- The incidence matrix of any directed graph is totally unimodular [11, Theorem 13.9].
- For a totally unimodular matrix together with an integer right-hand-side vector, their associated polyhedron is integer [11, Theorem 5.20].

Now, due to the integrality of the perfect matching polytope (i.e., the assignment problem polytope), we find an optimal integral perfect matching M in the bipartite graph of the turnaround arcs A_t. This induces a minimal periodic vehicle schedule $S_{\min}$ w.r.t. π on $\mathcal{N}$. Since S_{frac} is a fractional solution of this perfect matching polytope, we finally find

$$n(S_{\min}) \leq n(S_{\text{frac}}) \leq n(S^{(N)}_{\min}).$$

Since Lemma 6 asserts $n(S^{(N)}_{\min}) \leq n(S_{\min})$, this finishes the proof. ◀

5 Aperiodic vehicle scheduling

The standard way to compute the minimal number of vehicles required to operate a – not necessarily periodic – timetable is to use a network flow model [3, Â§2.4]. For a periodic timetable, the first step is to *expand* (or *roll out*) the timetable for a sufficient amount of time, e.g., a day.

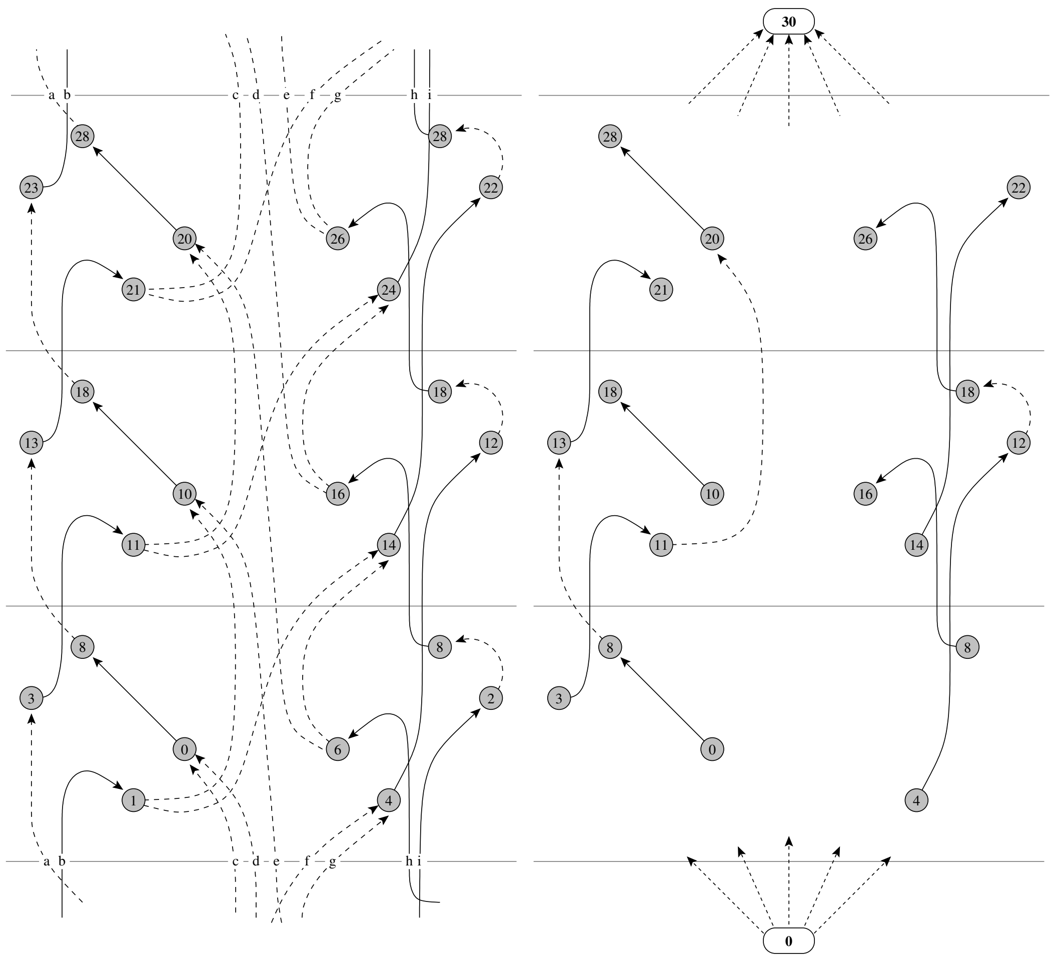


Figure 2 The first $N = 3$ layers of the periodic expansion with selected turnaround activities of the event-activity network in Figure 1 on the left, and its aperiodic counterpart on the right.

We formalize this process as follows: Starting from an event-activity network $\mathcal{N}$ with periodic timetable π, we construct the *N-th aperiodic expansion* $\mathcal{N}^{[N]} = (G^{[N]}, T^{[N]}, \ell^{[N]})$ with node set $V^{[N]}$ and arc set $A^{[N]}$ according to the following rules, see also Figure 2:

- Initialize $\mathcal{N}^{[N]}$ as the N-th periodic expansion $\mathcal{N}^{(N)}$.
- Delete all arcs $((v,i),(w,j))$ with $p^{(N)}_{(v,i),(w,j)} \geq 1$, i.e., those that leave the periodically expanded graph at time $N \cdot T$ and re-enter it at time zero.
- Remove departure nodes with out-degree zero and arrival nodes with in-degree zero, together with any incident turnaround arcs.
- Add a super-source s and arcs from s to all remaining departure nodes (v,i) with length $\ell^{[N]}_{s,(v,i)} = \pi^{(N)}_{(v,i)}$.
- Introduce a super-sink t. Add arcs from all remaining arrival nodes (w,j) to t with $\ell^{[N]}_{(w,j),t} = N \cdot T - \pi^{(N)}_{(w,j)}$.
- Finally make an extra arc (t,s) with $\ell^{[N]}_{ts} = 0$.

Deleting arcs with positive periodic offset $p^{(N)}$ means intuitively that all arcs $((v,i),(w,j))$ with $\pi_v + i \cdot T > \pi_w + j \cdot T$ ("backward in time") are omitted, as well as arcs whose duration $\ell^{(N)}_{(v,i),(w,j)}$ is at least $N \cdot T$ ("jump to the next period"). If we delete a driving arc, then we also remove the corresponding departure and arrival nodes. The arc (t, s) is the only arc in the aperiodic expansion that is allowed to go “backward in time”. Moreover, think of the deletion of a turnaround arc $((w, j), (v, i))$ as a kind of replacing it with the new pull-in arc $((w, j), t)$ together with the new pull-out arc $(s, (v, i))$.

▶ Remark.
(a) Every arc of the form $((v, i), (w, j)) \in A^{[N]}$ satisfies $p^{(N)}_{(v,i),(w,j)} = 0$ and hence $\ell^{(N)}_{(v,i),(w,j)} = \pi^{(N)}_{(w,j)} - \pi^{(N)}_{(v,i)} \in [0, N \cdot T)$.
(b) Suppose that γ is a directed cycle in $\mathcal{N}^{[N]}$ containing an arc of positive duration. Then γ contains also the arc from t to s, as $\pi^{(N)}$ increases along γ and the arc (t, s) is the only way to decrease $\pi^{(N)}$ again.

Define the sets of driving and turnaround arcs of $\mathcal{N}^{[N]}$ as $A^{[N]}_d := A^{(N)}_d \cap A^{[N]}$ and $A^{[N]}_t := A^{(N)}_t \cap A^{[N]}$, respectively. An *aperiodic vehicle schedule* is a collection $S^{[N]}$ of directed cycles in $\mathcal{N}^{[N]}$ such that each driving arc is contained in exactly one cycle of $S^{[N]}$.

By the previous remark, a vehicle starts at s, visits departure nodes and arrival nodes alternatingly until it reaches t, and finally goes back to s. The minimum number of vehicles $n(S^{[N]})$ of an aperiodic vehicle schedule $S^{[N]}$ is thus obtained by solving the following minimum cost circulation problem, see [3, Â§2.4]:

$$\begin{aligned}
&\text{Minimize} && x_{ts} && \\
&\text{s. t.} && \sum_{u:\, uv \in A^{[N]}} x_{uv} = \sum_{w:\, vw \in A^{[N]}} x_{vw}, && v \in V, \\
& && x_a = 1, && a \in A^{[N]}_d, \\
& && x_a \in \mathbb{Z}_{\geq 0} && a \in A^{[N]} \setminus A^{[N]}_d.
\end{aligned} \tag{5}$$

The *minimal aperiodic vehicle schedule* problem is to solve the above integer program, still for a given fixed timetable.

▶ **Lemma 8.** *Let $S^{[N]}$ be a minimal aperiodic vehicle schedule corresponding to an optimal solution x to the integer program* (5)*. Then the following numbers are equal:*
(a) $n(S^{[N]})$,
(b) $\displaystyle\frac{1}{N \cdot T} \sum_{a \in A^{[N]}} \ell^{[N]}_a x_a$,
(c) $\#A^{[N]}_d - \#\{a \in A^{[N]}_t \mid x_a = 1\}$,
(d) $\#A^{[N]}_d - \#M$, *where M is a maximum cardinality matching of* $(V^{[N]}, A^{[N]}_t)$,
(e) $\sum_{a=(s,v)} x_a = \sum_{a=(w,t)} x_a$.

Proof. If a feasible circulation x for (5) produces f units of flow on the t-s-arc, then it also contains f arc-disjoint paths from s to t. Let $q = (s, (v_1, i_1), \ldots, (v_k, i_k), t)$ be such an s-t-path. Then

$$\begin{aligned}
\ell^{[N]}(q) &= \ell^{[N]}_{s,(v_1,i_1)} + \sum_{j=1}^{k-1} \ell^{[N]}_{(v_j,i_j),(v_{j+1},i_{j+1})} + \ell^{[N]}_{(v_k,i_k),t} \\
&= \pi^{(N)}_{(v_1,i_1)} + \sum_{j=1}^{k-1} \left(\pi^{(N)}_{(v_{j+1},i_{j+1})} - \pi^{(N)}_{(v_j,i_j)}\right) + N \cdot T - \pi^{(N)}_{(v_k,i_k)} \qquad = N \cdot T,
\end{aligned}$$

by the definition of $\mathcal{N}^{[N]}$. In particular, $\sum_{a \in A^{[N]}} \ell^{[N]}_a x_a = f \cdot N \cdot T$. This shows (a) = (b).

Each simple cycle in a feasible circulation uses the arc from t to s, proceeds to a departure node, and then visits driving and turnaround activities alternatingly until it reaches its last driving activity, from which it goes back to t. In particular, for each such cycle γ holds

$$\#\{a \in A_d^{[N]} \mid a \in \gamma\} - \#\{a \in A_t^{[N]} \mid a \in \gamma\} = 1.$$

A minimum cost circulation decomposes into precisely $n(S^{[N]})$ such cycles, and covers each arc of $A_d^{[N]}$ precisely once. Summing over these cycles, we obtain (a) = (c).

Observe that $\{a \in A_t^{[N]} \mid x_a = 1\}$ is a matching of $(V^{[N]}, A_t^{[N]})$. Conversely, let M be any matching in $(V^{[N]}, A_t^{[N]})$. Consider the circulation consisting of the $\#A_d^{[N]}$ simple cycles $(s, (v,i), (w,j), t, s)$ for each driving arc $((v,i),(w,j)) \in A_d^{[N]}$. For each $a \in M$, connect the cycles of the driving arcs incident to a, thereby reducing the value of flow by one. This yields a circulation with value $\#A_d^{[N]} - \#M$.

Finally, (a) = (e) follows immediately from the structure of $\mathcal{N}^{[N]}$ and (5). ◀

▶ Remark. After $\mathcal{N}^{[N]}$ has been constructed, the number $n(S^{[N]})$ does neither depend on ℓ nor π. In other words, it is sufficient to look at feasible sequences of trips regardless of their actual duration.

Now, let's have a look at the cuts that are induced along the timelines $(i+1)T - \varepsilon$:

▶ **Lemma 9.** *Let $S^{[N]}$ be an aperiodic vehicle schedule with associated matching $M^{[N]}$ of $(V^{[N]}, A_t^{[N]})$. Then for any $i \in \{0, \dots, N-2\}$,*

$$n(S^{[N]}) \geq \sum_{a \in A_d} p_a + \#\{((v,i),(w,i+1)) \in M^{[N]}\}.$$

Proof. Let x be the corresponding solution to the IP (5). For small $\varepsilon > 0$, examine the flow x on all arcs at time $(i+1)T - \varepsilon$: At this point, there is one unit of flow on each driving arc departing before $(i+1)T$ and arriving at $(i+1)T$ or later. This means, there are p_a units of flow for each driving arc $a \in A_d$ in $\mathcal{N}$. Moreover, there is one unit of flow on each turnaround arc matched by $M^{[N]}$ with arrival before $(i+1)T$ and departure at $(i+1)T$ or later. In particular, this comprises turnaround arcs starting at some (v,i) and ending at some $(w,i+1)$. Finally, there is a non-negative flow on pull-in or pull-out arcs. ◀

We turn now to the comparison of periodic and aperiodic expansions:

▶ **Lemma 10.** *Let $\mathcal{N}$ be an event-activity network with periodic timetable π. Let $S^{[N]_{\min}}$ be a minimal aperiodic vehicle schedule on $\mathcal{N}^{[N]}$, and let $S^{(N)}$ be any periodic vehicle schedule on $\mathcal{N}^{(N)}$. Then $n(S^{[N]}_{\min}) \leq n(S^{(N)})$.*

Proof. Let $M^{(N)}$ be a perfect matching of the turnaround arcs in the N-th periodic expansion. By Theorem 4,

$$\begin{aligned} n(S^{(N)}) &= \sum_{a \in A_d^{(N)}} p_a^{(N)} + \sum_{a \in M^{(N)}} p_a^{(N)} \\ &\geq \#\{a \in A_d^{(N)} \mid p_a^{(N)} \geq 1\} + \#\{a \in M^{(N)} \mid p_a^{(N)} \geq 1\} \\ &= \#\{a \in A_d^{(N)} \mid p_a^{(N)} \geq 1\} + M^{(N)} - \#\{a \in M^{(N)} \mid p_a^{(N)} = 0\} \end{aligned}$$

Since $M^{(N)}$ is a perfect matching and in every directed cycle driving and (matched) turnaround arcs alternate, $\#M^{(N)} = \#A_d^{(N)}$, and we find

$$\begin{aligned} n(S^{(N)}) &\geq 2\#A_d^{(N)} - \#\{a \in A_d^{(N)} \mid p_a^{(N)} = 0\} - \#\{a \in M^{(N)} \mid p_a^{(N)} = 0\} \\ &= 2\#\{a \in A_d^{(N)} \mid p_a^{(N)} \geq 1\} + \#\{a \in A_d^{(N)} \mid p_a^{(N)} = 0\} \qquad (6) \\ &\quad - \#\{a \in M^{(N)} \mid p_a^{(N)} = 0\}. \end{aligned}$$

The intersection $M^{(N)} \cap A_t^{[N]}$ is some matching in $\mathcal{N}^{[N]}$. We will compare this with a maximum cardinality matching $M^{[N]}$ of the turnaround arcs in the N-th aperiodic expansion $\mathcal{N}^{[N]}$. The matching $M^{(N)} \cap A_t^{[N]}$ contains all arcs a from $M^{(N)}$ with $p_a^{(N)} = 0$, except those being incident to a driving arc a with $p_a^{(N)} \geq 1$. Since any such driving arc can be incident to two turnaround arcs in $M^{(N)}$, this means

$$\#M^{[N]} \geq \#M^{(N)} \cap A_t^{[N]} \geq \#\{a \in M^{(N)} \mid p_a^{(N)} = 0\} - 2\#\{a \in A_d^{(N)} \mid p_a^{(N)} \geq 1\}. \qquad (7)$$

Therefore, using (6) and (7), and then Lemma 8,

$$n(S^{(N)}) \geq \#\{a \in A_d^{(N)} \mid p_a^{(N)} = 0\} - \#M^{[N]} = \#A_d^{[N]} - \#M^{[N]} = n(S_{\min}^{[N]}). \qquad ◀$$

The following lemma is an interesting fact about the interplay of minimum-weight perfect matchings and maximum-weight matchings in the N-th periodic expansion. The proof makes use of the structure of the 2-matching polytope of a bipartite graph.

▶ **Lemma 11.** *Let $M^{(N)}$ be a minimum-weight perfect matching w.r.t. $p^{(N)}$ of the turnaround arcs in the N-th periodic expansion $\mathcal{N}^{(N)}$. Let $q := \lceil \log_2 \left(\sum_{a \in A_t} p_a + 1\right) \rceil$. If $N \geq 2^q$, then $M^{(N)}$ maximizes $\#\{a \in M \mid p_a^{(N)} = 0\}$ among all matchings of turnaround arcs in $\mathcal{N}^{(N)}$.*

Proof. Let $M^{(2)}$ be any matching of the turnaround arcs in the second periodic expansion of $\mathcal{N}$, giving rise to an incidence vector $x^{(2)} \in \{0,1\}^{A_t^{(2)}}$. Then the vector $x \in \{0,1,2\}^{A_t}$ with

$$x_{vw} := x^{(2)}_{(v,0),(w,0)} + x^{(2)}_{(v,0),(w,1)} + x^{(2)}_{(v,1),(w,0)} + x^{(2)}_{(v,1),(w,1)}, \quad vw \in A_t,$$

is a 2-matching of the turnaround arcs in $\mathcal{N}$. Since $\mathcal{N}$ is bipartite, the vertices of the 2-matching polytope correspond to matchings where each edge is taken twice [11, Theorem 31.10]. In particular, a matching maximizing the number of arcs with $p_a^{(2)} = 0$ in $\mathcal{N}^{(2)}$ can be found by considering instead a matching in $\mathcal{N}$. By construction of $\mathcal{N}^{(N)}$, any turnaround arc $a \in A_t$ produces $\max(2 - p_a, 0)$ copies in $\mathcal{N}^{(2)}$ with offset 0. We are hence interested in finding the maximum-weight matching in $\mathcal{N}$ w.r.t. the weight function $a \mapsto \max(2 - p_a, 0)$.

Repeating this process, we can analogously find for any $k \in \mathbb{N}$ the matching maximizing the number of turnaround arcs with $p_a^{(2^k)} = 0$ in $\mathcal{N}^{(2^k)}$ by computing a maximum-weight matching in $\mathcal{N}$ w.r.t. the weights $\max(2^k - p_a, 0)$, $a \in A_t$. If $2^k \geq \sum_{a \in A_t} p_a + 1$, then such a matching is automatically a perfect matching $M^{(1)}$ minimizing the periodic offsets p. Performing the construction of the proof of Lemma 6, we obtain from $M^{(1)}$ a perfect matching $M^{(2^k)}$ minimizing $p^{(2^k)}$. By Theorem 7, the weight of $M^{(1)}$ w.r.t. p equals the weight of $M^{(2^k)}$ w.r.t. $p^{(2^k)}$.

Finally let $N = 2^q + r$ for some $r \in \mathbb{N}$. Extending $M^{(1)}$ even further to a perfect matching $M^{(N)}$ in $\mathcal{N}^{(N)}$ yields in total $\sum_{a \in A_t} (2^q + r - p_a) = \sum_{a \in A_t} (2^q - p_a) + r\#A_d$ arcs with $p_a^{(N)} = 0$. If M is a matching in $\mathcal{N}^{(N)}$ maximizing $\mu := \#\{a \in M \mid p_a^{(N)} = 0\}$, then M matches at most $2r\#A_d$ vertices that do not appear in $\mathcal{N}^{(2^k)}$. As $M^{(2^k)}$ is maximum in $\mathcal{N}^{(2^k)}$, in particular $\mu - r\#A_d \leq \sum_{a \in A_t} (2^q - p_a)$, so that M has at most as many $p_a^{(N)} = 0$ arcs as $M^{(N)}$. ◀

We present now our main result, stating that rolling out and solving the minimal aperiodic vehicle schedule problem has no advantage over working on the periodic network itself:

▶ **Theorem 12.** *Let $\mathcal{N}$ be an event-activity network with periodic timetable π. Consider*

(a) *the number $n(S_{\min})$ of vehicles of a minimal periodic vehicle schedule $S_{\min}$ on $\mathcal{N}$ w.r.t. π,*

(b) *the number $n(S_{\min}^{(N)})$ of vehicles of a minimal periodic vehicle schedule $S_{\min}^{(N)}$ on the N-th periodic expansion $\mathcal{N}^{(N)}$ w.r.t. $\pi^{(N)}$, and*

(c) *the number $n(S_{\min}^{[N]})$ of vehicles of a minimal aperiodic vehicle schedule $S_{\min}^{[N]}$ on the N-th aperiodic expansion $\mathcal{N}^{[N]}$.*

Then $n(S_{\min}) = n(S_{\min}^{(N)}) \geq n(S_{\min}^{[N]})$. Moreover, $n(S_{\min}) = n(S_{\min}^{(N)}) = n(S_{\min}^{[N]})$ holds if $N \geq 2^q(2n(S_{\min})+1)$, where $q := \left\lceil \log_2 \left(\sum_{a\in A_t} p_a + 1\right)\right\rceil$.

Proof. The equality $n(S_{\min}) = n(S_{\min}^{(N)})$ has been established in Theorem 7. By Lemma 10, $n(S_{\min}^{(N)}) \geq n(S_{\min}^{[N]})$. Thus it remains to show that $n(S_{\min}^{[N]}) \geq n(S_{\min})$.

Fix a minimal aperiodic schedule $S_{\min}^{[N]}$. Let M be a minimum-weight perfect matching of the turnaround arcs in $\mathcal{N}$ w.r.t. the periodic offset p. Assume for the moment that

$$\begin{aligned} &p_a \in \{0,1\} \text{ for all } a \in A_t, \text{ and} \\ &M \text{ maximizes the number of arcs } a \text{ with } p_a = 0 \text{ among all matchings in } (V, A_t). \end{aligned} \tag{8}$$

By Lemma 8, the aperiodic schedule $S_{\min}^{[N]}$ uses at most $n(S_{\min}^{[N]}) \leq n(S_{\min})$ pull-out arcs and at most $n(S_{\min}^{[N]}) \leq n(S_{\min})$ pull-in arcs. Suppose now $N \geq 2n(S_{\min})+1$. Then, by the pigeonhole principle, we find an $i \in \{0,\dots,N-2\}$ such that no vertex (v,i) is preceded by a pull-out arc from s or followed by a pull-in arc to t.

Let $M^{[N]}$ be the matching in $(V^{[N]}, A_t^{[N]})$ corresponding to $S_{\min}^{[N]}$. By Lemma 9,

$$n(S_{\min}^{[N]}) \geq \sum_{a\in A_d} p_a + \#\{((v,i),(w,i+1)) \in M^{[N]}\}.$$

As there are neither pull-in nor pull-out arcs, all $\#A_d$ arrival vertices of the form (v,i) have to be matched by $M^{[N]}$. Moreover, each matching partner (w,j) of (v,i) has either $j = i$ or $j = i+1$ due to the assumption $p_a \in \{0,1\}$ in (8). Thus we can write

$$n(S_{\min}^{[N]}) \geq \sum_{a\in A_d} p_a + \#A_d - \{((v,i),(w,i)) \in M^{[N]}\}.$$

The set $\{((v,i),(w,i)) \in M^{[N]}\}$ yields naturally a matching in the unexpanded periodic network $\mathcal{N}$ using only turnaround arcs $a \in A_t$ with $p_a = 0$. With $\#M = \#A_d$, the assumptions (8) and Theorem 4,

$$n(S_{\min}^{[N]}) \geq \sum_{a\in A_d} p_a + \#A_d - \#\{a \in M \mid p_a = 0\} = \sum_{a\in A_d} p_a + \sum_{a\in M} p_a = n(S_{\min}).$$

Note that (8) might not be satisfied immediately. However, $\mathcal{N}$ can be replaced by its 2^q-th periodic expansion $\mathcal{N}^{(2^q)}$, where $q := \left\lceil \log_2 \left(\sum_{a\in A_t} p_a + 1\right)\right\rceil$: Then $2^q \geq p_a$ for each $a \in A_t$, so that a minimum-weight perfect matching $M^{(2^q)}$ constructed as in Lemma 6 uses only arcs a with $p_a^{(2^q)} \in \{0,1\}$. In particular, we can delete all arcs from $\mathcal{N}^{(2^q)}$ with $p_a^{(2^q)} \geq 2$, and still obtain the same perfect matching. Moreover, Lemma 11 now certifies the second assumption. In particular, for $N \geq 2^q(2n(S_{\min})+1)$, we finally obtain

$$n(S_{\min}^{[N]}) \geq n(S_{\min}^{(2^q)}) = n(S_{\min}).$$ ◀

6 Conclusion

To summarize, given that a public transportation network is to be operated with a purely periodic timetable, in order to compute the number of vehicles that are required to operate it, there is no need to expand the periodic network over time and solve a standard network flow model for vehicle scheduling. Rather, our results justify to keep the compact periodic structure and compute perfect matchings, where the graph is even likely to decompose and make the actual computation even easier. Moreover, this insight justifies that minimizing vehicle waiting time as early as in the step of optimizing the timetable itself, indeed points the timetable solution into the direction of a favorable efficient use of vehicles – right as it has already been common practice in several case studies.

References

1 Ralf Borndörfer, Martin Grötschel, and Ulrich Jäger. Planning problems in public transit. *Production Factor Mathematics*, pages 95–122, 2010. doi:10.1007/978-3-642-11248-5.

2 Ralf Borndörfer, Heide Hoppmann, and Marika Karbstein. Separation of cycle inequalities for the periodic timetabling problem. In *ESA*, volume 57 of *LIPIcs*, pages 21:1–21:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

3 Stefan Bunte and Natalia Kliewer. An overview on vehicle scheduling models. *Public Transport*, 1(4):299–317, 2009.

4 Soumya Dutta, Narayan Rangaraj, Madhu Belur, Shashank Dangayach, and Karuna Singh. Construction of periodic timetables on a suburban rail network-case study from Mumbai. In *RailLille 2017 — 7th International Conference on Railway Operations Modelling and Analysis*, 2017.

5 Marc Goerigk and Christian Liebchen. An Improved Algorithm for the Periodic Timetabling Problem. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 12:1–12:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2017.12.

6 Christian Liebchen. The first optimized railway timetable in practice. *Transportation Science*, 42(4):420–435, 2008.

7 Christian Liebchen and Leon Peeters. Integral cycle bases for cyclic timetabling. *Discrete Optimization*, 6(1):98–109, 2009.

8 Morten N. Nielsen, Bjørn Hove, and Jens Clausen. Constructing periodic timetables using MIP - a case study from DSB S-train. *International Journal of Operational Research*, 1(3):213–227, 2006.

9 Julius Pätzold, Alexander Schiewe, Philine Schiewe, and Anita Schöbel. Look-Ahead Approaches for Integrated Planning in Public Transportation. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 17:1–17:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2017.17.

10 Julius Pätzold and Anita Schöbel. A Matching Approach for Periodic Timetabling. In Marc Goerigk and Renato Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2016.1.

11 Alexander Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2003.

12 Paolo Serafini and Walter Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989.