15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS'15, September 17, 2015, Patras, Greece

Edited by

Giuseppe F. Italiano

Marie Schmidt

Editors

Giuseppe F. Italiano
University of Rome "Tor Vergata"
Rome, Italy
giuseppe.italiano@uniroma2.it

Marie Schmidt
Erasmus University Rotterdam
Rotterdam, the Netherlands
schmidt2@rsm.nl

ACM Classification 1998
F.2 Analysis of Algorithms and Problem Complexity, G.1.6 Optimization, G.2.1 Combinatorics, G.2.2 Graph Theory, G.2.3 Applications

ISBN 978-3-939897-99-6

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/978-3-939897-99-6.

Publication date
September, 2015

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.2015.i

OASIcs – OpenAccess Series in Informatics

OASIcs aims at a suitable publication venue to publish peer-reviewed collections of papers emerging from a scientific event. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 2190-6807

www.dagstuhl.de/oasics

Contents

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ATMOS'15 Best Paper Award

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 15th ATMOS workshop (ATMOS'15) was held in connection with ALGO'15, hosted by the University of Patras and its Department of Computer Engineering and Informatics in Patras, Greece, on September 17, 2015. 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, and traffic guidance. 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.

All submissions were reviewed by at least three referees and judged on originality, technical quality, and relevance to the topics of the workshop. Based on the reviews, the program committee selected eleven submissions to be presented at the workshop. In addition, Ralf Borndörfer kindly agreed to complement the program with an invited talk that was presented as a global key-note talk of ALGO'15. This volume collects the corresponding papers for ten of the submissions, as well as the short paper for the eleventh one. Together, they quite impressively demonstrate the range of applicability of algorithmic optimization to transportation problems in a wide sense.

Based on the program committee's reviews, René van Bevern, Christian Komusiewicz, and Manuel Sorge won the Best Paper Award of ATMOS'15 with their paper "Approximation algorithms for mixed, windy, and capacitated arc routing problems".

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'15, all the authors who submitted papers, Ralf Borndörfer 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, and the local organizers for hosting the workshop as part of ALGO'15. We also 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'15 in its OASIcs series.

September, 2015

Giuseppe F. Italiano
Marie Schmidt

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Program Committee

Hannah Bast	University of Freiburg, Germany
Giuseppe F. Italiano (co-chair)	University of Rome "Tor Vergata", Italy
Gilbert Laporte	HEC Montréal, Canada
Marco Laumanns	IBM Research, Switzerland
Carlo Mannino	University of Oslo, Norway
Juan A. Mesa	University of Sevilla, Spain
Matúš Mihalák	Maastricht University, the Netherlands
Matthias Müller-Hannemann	MLU Halle-Wittenberg, Germany
Karl Nachtigall	TU Dresden, Germany
Thomas Pajor	Microsoft Research, USA
Federico Perea	Polytechnic University of Valencia
Marie Schmidt (co-chair)	Erasmus University Rotterdam, the Netherlands
Dorothea Wagner	KIT, Germany

Steering Commitee

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

List of Additional Reviewers

Moritz Baum, Julian Dibbelt, Tim Nonner, Jacint Szabo, Tobias Zündorf

Local Organizing Committee

Kalliopi (Lina) Giannakopoulou, Ioannis Katsidimas, Spyros Kontogiannis, George Michalopoulos, Andreas Paraskevopoulos, Christos Zaroliagis (chair)

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Towards Realistic Pedestrian Route Planning*

Simeon Andreev[1], Julian Dibbelt[1], Martin Nöllenburg[1], Thomas Pajor[2], and Dorothea Wagner[1]

1 **Karlsruhe Institute of Technology, Germany**
simeon.andreev@student.kit.edu,
{dibbelt,noellenburg,dorothea.wagner}@kit.edu
2 **Microsoft Research, USA**
tpajor@microsoft.com

Abstract

Pedestrian routing has its specific set of challenges, which are often neglected by state-of-the-art route planners. For instance, the lack of detailed sidewalk data and the inability to traverse plazas and parks in a natural way often leads to unappealing and suboptimal routes. In this work, we first propose to augment the network by generating sidewalks based on the street geometry and adding edges for routing over plazas and squares. Using this and further information, our query algorithm seamlessly handles node-to-node queries and queries whose origin or destination is an arbitrary location on a plaza or inside a park. Our experiments show that we are able to compute appealing pedestrian routes at negligible overhead over standard routing algorithms.

1998 ACM Subject Classification G.2.2 Graph Theory, G.2.3 Applications, H.2.8 Database Applications, I.3.5 Computational Geometry and Object Modeling

Keywords and phrases pedestrian routing, realistic model, shortest paths, speed-up technique

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.1

1 Introduction

The computation of routes in street networks has received tremendous attention from the research community over the past decade, and for many applications efficient algorithms now exist; see [4] for a recent survey. The bulk of work, however, focuses on computing driving directions for cars. Other scenarios, such as computing routes for pedestrians, have been neglected or simply dismissed as a trivial matter of applying a different cost function.

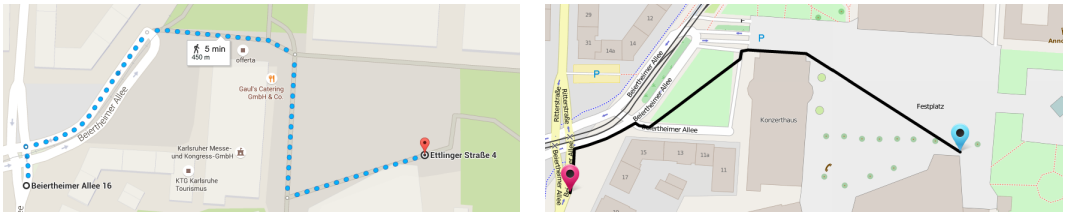
We argue that this naïve approach may lead to unnatural and suboptimal solutions. In fact, pedestrians utilize the street network quite differently from cars, which is often not captured by traditional approaches. For example to save distance, pedestrians are free to deviate from the streets, using the walkable area of public open spaces such as plazas and parks. On the other hand, crossing large avenues can be expensive (due to traffic), and it may be faster and safer to walk a small detour in order to use a nearby bridge or underpass.

In this work, we address the unique challenges that come with computing pedestrian routes. In order to obtain as realistic routes as possible, we propose to first augment the underlying street network model, and then to apply a tailored routing algorithm on top of it. After setting some basic definitions (Section 2), we propose geometric approaches for automatically adding sidewalks, calculating realistic crossing penalties for major roads, and

* Partially supported by DFG grant WA654/23-1, EU grant 609026 (MOVESMART), and Google Focused Research Award. Most of the work done while Thomas Pajor was at Karlsruhe Institute of Technology.

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt; pp. 1–15

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



(a) Google Maps. **(b)** Our approach.

Figure 1 In contrast to current approaches, our route in this example makes use of sidewalks (avoiding unnecessary street crossings), begins on a plaza and traverses it in a natural way.

preprocessing plazas and parks in order to traverse them in a natural way (Section 3). Our integrated routing algorithm seamlessly handles node-to-node queries and queries whose origin or destination is an arbitrary geographic location inside a plaza or park (Section 4). To efficiently support long-range queries, we also adapt the *Customizable Route Planning* (CRP) algorithm [9]—a well-known speed-up technique for computing driving directions in road networks—to our scenario. We evaluate our approach on OpenStreetMap data of Berlin and the state of Baden-Württemberg, Germany (Section 5). Our algorithm runs in the order of milliseconds, which is practical for interactive applications. We observe that we are able to compute pedestrian routes that are much more appealing than those by state-of-the-art route planners, such as shown in Figure 1. Section 5.2 shows further examples and an illustrated comparison of our method with three popular external services.

Related Work. We touch several subjects: sidewalk generation, traversal of open areas, and graph-based routing. See [12] for an overview and assessment of different sidewalk generation approaches. Many works consider extraction of street networks from satellite images, e. g., [15, 18]. While this approach is promising for roads, extracting sidewalks is problematic due to poor image resolution and occlusion (e. g., by trees). Moreover, satellite imagery is not as easily available as street data. In contrast, a street network analysis technique [17] generates sidewalk information directly from street layouts, but it does not handle multiple lanes and streets that are close to each other very well. An alternative technique [3] leverages building layouts to generate sidewalks, however, not all streets that have sidewalks are also adjacent to a building, resulting in incomplete output.

Traversing open areas is a classical problem in robotics and computational geometry, and numerous works exist on the subject [20]. Cell decomposition [13] yields paths that are offset from the obstacles and area boundary, and [14] combines several techniques—including Voronoi diagrams—to obtain robust collision-free robot motion paths. Visibility graphs [1] are specifically important to us, since they represent geometric shortest paths.

Given source and target nodes in a graph, Dijkstra's algorithm [11] computes shortest paths between them. A plethora [4] of work deals with accelerating Dijkstra's algorithm by using an additional offline preprocessing stage. In our work, we adapt the *Customizable Route Planning* (CRP) algorithm [9], which offers an excellent tradeoff between query performance and preprocessing effort. In essence, its preprocessing uses a nested multilevel partition of the graph to compute shortcuts between the boundary vertices in each cell. Traversing these shortcuts then enables the query to skip over large parts of the graph.

2 Preliminaries

We model the street network as a *undirected graph* $G = (V, E)$ with a set V of *nodes* and a set $E \subseteq \binom{V}{2}$ of *edges*. A node that is incident to exactly two edges is called a *2-node*. For a specific subset of edges $E' \subseteq E$ the *induced graph* $G[E'] = (V', E')$ contains E' and the nodes V', which are incident to the edges of E'. An *s–t path* in G is a node sequence $P_{s,t} = (s = v_1, \ldots, v_k = t)$, with each $e_i = \{v_i, v_{i+1}\}$ contained in E. A graph G is *planar* if a crossing-free drawing of G in the plane exists. A specific *embedding* of G maps each node to a coordinate in the plane. The embedding of G subdivides the plane into disjoint polygonal regions called *faces* bounded by the edges of G. Note that in our street networks each node $v \in V$ corresponds to a physical location. Likewise, each edge $e \in E$ represents a street segment. The *cost* of e is given by $c\colon E \to \mathbb{R}_+$, where $c(e)$ is the time (in seconds) a pedestrian requires to traverse e. This value may, e.g., depend on the street category and the segment's physical length. For source and target nodes s and t, Dijkstra's algorithm [11] computes a *shortest s–t path* $P_{s,t}$, i.e., an s–t path whose cost $\sum_{i=1}^{k-1} c(e_i)$ is minimal.

Besides the street network, we consider the *walkable area* of public open spaces such as *plazas* and *parks*. We represent them by polygons, as follows. A *(simple) polygon* $Q \subset \mathbb{R}^2$ is defined as the interior of a sequence of *vertices* $Q = (p_1, \ldots, p_n), p_i \in \mathbb{R}^2$ sorted clockwise and connected by non-self-intersecting *segments* $\overline{p_1p_2}, \overline{p_2p_3}, \ldots, \overline{p_n, p_1}$. (We distinguish the *nodes* of a graph from the *vertices* of a polygon.) A *polygon with holes* Q is defined by a *boundary cycle* b_Q and *holes* $h_Q^1, \ldots, h_Q^k$ in the interior of b_Q, where b_Q and h_Q^i again define simple polygons and their vertices are the vertices of Q. The *interior* of Q is $b_Q \setminus (\cup_i h_Q^i)$ and a point o or a segment s is *within* Q if o or s lie within this interior. The *visibility graph* $\mathrm{VG}(Q)$ of a polygon with holes Q is a geometric graph that consists of all vertices $p_1, \ldots, p_n$ of Q and all segments $\overline{p_ip_j}$ which lie within Q. The *visibility polygon* $\mathrm{VP}(p, Q)$ of a point $p \in Q$ with respect to the containing polygon Q is the region within Q that is *visible* from p, i. e., for each $q \in \mathrm{VP}(p, Q)$ the segment $\overline{pq} \subseteq Q$. With $|Q|$ we denote the number of vertices of Q.

For many geometric computations, we use functionality of the computational geometry library CGAL [22], in particular for computing line segment intersections, polygon unions and differences, visibility graphs and polygons, range queries, and point-in-polygon queries; see [8] for descriptions of the algorithms. Furthermore, we implement a custom sweep line algorithm, which, given a set of disjoint polygons and a set of query points, determines for each point the polygon that contains it (if any); see Appendix A for details. To apply these geometric algorithms to our street data, we map geographic coordinates to points in $\mathbb{R}^2$ using the Mercator projection. We use Euclidean distances $||p - q||$ between points p and q.

3 Augmented Graph Model for Pedestrian Routing

We consider three key aspects where pedestrian routes differ from those of vehicles: (a) sidewalks are preferred over streets, if present; (b) plazas can be traversed freely; (c) in parks pedestrians may walk freely on the lawn, but park walkways are preferred. In this section, we present algorithms that process the street network in order to accommodate these differences. (We then discuss queries in Section 4.) Most of this preprocessing is independent of the edge costs in the network, hence, new costs can be integrated with little effort.

3.1 Sidewalks and Street Crossings

Unlike features, such as street direction, turn restrictions and separation into lanes, sidewalk data is often lacking (or inconsistently modeled) in popular street databases, such

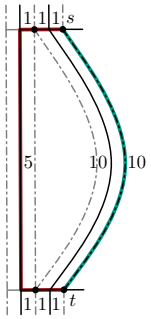


Figure 2 Shortest path when using streets (left) or sidewalks (right).

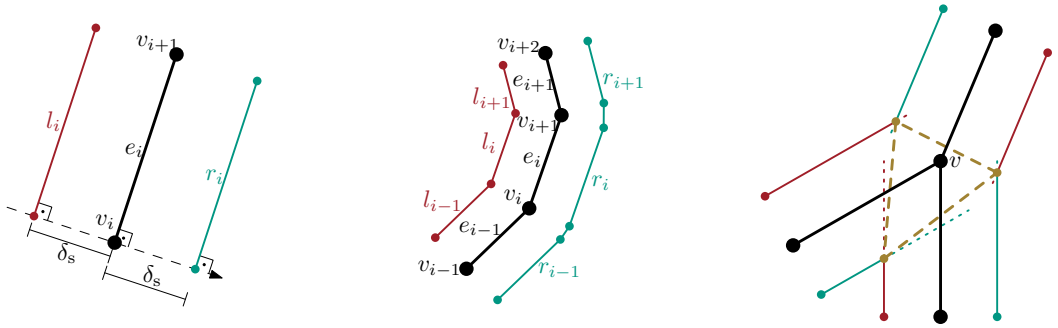
as OpenStreetMap (`openstreetmap.org`). As a result, state-of-the art pedestrian route planners mostly use the streets themselves and not their sidewalks. However, this may lead to unnecessary street crossings, which can either be costly (due to traffic lights), or even be impossible. In contrast, when sidewalks are considered properly, a seeming detour may actually be the shorter path, see Figure 2. We therefore propose to replace some streets (as given by the input) with automatically generated sidewalks. We distinguishing between three street types: *Highways* represent streets that are inaccessible for pedestrians, hence, they have no sidewalks; *regular streets*, such as city streets, have sidewalks; and *walkways* are footpaths and streets small enough to require no sidewalks.

Street Polygons. Naïvely, one could add sidewalks to the left and to the right of every regular street in the input [17]. Unfortunately, this results in sidewalks being placed in the middle of multi-lane streets or in median strips, which is clearly unwanted. We therefore propose to avoid areas enclosed by regular or highway streets that are too small or thin to hold sidewalks.

To achieve this, we first compute a set $\mathcal{S}$ of *street polygons*, representing such areas without sidewalks. Consider the embedded graph G_{hr} induced by the set of highway and regular street edges. We obtain the planarization G'_{hr} of G_{hr} using a standard sweep-line algorithm for line segment intersections [8]. Let f be a face in G'_{hr} and let a_f and p_f denote its area and perimeter, respectively. Then f is considered a street polygon, if $a_f/p_f \leq \beta_{\text{r}}$ (too thin) or $a_f \leq \beta_{\text{a}}$ (too small) for suitably chosen thresholds $\beta_{\text{r}}, \beta_{\text{a}}$.

Sidewalks. Our goal is to place sidewalks to the left and to the right of each street edge at some offset, unless they would be placed inside a street polygon. They should also follow curves and handle street intersections correctly, see Figure 3. To do so, we consider the embedded graph G_{r}, induced by the regular street edges. Recall that in G_{r} street intersections are modeled by nodes v of degree $\deg(v) \geq 3$, while the street's curvature is modeled as paths of 2-nodes. For each maximal 2-node path $(v_1, \ldots, v_k)$ and its adjacent intersection nodes v_0 and v_{k+1} (where we treat dead ends as intersection nodes, too), we consider the edge sequence $(e_0, \ldots, e_k)$, where $e_i = \{v_i, v_{i+1}\}$. For each edge e_i, we create two sidewalk edges l_i and r_i, and offset them (from e_i) by a distance δ_{s}; see Figure 3a. In order to form correct paths along bends, these edges need to be trimmed or linked via auxiliary edges, depending on the bend angles; see Figure 3b.

At each street intersection $v \in G_{\text{r}}$ with $\deg(v) \geq 3$, we sort the incident edges in cyclic order. This order yields adjacent sidewalks, which we again trim at their respective intersection points or link by an auxiliary edge; see Figure 3c. For each street edge e incident



(a) Sidewalks for single segment.

(b) Connecting sidewalk paths.

(c) Adjusting sidewalks at a street intersection.

Figure 3 Generating sidewalks. Regular street segments are replaced by two sidewalk edges (a). Subsequent pairs of sidewalk edges are then connected along each 2-node path (b). Finally, the resulting sidewalks are adjusted at street intersections (dotted parts removed), and crossing edges (dashed) are added (c).

to v, we also add an edge between the two sidewalks at v associated with e, which allows to cross e at v; see again Figure 3c.

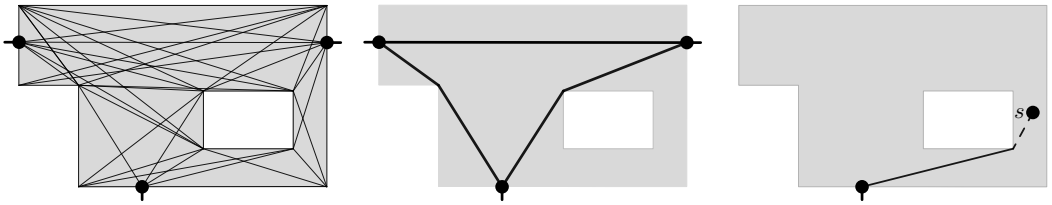
Next, we remove all sidewalk portions contained in street polygons of $\mathcal{S}$. Using a standard line segment intersection algorithm [8], we first subdivide sidewalks at the boundaries of street polygons. Then, we use our point-in-polygon algorithm (see Appendix A) to remove all sidewalk segments with both endpoints inside a polygon of $\mathcal{S}$. This results in (at most) two sidewalks per street, as opposed to two sidewalks per lane.

Finally, we assemble the *routing graph* G induced by sidewalk, crossing and walkway edges (but not highway and regular street edges). For connectivity, we add nodes at the intersections of sidewalks with walkways, subdividing the intersecting edges, again by running a line segment intersection algorithm [8].

Crossing Penalties. We may further utilize the street polygons $\mathcal{S}$ in order to penalize certain street crossings where waiting times can be expected. As the area covered by parallel street lanes is represented in $\mathcal{S}$, an edge e of G which passes through a multi-lane street also has a portion within $\mathcal{S}$, and we may penalize this portion in our cost function. We use two types of penalties. The "one-time" penalty α_e models a general waiting time, either for a pedestrian light or for traffic to clear. We add α_e to the cost of each edge that *enters* $\mathcal{S}$. More precisely, an edge $e = \{u, v\}$ in G enters $\mathcal{S}$ if u is outside $\mathcal{S}$ and the segment of e has common points with $\mathcal{S}$. The second penalty, denoted α_w, is a *penalty per unit of length* spent within $\mathcal{S}$. It reflects that wider streets generally require longer waiting times to cross. We find the edge portions of G within $\mathcal{S}$ while we remove sidewalks within $\mathcal{S}$. We use our sweep line algorithm (cf. Appendix A) to find edges starting outside $\mathcal{S}$. Such edges with portions within $\mathcal{S}$ also enter the street polygons.

3.2 Plazas

Pedestrians may traverse plazas freely. However, somewhat surprisingly, most state-of-the-art pedestrian navigation services route around such walkable areas, not through them. We propose to utilize visibility graphs to remedy this shortcoming. We assume that the street network database provides traversable plazas as a set $\mathcal{P}$ of *plaza polygons*, possibly with



(a) A visibility graph. **(b)** Shortest path visibility edges. **(c)** Route origin in plaza polygon.

Figure 4 Small polygon Q with a hole and all visibility edges (a) and the ones that are also on shortest paths (b). Routing from within Q requires all visibility edges (c).

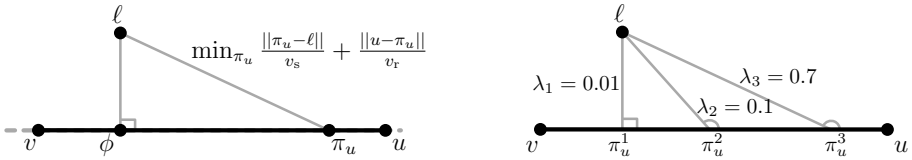
holes due to obstacles. Given $\mathcal{P}$ and the previously obtained routing graph G, we compute the *entry nodes* of each plaza: These lie on the intersection of a plaza polygon's boundary and the routing graph and are obtained by a line segment intersection algorithm [8]. We add each entry node both to the plaza polygon (as a vertex) and to the routing graph. For each polygon $Q \in \mathcal{P}$, we then compute the *visibility graph* $\mathrm{VG}(Q)$. If Q has no holes, we require quadratic time [16, 22], otherwise cubic time. (Since we encounter only very few polygons with holes in practice, we did not implement a more efficient algorithm, such as [1].) Let $E_{\mathrm{vis}}(Q)$ be the visibility edges of $\mathrm{VG}(Q)$, and $E_{\mathrm{vis}} = \cup_{Q \in \mathcal{P}} E_{\mathrm{vis}}(Q)$. We add E_{vis} as further pedestrian edges to the routing graph G.

Since the number of visibility edges $E_{\mathrm{vis}}(Q)$ of a plaza polygon $Q \in \mathcal{P}$ is generally quite high (see Figure 4a), routing through plazas can become expensive. We therefore mark the subset $E^{\mathrm{sp}}_{\mathrm{vis}} \subset E_{\mathrm{vis}}$ of visibility edges that are part of shortest paths between any pair of entry nodes (the query may then ignore unmarked edges); see Figure 4b. We do so by running Dijkstra's algorithm from each entry node, only relaxing visibility edges of the node's plaza. Note that $E^{\mathrm{sp}}_{\mathrm{vis}}$ suffices to route *across* plazas, but queries that begin or end on a plaza may still require all edges in E_{vis}; see Figure 4c and Section 4. Also note that computing $E^{\mathrm{sp}}_{\mathrm{vis}}$ requires knowledge of the routing cost function (all other preprocessing does not). However, since the necessary shortest path queries are restricted to each plaza and the number of entry nodes is typically small, this step is not costly compared to the total preprocessing effort.

3.3 Parks

Unlike plazas, parks have designated walkways, which we favor by routing on walkable park areas (such as lawn) only at the beginning or end of a route. In order to quickly locate nearby walkways during queries, we precompute the faces of a park induced by its walkways.

Similarly to plazas, we assume that the walkable area of parks is given as the set $\mathcal{L}$ of *park polygons* (possibly with holes) by the street network database. We compute the entry nodes the same way we do for plazas. We then use our algorithm from Appendix A to compute the set E_L of edges in G contained in each $L \in \mathcal{L}$ (in a single sweep). Thus, $G_L = G[E_L]$ contains exactly the park walkways within L. We add the boundary of L to G_L (as nodes and edges) and planarize G_L. We define the set of *park faces* F_L to be the faces of G_L, and $\mathcal{F} = \bigcup_{L \in \mathcal{L}} F_L$. During queries, we will use $\mathcal{F}$ for locating park walkways and routing to/from them (see Section 4).



(a) Reaching park walkways via park area.

(b) Different penalties for walking on park area.

Figure 5 Using the park area and walkway. We minimize the walking time on the park area plus that on the walkway (left). The manner in which the park area is utilized varies with v_s (right).

4 Computing Routes

We now discuss how we leverage our model from Section 3 to compute realistic pedestrian routes. We are generally interested in queries between arbitrary locations ℓ_o (origin) and ℓ_d (destination). Usually, one handles such *location-to-location queries* by first mapping the locations to their nearest nodes (or edges) of the network, and then invoking a shortest path algorithm between those. However, for locations inside plazas and parks this method would result in inaccurate routes. Instead, we propose the following approach. First, we test whether $\ell \in \{\ell_o, \ell_d\}$ is located inside a plaza or a park. In either case, we first connect ℓ to G with sensible edges and then run Dijkstra's algorithm between ℓ_o and ℓ_d on this augmented graph. If neither is the case, we just find the nearest nodes in G using a k-d tree [5], as in the classic scenario. We discuss more details next.

Plazas. To test whether the origin or destination location ℓ is on a plaza, we simply perform a point-in-polygon test [8]. Now, assume that $Q \in \mathcal{P}$ is the polygon, which contains ℓ. We compute the visibility polygon $\mathrm{VP}(\ell, Q)$ of ℓ with respect to Q by applying the recent algorithm of Bungiu et al. [7]. We also use our sweep line algorithm from Appendix A to obtain the nodes V_Q^ℓ in $\mathrm{VG}(Q) \subset G$ that are located within $\mathrm{VP}(\ell, Q)$. We then simply connect ℓ to each node $p \in V_Q^\ell$ by adding edges $\{\ell, p\}$ to the graph G.

Recall from Section 3.2 that to route across plazas, the visibility edges in $E_{\mathrm{vis}}^{\mathrm{sp}}$ suffice. Hence, we ignore edges $e \in E_{\mathrm{vis}} \setminus E_{\mathrm{vis}}^{\mathrm{sp}}$ during the query, unless $e \in \mathrm{VG}(Q)$ for the polygon Q containing ℓ, in which case it is required for correctness.

Parks. For the case that ℓ is contained in a park, we first obtain the enclosing park face f (similarly to the plaza case). We now consider two different walking speeds: the regular walking speed v_r, and another (slower) one v_s for park faces (e. g., lawn). We set $\lambda = v_s/v_r$ (with $\lambda \in (0, 1]$) as a query time parameter; values $\lambda < 1$ penalize walking on the lawn, with smaller λ values leading to higher penalization.

Taking this into account, our goal is to connect ℓ to the walkways of f, such that the total walking duration is minimized. We thereby compute the optimal path toward each edge $e = \{u, v\} \in f$ separately, as follows. Consider a point π_u on e. To reach u from ℓ via π_u, one requires total walking time $w = \frac{||\pi_u-\ell||}{v_s} + \frac{||u-\pi_u||}{v_r}$; see Figure 5a. For a given λ, the minimum walking time w^* is achieved by the *projection point* $\pi_u^* = \phi + \frac{\lambda}{1-\lambda^2} \cdot \frac{||\ell-\phi||}{||u-\phi||} \cdot (u - \phi)$, where ϕ is the perpendicular projection of ℓ on the line through e; see [2] for a derivation of this formula. As seen in Figure 5b, a small value of λ causes a perpendicular projection: walking on the lawn is costly and therefore minimized. A larger value of λ allows for a more direct, target-aimed projection, saving distance but using more of the walkable park area.

We now use the aforementioned formula to compute for each edge $e = \{u, v\} \in f$ the projection points π_u^* and π_v^*. To check whether a segments $s_u = \overline{\pi_u^* \ell}$ is walkable within the park, we test whether the point π_u^* lies within the visibility polygon $\mathrm{VP}(\ell, Q)$. If so, we add the edge $\{\ell, u\}$ with cost $\frac{||\pi_u^* - \ell||}{v_s} + \frac{||u - \pi_u^*||}{v_r}$ to G. Node v is handled analogously.

Note that since we directly connect the origin ℓ_o and destination ℓ_d to the edges of their enclosing faces, we are unable to route around obstacles in parks. Moreover, we are unable to walk across other park faces (except the ones containing the origin and destination locations). However, this may result in unnatural routes, if origin and destination are in the same park separated by a thin face; see Figure 6. We solve this issue by introducing a radius parameter ϵ, and additionally compute edges to the boundaries of all faces (of the same park) that have vertices within distance ϵ of ℓ. We use range queries [22] to obtain those faces. If, both, origin ℓ_o and destination ℓ_d are in the same park and within distance ϵ, we additionally consider the direct route $\overline{\ell_o \ell_d}$ with cost $\frac{||\ell_d - \ell_o||}{v_s}$ explicitly.

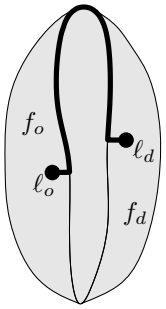


Figure 6 Detour due to long thin face.

Customizable Route Planning. Typical pedestrian routes are very short, thus, one might argue that Dijkstra's algorithm computes them sufficiently fast. Still, a practical routing engine should be robust against long-distance queries as well. We therefore propose to make use of the *Customizable Route Planning* (CRP) algorithm [9]. It is a state-of-the-art speedup technique, developed for computing driving directions in road networks. CRP employs three phases: The *preprocessing phase* uses a nested multilevel partition to compute (for each level) a metric-independent overlay graph over the boundary nodes of the partition. The *customization phase* takes a cost function as input and computes the actual edge weights of the overlay graph. Finally, the *query phase* runs bidirectional Dijkstra's algorithm, using the overlay graph to the effect of "skipping" over large parts of the network. See [9] for details.

Adapting CRP to our scenario requires little effort. We use the routing graph G for computing both the multilevel partition and the overlay graph. To easily support queries beginning or ending within parks or plazas, we enforce that nodes within the same park or plaza are never put into different cells of the partition. (We do this by running the partitioner on a slightly modified graph, in which we contract all nodes associated with the same park or plaza.) To see why this is correct, recall that the temporary edges added by the query only point to nodes within the park or plaza which contains the origin (or destination) location. By construction these nodes are all part of the same cell (on every level of the partition), therefore, the distances in the overlay graph are unaffected and still correct.

Note that in our CRP query we do not bother ignoring visibility edges in G that are not on shortest paths: They are only present on the bottom level, therefore, the query skips over them automatically in most cases.

5 Experiments

We implemented all algorithms in C++ using g++ 4.8.3 (flag -O3) and CGAL 4.6. We conducted our experiments on a single core of a 4-core Intel Xeon E5-1630v3 CPU clocked at 3.7 GHz with 128 GiB of DDR4-2133 RAM. Our data set was extracted from OpenStreetMap (OSM) on May 15, 2015, and includes roads, plazas and parks.[1] We use two

[1] Note that OSM offers a tag for indicating availability of sidewalks at streets, however, it has not been widely adopted as of now, cf. `http://taginfo.openstreetmap.org/keys/?key=sidewalk`.

Table 1 Size figures before and after preprocessing. Besides graph size, we report the total number of vertices for plaza, park, and obstacle polygons. Preprocessing time is given in [m:s].

	OSM Input					Pedestrian Output				
	Nodes	Edges	Plaza	Park	Obst.	Nodes	Edges	Plaza	Park	Time
BE	378 298	890 682	9 727	33 072	1 116	452 586	1 132 928	7 276	19 903	1:26
BW	8 235 762	17 740 940	74 547	86 380	4 439	10 209 641	22 750 644	63 300	43 632	32:45

Table 2 Detailed preprocessing figures. Besides running time, we report the number of added sidewalks, substituted streets, avg. vertices per plaza polygon (Plaza avg.), visibility edges (Vis.) and the fraction of them on shortest paths (SP. [%]), avg. vertices per park (Park avg.), avg. faces per park (Faces/park), avg. vertices per park face (Face avg.), and the vertices of all park faces (Faces total).

	Sidewalks			Plazas				Parks				
	Added sidewalks	Subst. streets	Time [s]	Plaza avg.	Vis. total	SP. [%]	Time [s]	Park avg.	Faces/ park	Face avg.	Faces total	Time [s]
BE	266 336	105 146	32.6	15.38	86 912	9.5	23.0	22.4	10.4	10.68	98 108	31.6
BW	5 580 842	1 824 185	743.7	17.45	772 416	7.7	563.6	20.8	6.6	11.26	155 840	657.4

instances: Berlin (BE) and the state of Baden-Württemberg (BW), both in Germany. While BE is an eclectic city with plenty of large streets, parks and plazas (making it interesting for evaluating pedestrian routes), we use BW to demonstrate the scalability of our approach.

This section first presents a quantitative evaluation of our approach before it compares the quality of our routes to the state of the art in a case study.

5.1 Quantitative Evaluation

We determined sensible values for the parameters of our preprocessing (cf. Section 3) by running preliminary experiments. We set the sidewalk offset to $\delta_s = 3\,\mathrm{m}$, and set values for sidewalks suppression of small and thin street polygons to $\beta_a = 1000\,\mathrm{m}^2$ and $\beta_r = 3.17\,\mathrm{m}$. For queries we assume a regular walking speed of $v_r = 1.4\,\frac{\mathrm{m}}{\mathrm{s}}$ [6], and we set $v_s = 0.9\,\frac{\mathrm{m}}{\mathrm{s}}$ for walkable park areas, i. e., $\lambda \approx 0.6$. We also set the park face expansion value to $\epsilon = 20\,\mathrm{m}$. Regarding intersections, we set the crossing penalties to $\alpha_e = 10\,s$ and $\alpha_w = 1\,\frac{\mathrm{s}}{\mathrm{m}}$, which leads to about 30 s of expected waiting time for typically-sized intersections.

Note that though we set these parameter values uniformly for our experiments, the approach would easily allow setting specific values per intersection or park face, if such detailed data was available. Also note that in our instances we do not add crossing edges within street polygons, i. e., at large multi-lane intersections (cf. Section 3). In fact, OpenStreetMap provides these already, and adding further crossings may result in dangerous paths, forcing the pedestrian to cross several lanes without the aid of traffic regulations.

Preprocessing. Table 1 presents size figures for the input and output of our preprocessing. Note that BW is significantly larger than BE (factor of 20 in graph size and factor of 9 in plaza polygons). This is reflected by the preprocessing effort, which takes about 23 times longer on BW. However, the graph size increases by less than 30 % (nodes and edges) by our preprocessing. Unfortunately, polygons representing walkable areas (parks and plazas) in

Table 3 Evaluating the query performance of our approach. We distinguish each combination of the origin/destination being on a street node (s), plaza polygon (p), or park face (f). We report the time in milliseconds to check for each of these cases (Localization), the time for our initialization stage (Initialization) or not applicable (—), and the time for running Dijkstra's algorithm (Dij.).

	BE						BW					
	Localization			Initialization		Dij.	Localization			Initialization		Dij.
Query	Plaza	Park	Street	Plaza	Park	[ms]	Plaza	Park	Street	Plaza	Park	[ms]
s–s	0.021	0.027	0.004	—	—	31.8	0.033	0.040	0.005	—	—	808.0
s–p	0.021	0.016	0.002	0.165	—	30.4	0.032	0.020	0.003	0.173	—	871.6
p–p	0.016	—	—	0.264	—	27.9	0.027	—	—	0.351	—	889.4
s–f	0.021	0.022	0.002	—	0.359	28.3	0.029	0.026	0.002	—	0.310	758.7
p–f	0.017	0.011	—	0.145	0.362	30.6	0.027	0.014	—	0.178	0.303	810.1
f–f	0.020	0.021	—	—	0.733	27.6	0.029	0.027	—	—	0.622	733.6

OSM may overlap and, moreover, polygons with holes are not supported. Instead, obstacles are represented as an additional type of polygon. We therefore first compute the union of overlapping polygons and then subtract potential obstacles from it [22]. This explains the (somewhat peculiar) drop of 50 % in the number of park polygon vertices in our output. Note that only less than 3 % of the resulting plaza polygons have holes in them (not reported in the table).

Table 2 presents more detailed figures. We observe that each part of our preprocessing requires a similar amount of time. Regarding sidewalks, only a small subset (12 %) of the roads is actually substituted. (Recall that we replace neither highways nor walkways.) However, the number of sidewalk edges per substituted road segment is more than two on average, due to complex intersections and other effects (cf. Section 3). Regarding plazas, we observe that the number of visibility edges is only a small fraction of the graph (less than 10 %), with less than 10 % of those actually being on shortest paths. The necessary shortest path computations take less than 3 seconds on BW (not reported in the table). For parks, we observe that including walkways (to compute park faces) increases the number of park vertices by a factor of 5 ("Faces total" in the table). While this results in a high average number of vertices per entire park (111 for BE), the number of vertices per park face remains small, which is the influential performance figure for queries that begin or end in a park.

Queries. We now evaluate the query performance. Recall that our query algorithm takes as input two arbitrary locations, which may be inside a plaza or park, and in which case the query will route from the precise location to the vertices of its surrounding polygon. Table 3 separately evaluates our algorithm for each scenario of placing the origin or destination on a street node (s), inside a plaza (p), or a park face (f). Per scenario, we generated 1,000 queries, choosing origin and destination (i.e., node, plaza polygon or park face) uniformly at random. For the plaza or park case, we further chose an interior point at random.

The query is oblivious to the specific scenario, i.e., we only pass geographic locations as input, and it needs to perform the necessary checks to figure out the right scenario itself. However, at below 80 µs these checks (including the determination of the specific street node or enclosing polygon) take negligible time. The initialization stage for plazas (computing additional visibility edges) or parks (computing and testing projections) is considerably more expensive, but still runs well below a millisecond, orders of magnitude faster than

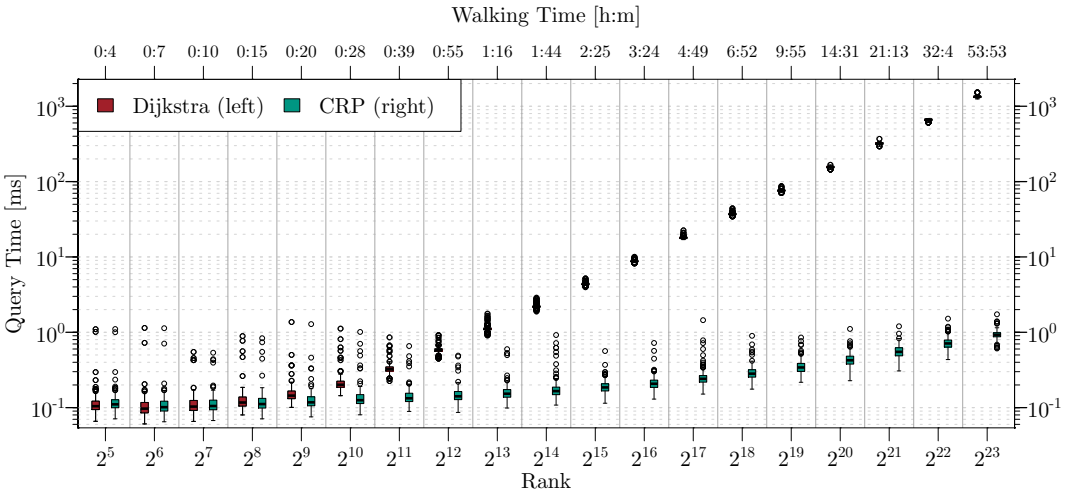
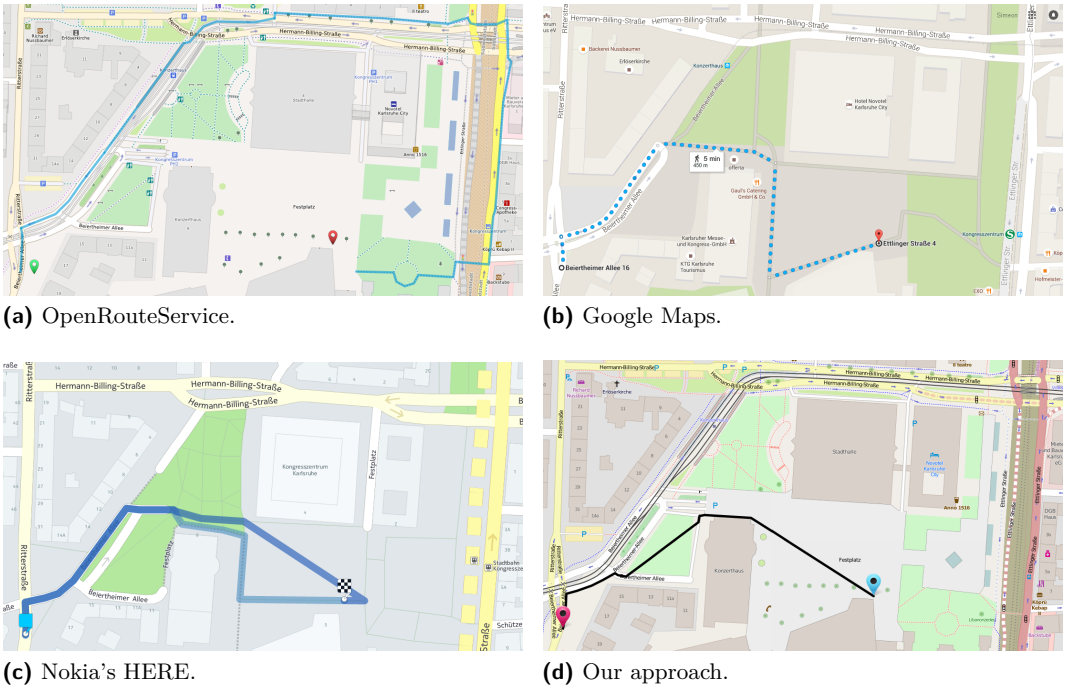


Figure 7 Dijkstra rank plot on our BW instance, comparing the performance of Dijkstra's algorithm with CRP. The top axis shows the average walking time for the queries in each bucket.

the subsequent run of Dijkstra's algorithm. Note that in our implementation we never add any edges explicitly (cf. Section 4), but rather simply initialize Dijkstra's algorithm with all vertices (and their respective distances) to which these temporary edges would point.

Customizable Route Planning. We finally evaluate the combination of our query algorithm with the Customizable Route Planning (CRP) approach [9] on our larger BW network. We use PUNCH [10] for partitioning. Our partition has five nested levels with at most $[2^8, 2^{11}, 2^{14}, 2^{17}, 2^{20}]$ vertices per cell. (This is the same configuration as in [9].) We compute the partition on the routing graph (that is output by our preprocessing), however, we temporarily replace nodes of the same plaza or park by a single supernode. This keeps polygons from spreading over cell boundaries and simplifies the CRP query. Computing the metric-independent partition takes several minutes and the subsequent customization phase takes about five seconds. Note that to integrate a new cost function, e.g., due to different crossing penalties, only the customization phase has to be rerun, which is very fast.

Figure 7 compares the performance of CRP with Dijkstra's algorithm using the *Dijkstra rank* methodology [19]: When running Dijkstra's algorithm from node s, node u has *rank* x, if it is the x-th node taken from the priority queue. By selecting random origin and destination pairs according to ranks $2^1, 2^2, \ldots, 2^{\lfloor \log |V_r| \rfloor}$ (we select 1,000 queries per bucket), the plot simultaneously captures short- mid- and long-range queries. We observe that for short-range queries the performance of Dijkstra's algorithm is very similar to that of CRP (below 200 µs on average). However, from rank 2^{10} onward, Dijkstra's algorithm becomes significantly slower (rising to more than a second), while the average running time of CRP remains below 1 ms at any rank. Note that while most pedestrian queries are likely of short range, a production system must nevertheless be robust against any type of query.



(a) OpenRouteService. **(b)** Google Maps.

(c) Nokia's HERE. **(d)** Our approach.

Figure 8 Comparison with several readily available pedestrian route planning services. The origin of the route is a street address, and its destination is inside a plaza.

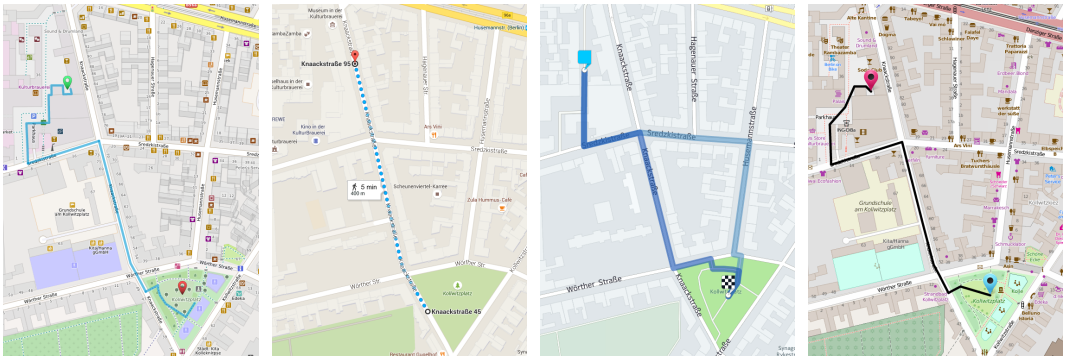
5.2 Case Study

We now present a case study, which compares the output of our approach to OpenRouteService[2], Google Maps[3] and Nokia HERE[4].

Figure 8 shows an example in the city of Karlsruhe, Germany. It highlights the importance of, both, the presence of sidewalks and being able to route across walkable areas. Clearly, OpenRouteService has the worst result, as it does not consider the boundary of the plaza (Festplatz) for routing, which results in a large detour. Because of improper sidewalk data, the routes of Google Maps and HERE suggest to go across the same street (Beiertheimer Allee) twice, which is unnatural and unnecessary. While Nokia HERE is the only competing approach that has some additional edges for walking across open areas (thus yielding a more realistic route), the utilization of these edges seems to be heuristic, still yielding an (unnatural) detour. In contrast, our route has no unnecessary street crossings (because of our generated sidewalk data), and the plaza is traversed in a natural way.

Figure 9 shows an example of a route starting on a plaza between buildings and ending in a park (Berlin, Germany). Unlike the previous example, OpenRouteService is able to route around (but not across) the plaza, because the plaza's boundary has been tagged as walkable. On the other hand, GoogleMaps seems to lack information in that region and so maps the query locations to the nearest street network node (which is actually blocked by the building structure). HERE has walkways on the plaza, but also uses a shortcut which

[2] http://www.openrouteservice.org/
[3] https://maps.google.de/
[4] https://www.here.com/



(a) OpenRouteService. **(b)** Google Maps. **(c)** Nokia's HERE. **(d)** Our approach.

Figure 9 Comparison with several readily available pedestrian route planning services. The route begins in a plaza and ends in a park.

passes through a cinema; it yields a shorter path but is obscure and unlikely: guiding the pedestrian to a door is puzzling, the building may be closed, etc. As before, the route of our approach traverses the plaza without detours.

Towards the destination, the routes of OpenRouteService, GoogleMaps and HERE are all incomplete: they find the nearest node and simply use it as the query target. In contrast, our approach allows walking directly across the lawn and avoids the small detours introduced by the other approaches.

6 Conclusion

In this paper, we presented an approach for quickly computing realistic pedestrian routes. We proposed geometric algorithms to automatically augment the street network with sensible sidewalks and edges in plazas, making it possible to walk across them in a natural way. Our query algorithm extends classic node-to-node queries by allowing the origin or destination to be an arbitrary location inside a park or plaza. We also combined our algorithm with the well-known Customizable Route Planning technique, which enabled us to compute appealing pedestrian routes within milliseconds, fast enough for interactive applications.

Future work includes more realistic models (e.g., for traffic lights or more precise human walking behavior); leveraging of building layouts [3]; and additional optimization criteria like elevation and stairs, which have been used in the context of bicycle routing [21]. We would also be interested in using our sidewalk generation algorithm in a semi-automatic tool for adding sidewalk data back to OpenStreetMap.

References

1 H. Alt and E. Welzl. Visibility Graphs and Obstacle-avoiding Shortest Paths. *Zeitschrift für Operations Research*, 32(3-4):145–164, 1988.

2 Simeon Danailov Andreev. Realistic Pedestrian Routing. Bachelor thesis, Karlsruhe Institute of Technology, November 2012.

3 Miquel Ginard Ballester, Maurici Ruiz Pérez, and John Stuiver. Automatic Pedestrian Network Generation. In *Proceedings 14th AGILE International Conference on GIS*, pages 1–13, 2011.

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. Technical Report abs/1504.05140, ArXiv e-prints, 2015.

5 Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, 18(9):509–517, September 1975.

6 Raymond C. Browning, Emily A. Baker, Jessica A. Herron, and Rodger Kram. Effects of Obesity and Sex on the Energetic Cost and Preferred Speed of Walking. *Journal of Applied Physiology*, 100(2):390–398, 2006.

7 Francisc Bungiu, Michael Hemmer, John Hershberger, Kan Huang, and Alexander Kröller. Efficient Computation of Visibility Polygons. *CoRR*, abs/1403.3905, 2014.

8 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

9 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable Route Planning in Road Networks. *Transportation Science*, 2015.

10 Daniel Delling, Andrew V. Goldberg, Ilya Razenshteyn, and Renato F. Werneck. Graph Partitioning with Natural Cuts. In *25th International Parallel and Distributed Processing Symposium (IPDPS'11)*, pages 1135–1146. IEEE Computer Society, 2011.

11 Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

12 Hassan A. Karimi and Piyawan Kasemsuppakorn. Pedestrian Network Map Generation Approaches and Recommendation. *International Journal of Geographical Information Science*, 27(5):947–962, 2013.

13 Jean-Claude Latombe. *Robot Motion Planning*, volume 124 of *Springer International Series in Engineering and Computer Science*. Springer, 1991.

14 Ellips Masehian and M. R. Amin-Naseri. A Voronoi Diagram-visibility Graph-potential Field Compound Algorithm for Robot Path Planning. *J. Robotic Systems*, 21(6):275–300, 2004.

15 M. Mokhtarzade and M.J. Valadan Zoej. Road Detection from High-Resolution Satellite Images Using Artificial Neural Networks. *International Journal of Applied Earth Observation and Geoinformation*, 9(1):32–40, 2007.

16 M. H. Overmars and Emo Welzl. New Methods for Computing Visibility Graphs. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 164–171, 1988.

17 Scott Parker and Ellen Vanderslice. Pedestrian Network Analysis. In *Walk 21 IV*, Portland, OR, 2003.

18 Ting Peng, Ian H. Jermyn, Veronique Prinet, and Josiane Zerubia. Extended Phase Field Higher-Order Active Contour Models for Networks. *International Journal of Computer Vision*, 88(1):111–128, 2010.

19 Peter Sanders and Dominik Schultes. Highway Hierarchies Hasten Exact Shortest Path Queries. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA'05)*, volume 3669 of *Lecture Notes in Computer Science*, pages 568–579. Springer, 2005.

20 J.T. Schwartz and M. Sharir. A Survey of Motion Planning and Related Geometric Algorithms. *Artificial Intelligence*, 37(1—3):157–169, 1988.

21 Sabine Storandt. Route Planning for Bicycles – Exact Constrained Shortest Paths Made Practical Via Contraction Hierarchy . In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*, pages 234–242, 2012.

22 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.6 edition, 2015.

A Batched Point in Polygon Tests

While we could use the computational geometry library CGAL [22] for most of the required geometric computations, we implemented our own sweep line algorithm for a batched point in polygon test. Given a set of disjoint polygons (without holes) and a set of query points, our algorithm determines for each point the polygon that contains it (if any). The algorithm works by sweeping the query points and the end points of the polygon segments, which define our event points, from left to right, maintaining a self-balancing binary tree of the segments which intersect the current (vertical) sweep line. Whenever the current event point is a query point $o = (x, y)$, we find the two segments s_a and s_b, which lie vertically directly above and below o. If s_a and s_b belong to a polygon $Q = (p_1, \ldots, p_n)$ with leftmost vertex p_1, we check whether $s_a = \overline{p_i p_{i+1}}$, $s_b = \overline{p_j p_{j+1}}$, and $j > i$. If so, o lies within Q, otherwise it is not contained in any polygon. If m is the number of polygon edges and n is the number of points and polygon vertices then this algorithm requires $O(n \log m)$ time. For a set of polygons with holes we may use the same approach once for the boundaries and once for the holes.

Speedups for Multi-Criteria Urban Bicycle Routing

Jan Hrncir, Pavol Zilecky, Qing Song, and Michal Jakob

Agent Technology Center, Department of Computer Science
Czech Technical University in Prague, Karlovo namesti 13, Czech Republic
{hrncir,zilecky,song,jakob}@agents.fel.cvut.cz

Abstract

Increasing the adoption of cycling is crucial for achieving more sustainable urban mobility. Navigating larger cities on a bike is, however, often challenging due to cities' fragmented cycling infrastructure and/or complex terrain topology. Cyclists would thus benefit from intelligent route planning that would help them discover routes that best suit their transport needs and preferences. Because of the many factors cyclists consider in deciding their routes, employing multi-criteria route search is vital for properly accounting for cyclists' route-choice criteria. Direct application of optimal multi-criteria route search algorithms is, however, not feasible due to their prohibitive computational complexity. In this paper, we therefore propose several heuristics for speeding up multi-criteria route search. We evaluate our method on a real-world cycleway network and show that speedups of up to four orders of magnitude over the standard multi-criteria label-setting algorithm are possible with a reasonable loss of solution quality. Our results make it possible to practically deploy bicycle route planners capable of producing high-quality route suggestions respecting multiple real-world route-choice criteria.

1998 ACM Subject Classification G.2.2 Graph Theory – Graph algorithms

Keywords and phrases bicycle routing, multi-criteria shortest path, heuristic speedups

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.16

1 Introduction

Utility cycling, i.e., using the bicycle as a mode of transport, is the original and the most common type of cycling in the world [13]. Cycling provides a convenient and affordable form of transport for most segments of the population. It has a range of health, environmental, economical, and societal benefits and is therefore promoted as a modern, sustainable mode of transport [10, 16].

In contrast to car drivers, cyclists consider a significantly broader range of factors while deciding on their routes. By employing questionnaires and GPS tracking, researchers have found that besides travel time and distance, cyclists are sensitive to slope, turn frequency, junction control, noise, pollution, scenery, and traffic volumes [3, 28]. Moreover, the relative importance of these factors varies among cyclists and can also be affected by weather conditions and the purpose of the trip [3].

Finding routes that properly take all the above factors into account is no easy task, particularly when cycling in complex urban environments. Consequently, cyclists would benefit from intelligent route planning software to help them discover routes that best suite their transport needs and preferences. Such route planners would be particularly useful for inexperienced cyclists with limited knowledge of their surroundings but they would also benefit experienced riders who want to fine-tune their routes [11], in effect making cycling a more attractive and accessible transport option.

The vast majority of existing approaches to bicycle routing, however, do not use multi-criteria search methods and they thus cannot properly account for cyclists' multiple route-

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt; pp. 16–28

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

choice criteria. A recent exception is [24] where the authors applied optimal multi-criteria shortest path algorithms for multi-criteria bicycle routing. Unfortunately, the proposed algorithm is slow on realistic problem instances and cannot be used for interactive route planning.

In this paper, we overcome this limitation and present the first bicycle routing algorithm that properly considers multiple realistic cyclists' route-choice criteria yet is fast enough for interactive use. Our algorithm extends the well-known multi-criteria label-setting algorithm [19] with several speedup heuristics in order to generate, in a much shorter time, routes that closely approximate the full set of Pareto optimal routes. In contrast to the majority of existing work, our algorithm employs a formulation of the multi-criteria bicycle routing problem that incorporates realistic route choice factors based on recent studies of cyclists' behaviour [3, 28]. We thoroughly evaluate our algorithm in terms of the speed and quality of suggested routes on a diverse set of real-word urban areas.

2 Related Work

In contrast to car and public transport route planning, for which advanced algorithms and mature software implementations exist [1], bicycle route planning is a relatively underexplored topic. Furthermore, despite the highly multi-criterial nature of cyclists' route-choice preferences, almost all existing approaches to bicycle routing do not use multi-criteria search methods to properly account for such a multi-criteriality. This contrasts with other categories of routing problems for which the application of multi-criteria shortest path search techniques has been widely studied (multimodal routing [2, 7], train routing [20], and car routing [8]). Instead, existing bicycle routing approaches transform multi-criteria search to single-criterion search either by optimising each criteria separately [14, 26] or by using a weighted combination of all criteria [15, 27]. Unfortunately, the scalarisation of multi-criteria problems using a linear combination of criteria may miss many Pareto optimal routes [4, 6] and consequently reduce the quality and relevance of suggested routes. Scalarisation also requires the user to weight the importance of individual route criteria a priori, which is difficult for most users.

Avoiding scalarisation, [23] thus showed how to effectively search for a best compromise solution for a biobjective shortest path problem in the context of bicycle routing. Recently, [24] explored the use of optimal multi-criteria shortest path algorithms for multi-criteria bicycle routing; however, the proposed algorithm is too slow for interactive route planning.

As far as general multi-criteria shortest path algorithms are concerned, the first optimal, multi-criteria label-setting (MLS) algorithm [19] extended Dijkstra's algorithm by operating on labels that have multiple cost values. A minimum label from the priority queue is processed in every iteration. On the contrary, the multi-criteria label-correcting (MLC) algorithm [5, 9] processes the whole bag of nondominated labels associated with a current node at once. Recently, heuristic accelerations of the MLS and MLC algorithms have attracted considerable attention, aiming at finding a set of routes that is similar to the optimal Pareto solution. In [7], the authors developed several heuristics to weaken the domination rules during the search. In [22], the authors proposed a near admissible multi-criteria search algorithm to approximate the optimal set of Pareto routes in a state space graph by using the ϵ-dominance approach. An alternative approach is represented by multi-criteria extensions [18, 25] of the standard A* algorithm, the latter of which was recently shown [17] to achieve an order of magnitude speedup for bicriteria road routing.

3 Multi-Criteria Bicycle Routing Problem

We represent the cycleway network as a weighted directed connected *cycleway graph* $G = (V, E, \overrightarrow{c})$, where V is the set of nodes representing start and end points (i.e., cycleway junctions) of cycleway segments and $E \subseteq \{(u,v)|(u,v \in V) \wedge (u \neq v)\}$ is the set of edges representing cycleway segments. The cycleway graph is directed due to the fact that some cycleway segments in the map are one-way only. The cost of each edge is represented as a k-dimensional vector of criteria $\overrightarrow{c} = (c_1, c_2, \ldots, c_k)$. The non-negative cost value c_i of i-th criterion for the given edge $(u,v) \in E$ is computed by the cost function $c_i : E \to \mathbb{R}_0^+$. The *multi-criteria bicycle routing problem* is then defined as a triple $C = (G, o, d)$:

- $G = (V, E, \overrightarrow{c})$ is the *cycleway graph*
- $o \in V$ is the route origin
- $d \in V$ is the route destination

A route π, i.e., a finite path π with a length $|\pi| = n$ from the origin o to the destination d in the cycleway graph G has an additive cost value

$$\overrightarrow{c}(\pi) = \left(\sum_{j=1}^{|\pi|} c_1(u_j, v_j), \ldots, \sum_{j=1}^{|\pi|} c_k(u_j, v_j) \right)$$

The solution of the multi-criteria bicycle routing problem is a full Pareto set of routes $\Pi \subseteq \{\pi|\pi = ((u_1, v_1), \ldots, (u_n, v_n))\}$ non-dominated by any other solution (a solution π_p dominates another solution π_q iff $c_i(\pi_p) \leq c_i(\pi_q)$, for all $1 \leq i \leq k$, and $c_j(\pi_p) < c_j(\pi_q)$, for at least one j, $1 \leq j \leq k$).

Based on the studies of real-word cycle route choice behaviour [28, 3], we further consider a tri-criteria bicycle routing problem. The formulation of the problem is a compact version of the earlier formulation proposed in [24] and considers the following three route-choice criteria:

Travel time: The travel time criterion c_{time} reflects the duration in seconds of the cyclist's journey. Travel time is a sensitive factor in cyclists' route planning especially for commuting purposes. Our travel time calculation takes into account average cyclist's speed, uphills and downhills, quality of the road surface, and obstacles. To model the slowdown caused by obstacles such as stairs or crossings, we define the slowdown coefficient $r_{\text{slowdown}} : E \to \mathbb{R}_0^+$ which returns the slowdown in seconds on a given edge $(u,v) \in E$. For the case of uphill rides, we define the positive vertical ascend $a : E \to \mathbb{R}_0^+$ for a given edge $(u,v) \in E$ as

$$a((u,v)) := \begin{cases} h(v) - h(u) & \text{if } h(v) > h(u) \\ 0 & \text{otherwise} \end{cases}$$

where $h : V \to \mathbb{R}$ returns the elevation for each node $u \in V$. Analogously, for the case of downhill rides, we define the positive vertical descend $d : E \to \mathbb{R}_0^+$. We also define the positive descend grade $d' : E \to \mathbb{R}_0^+$ as $d'((u,v)) := \frac{d((u,v))}{l((u,v))}$ where $l((u,v))$ is the length of the edge $((u,v))$. To model the speed acceleration caused by vertical descend for a given edge $(u,v) \in E$, we define the downhill speed multiplier $s_d : E \to \mathbb{R}^+$ which depends on the positive descend grade d' and it is in the interval $[1, 2.5]$.

Considering the integrated effect of the edge length, the change in the elevation, and edge associated features, the travel time criterion is defined as

$$c_{\text{time}}((u,v)) = \frac{\text{distance}}{\text{speed}} + \text{slowdown} = \frac{l((u,v)) + a_l \cdot a((u,v))}{s \cdot s_d((u,v), s_{\text{dmax}}) \cdot r_{\text{time}}((u,v))} + r_{\text{slowdown}}((u,v))$$

where s is the average cruising speed of a cyclist and a_l is the penalty coefficient for uphill rides. The criteria coefficient $r_{\text{time}}((u,v))$ expresses the effect of a set of features $f((u,v))$ assigned to a given edge $(u,v) \in E$ with respect to travel time criterion.

Comfort: The comfort criterion c_{comfort} captures the preference towards comfortable routes with good-quality surfaces and dedicated cycleways or streets with low traffic. The cost function for the comfort is defined as

$$c_{\text{comfort}}((u,v)) = \max\{r_{\text{surface}}((u,v)), r_{\text{traffic}}((u,v))\} \cdot l((u,v))$$

where the surface coefficient $r_{\text{surface}}((u,v))$ penalises bad road surfaces, obstacles such as steps, and places where the cyclist needs to dismount his or her bicycle, with small values indicating cycling-friendly surfaces. The traffic coefficient $r_{\text{traffic}}((u,v))$ measures traffic volumes by considering the infrastructure for cyclists and the types of roads, where low-traffic cycleways are assigned a small coefficient value. The comfort is weighted by the edge length $l((u,v))$, i.e., 500 m of cobblestones is worse than 100 m of cobblestones.

Elevation gain: The elevation gain criterion c_{gain} captures the cyclists' preference towards flat routes with minimum uphill segments. The cost function is defined as

$$c_{\text{gain}}((u,v)) = \frac{\text{distance}}{\text{speed}} = \frac{a_l \cdot a((u,v))}{s}$$

where s is the average cruising speed of a cyclist, $a((u,v))$ is the positive vertical ascend of the edge (u,v), and a_l is the penalty coefficient for uphill rides.

4 Heuristic-Enabled Multi-Criteria Label-Setting Algorithm

Our newly proposed *heuristic-enabled multi-criteria label-setting* (HMLS) algorithm extends the standard multi-criteria label-setting (MLS) algorithm [19] with several points for inserting speedup heuristic logic. The algorithm uses the following data structures: for each node $u \in V$, $L(u) := (u, (l_1(u), l_2(u), \ldots, l_k(u)), L^P(u))$ represents the *label* at a node u, which is composed of the node u, the cost vector $l(u)$ indicating the current cost values from the origin to the node u, and the predecessor label $L^P(u)$, which precedes $L(u)$ in an optimal route from an origin. A priority queue Q is defined to maintain all labels created during the search. Since each node may be scanned multiple times, we define the bag structure $Bag(u)$ for each node u to maintain the non-dominated labels at u.

The pseudocode of the heuristic-enabled MLS algorithm is given in Algorithm 1; the speedup specific logic of functions `terminationCondition`, `skipEdge`, and `checkDominance` is described in Section 5. The algorithm consists of the following steps:

Step 1 – Initialisation: For a k-criteria optimisation problem, the algorithm first initialises the priority queue Q and Bag for each $v \in V$. Then it initialises the label at the origin to $L(o) := (o, (l_1(o), l_2(o), \ldots, l_k(o)), null)$, where $l_i(o) = 0$ for $i = 1, 2, \ldots, k$. Finally, it inserts the initial label $L(o)$ into the queue Q and the $Bag(o)$.

Step 2 – Label expansion: The algorithm extracts a minimum label $current := (u, (l_1(u), l_2(u), \ldots, l_k(u)), L^P(u))$ from the priority queue Q (in a lexicographic order of a cost vector). For each outgoing edge (u,v), the algorithm computes a new cost vector $(l_1(v), l_2(v), \ldots, l_k(v))$ by adding the costs of the edge (u,v) to the current cost values $(l_1(u), l_2(u), \ldots, l_k(u))$. Then,

Algorithm 1: Heuristic-enabled multi-criteria label-setting algorithm.

Input: cycleway graph $G = (V, E, \overrightarrow{c})$, origin node o, destination node d
Output: full Pareto set of labels

```
1  Q := empty priority queue
2  Bag(∀v ∈ V) := empty set
3  L(o) := (o, (0, 0, ..., 0), null)
4  Q.insert(L(o))
5  Bag(o).insert(L(o))
6  while Q is not empty do
7      current := Q.pop()
8      u := current.getNode()
9      (l_1(u), l_2(u), ..., l_k(u)) := current.getCost()
10     L^P(u) := current.getPredecessorLabel()
11     if terminationCondition(current) then
12         break
13     end
14     foreach edge (u, v) do
15         l_i(v) := l_i(u) + c_i(u, v) for i = 1, 2, ..., k
16         next := (v, (l_1(v), l_2(v), ..., l_k(v)), current)
17         if skipEdge(next) then
18             continue
19         end
20         if checkDominance(next) then
21             Bag(v).insert(next)
22             Q.insert(next)
23         end
24     end
25 end
26 return Bag(d)
```

it creates a new label *next* using the node v, the cost vector $(l_1(v), l_2(v), \ldots, l_k(v))$ and the predecessor label *current*.

Function `skipEdge` (cf. Algorithm 1, line 17) prevents looping the path by checking the predecessor label in the label data structure, i.e., if previous node $L^P(u).getNode()$ is equal to the node v then the edge (u, v) is skipped.

Function `checkDominance` (cf. Algorithm 1, lines 20–23), by default, controls dominance between the label *next* and all labels inside $Bag(v)$. If *next* is not dominated, the algorithm inserts it into $Bag(v)$ and Q. Also, if some label inside $Bag(v)$ is dominated by *next*, it is eliminated from the bag structure and not considered in future search.

Step 3 – Pruning condition: The algorithm exits if the queue Q becomes empty. Otherwise, it continues with *Step 2*.

After the algorithm has finished, the optimal Pareto set of routes Π^* is extracted. Let $|Bag(d)| = |\Pi| = m$. Then, from labels $L_1, \ldots, L_m$ in the destination Pareto set of labels

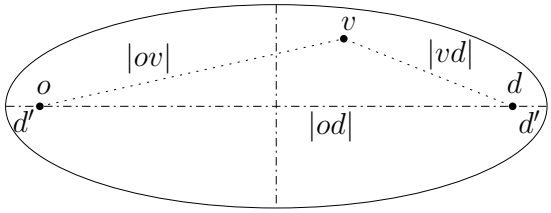


Figure 1 Geometry of the ellipse pruning condition.

$Bag(d)$, the routes $\pi_1, \ldots, \pi_m$ are extracted using the predecessor labels $L^P(\cdot)$. These routes comprise the set $\Pi^* = \{\pi_1, \pi_2, \ldots, \pi_m\}$ of optimal Pareto routes.

5 Speedups for the HMLS Algorithm

A significant drawback of the standard MLS algorithm is that it is very slow. The main parameter that affects the runtime of the algorithm is the size of the Pareto set. In general, the Pareto set can be exponentially large in the input graph size [21]. Furthermore, the MLS algorithm always explores the whole cycleway graph.

To accelerate the multi-criteria shortest path search, we introduce four speedup heuristics. Two of the heuristics are newly proposed by us: *ratio-based pruning* and *cost-based pruning*, while two are existing heuristics: *ellipse pruning* and *buckets*. Implementation-wise, the heuristics are incorporated into the heuristic-enabled MLS algorithm by defining the respective three heuristic-specific functions used in Algorithm 1.

Ellipse Pruning: The first speedup heuristic taken from [12] prevents the MLS algorithm from always searching the whole cycleway graph, even for a short origin-destination distance[1]. The heuristic permits visiting only the nodes that are within a predefined ellipse. The focal points of the ellipse correspond to the journey origin o and the destination d. Let $|od|$ be the direct origin-destination distance and d' the distance between origin and a peripheral point on the main axis of the ellipse. Then the length of the main axis $2a$ is equal to $|od| + 2d'$. During the search, in `skipEdge` function (cf. Algorithm 1, line 17), it is checked whether an edge (u, v) has its target node v inside the ellipse by checking the inequality $|ov| + |vd| \leq |od| + 2d'$, cf. Figure 1.

Ratio-Based Pruning: The ratio-based pruning terminates the search (long) before the priority queue gets empty (which means that the whole search space has been explored). A pruning ratio $\alpha \in \mathbb{R}^+$ is defined and the search is terminated when one of the criteria cost values, e.g., $l_1(u)$, in the *current* label exceeds α times the best so far value of the same criterion for a route that has already reached the destination (this is checked in the `terminationCondition` function, cf. Algorithm 1, line 11).

Cost-Based Pruning: The third heuristic we use does not expand the search to a label $L(v)$ which is very close in the cost space (criteria $c_1, \ldots, c_k$) to the existing non-dominated labels at the node v. The newly generated label $L(v)$ with a closer Euclidean distance than $\gamma \in \mathbb{R}^+$

[1] Note that in contrast with single-criterion Dijkstra's algorithm, the MLS algorithm does not stop when the destination node is first reached.

Table 1 Graph sizes for the experiments.

Graph	Nodes	Edges	Area
Prague A	9411	20420	Old Town, Vinohrady
Prague B	9665	20808	Strahov, Brevnov
Prague C	10652	24121	Liben, Vysocany

is discarded inside the `checkDominance` function (cf. Algorithm 1, line 20). Therefore, the search process is accelerated since fewer labels are inserted into the queue and the bag.

Buckets: The last heuristic defined in [7] discretizes the cost space using buckets for the criteria values. The heuristic is executed in the `checkDominance` function (cf. Algorithm 1, line 20). A function $bucketValue : \mathbb{R}_0^+ \rightarrow \mathbb{N}$ is used to assign a real cost value l_i an integer bucket value $bucketValue(l_i)$.

6 Evaluation

To evaluate our approach, we consider the real cycleway network of Prague. Prague is a challenging experiment location due to its complex geography and fragmented cycling infrastructure, which raises the importance of proper multi-criteria routing.

6.1 Experiment Setting

We evaluate our solution on cycleway graphs corresponding to three distinct areas of the city of Prague. We have chosen parts Prague A, Prague B, and Prague C to be different in terms of network density, nature of the cycling network and terrain topology so as to evaluate the performance of heuristics across a range of conditions. The sizes of the evaluation graphs are depicted in Table 1. The graphs are also shown in Figure 2 in the map of Prague. The specifics of each evaluation area are the following:

- Prague A: This graph covers a flat city centre area of the Old Town with many narrow cobblestone streets and Vinohrady with the grid layout of streets.
- Prague B: This graph covers a very hilly area of Strahov and Brevnov with many parks.
- Prague C: This graph covers residential areas of Liben and Vysocany further from the city centre. There are many good cyclepaths in this area.

All evaluation cycleway graphs are strongly connected. The size of evaluation graphs allows us to run the MLS algorithm without any speedups, which is crucial for comparing the quality of heuristic and optimal solutions.

For each graph evaluation area, a set of origin-destination pairs generated randomly with a uniform spatial distribution, was used in the evaluation. First, we generated 130 origin-destination pairs for each of graphs Prague A, B, and C. The minimum origin-destination distance is set to 500 m. The longest routes have approximately 4.5 km. From these 130 origin-destination pairs, we filtered out 15 pairs with the smallest size of the optimal Pareto set and 15 pairs with the largest size of the optimal Pareto set to receive a set of 100 origin-destination pairs. We executed the MLS algorithm and the HMLS with all 11 heuristic combinations using the same generated 100 origin-destination pairs for each graph Prague A, B, and C. Therefore, each heuristic combination is evaluated on 300 origin-destination pairs.

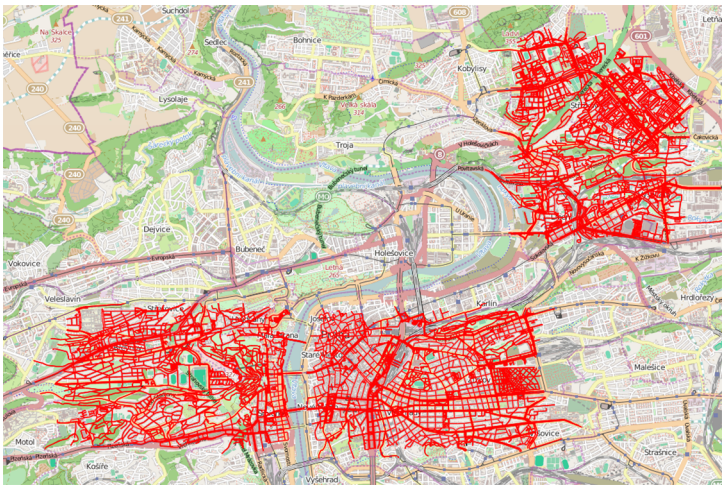


Figure 2 Evaluation graphs Prague B, Prague A, and Prague C (from left to right).

The parameters in the cost functions were set as follows. The average cruising speed is $s = 14$ km/h and the penalty coefficient for uphill is $a_l = 13$ (according to the route choice model developed in the user study [3]). Configuration parameters for the heuristics were set so as to maximize the ratio between the algorithm runtime and the quality of the solution (see the next section), as measured on the three graphs Prague A, B, and C. Specifically, the following values were used: $d' = 500$ m for ellipse pruning, $\alpha = 1.6$ for ratio-based pruning, $\gamma = \frac{C_1}{5}$ for cost-based pruning, and $(15, 2500, 4)$ for buckets. The multi-criteria route planning algorithm is implemented in JAVA 7. The results obtained are based on running the algorithm on a single core of a 2.4 GHz Intel Xeon E5-2665 processor of a Linux server. OpenStreetMap data is used to create the Prague cycleway graphs.

6.2 Evaluation Metrics

We consider two categories of evaluation metrics: *speed* and *quality*. We use the following metrics to measure the algorithm speed:

- Average runtime in ms for each origin-destination pair together with its standard deviation σ_{runtime}.
- Average speedup over the MLS algorithm in terms of algorithm runtime.

We use the following metrics to measure the quality of returned routes:

- Average distance $d_c(\Pi^*, \Pi)$ of the heuristic Pareto set Π from the optimal Pareto set Π^* in the cost space. Distance $d_c(\pi^*, \pi)$ between two routes π^* and π is measured as the Euclidean distance in the unit three-dimensional space of criteria values normalized to the $[0, 1]$ range.

$$d_c(\Pi^*, \Pi) := \frac{1}{|\Pi^*|} \sum_{\pi^* \in \Pi^*} \min_{\pi \in \Pi} d_c(\pi^*, \pi)$$

  Intuitively, $d_c(\pi^*, \pi) = 0.1$ corresponds to a 6% difference in each criterion, assuming the difference to optimum is distributed equally across all three criteria.
- Average number of routes $|\Pi|$ in the Pareto set Π together with its standard deviation $\sigma_{|\Pi|}$.
- The percentage of Pareto routes $\Pi_{\%}$ in heuristic Pareto set Π that are equal to routes in the optimal Pareto set Π^*.

Table 2 Evaluation of the heuristic performance on three graphs Prague A, B, and C. Primary metrics are marked by bold column headings (runtime in ms and average distance $d_c(\Pi^*, \Pi)$). Non-dominated heuristic combinations with respect to speed and quality are denoted by bold font. Abbreviations used: Buckets → B., Ellipse → E., Ratio → R.

Heuristic	Speedup	**Runtime**	σ_{runtime}	$\lvert\Pi\rvert$	$\sigma_{\lvert\Pi\rvert}$	$\mathbf{d_c}$	$\Pi_{\%}$
MLS	-	3 586 263	4 390 939	1 351	1 304	-	100.0
HMLS+B.	875	**4 100**	3 335	37	32	**0.131**	60.9
HMLS+Cost	399	8 983	4 508	92	49	0.232	58.1
HMLS+R.	14	264 174	426 887	835	868	0.095	99.9
HMLS+R.+B.	2966	1 209	1 353	31	28	0.193	65.1
HMLS+R.+Cost	734	4 887	3 265	82	44	0.275	60.8
HMLS+E.	19	**184 734**	287 402	1 310	1 275	**0.008**	99.6
HMLS+E.+B.	6791	**528**	721	36	32	**0.136**	60.9
HMLS+E.+Cost	1732	2 070	1 976	91	49	0.235	58.5
HMLS+E.+R.	46	**77 468**	128 784	823	858	**0.098**	99.8
HMLS+E.+R.+B.	10308	**348**	461	31	28	**0.196**	65.1
HMLS+E.+R.+Cost	1921	1 866	1 902	82	44	0.276	61.0

6.3 Results

Table 2 summarizes the evaluation of the HMLS algorithm and its heuristics. The MLS algorithm is used as a baseline for the evaluation of the proposed heuristics and their combinations. Columns d_c and $\Pi_{\%}$ are calculated with respect to the optimal Pareto set Π^* returned by the MLS algorithm. The MLS algorithm returns optimal solutions (1351 routes in the Pareto set on average) at the expense of a prohibitively high runtime (one hour per one origin-destination pair on average).

As anticipated, all heuristic methods are significantly faster than the pure MLS algorithm. First, we have compared the methods using the two primary metrics in each category – the average runtime and the heuristic measured by the average distance $d_c(\Pi^*, \Pi)$ in the cost space. From the perspective of this two metrics, there are five non-dominated combinations of heuristics, cf. filled bars in Figure 3 and bold values in Table 2. In the following, we only discuss non-dominated combinations of heuristics.

The *HMLS+Ellipse* heuristic performs best in terms of the quality of the solution. It successfully prunes the search space with $d_c(\Pi^*, \Pi) = 0.008$. The average runtime of this heuristic is around three minutes. This heuristic is very good for combining with other heuristics, it offers one order of magnitude speedup over the MLS algorithm with a negligible quality loss (99.6% of the routes in the heuristic Pareto set Π are equal to the ones in the optimal Pareto set Π^*).

The *HMLS+Ellipse+Ratio* heuristic offers very good quality with $d_c(\Pi^*, \Pi) = 0.098$, the average runtime is around 80 seconds. The search space is pruned geographically by the ellipse pruning and the search is also terminated sooner by the ratio-based pruning method.

With only a small decrease of the solution quality to $d_c(\Pi^*, \Pi) = 0.131$, *HMLS+Buckets* heuristic offers a significant additional speedup in average runtime to approximately 4.1 seconds. This makes this heuristic (and also the two following ones) usable for real time applications, e.g., a web-based bicycle journey planner.

When the ellipse pruning method is combined with the *Buckets* heuristic, the average runtime of *HMLS+Ellipse+Buckets* is lowered to approximately 528 ms while keeping almost the same quality $d_c(\Pi^*, \Pi) = 0.136$.

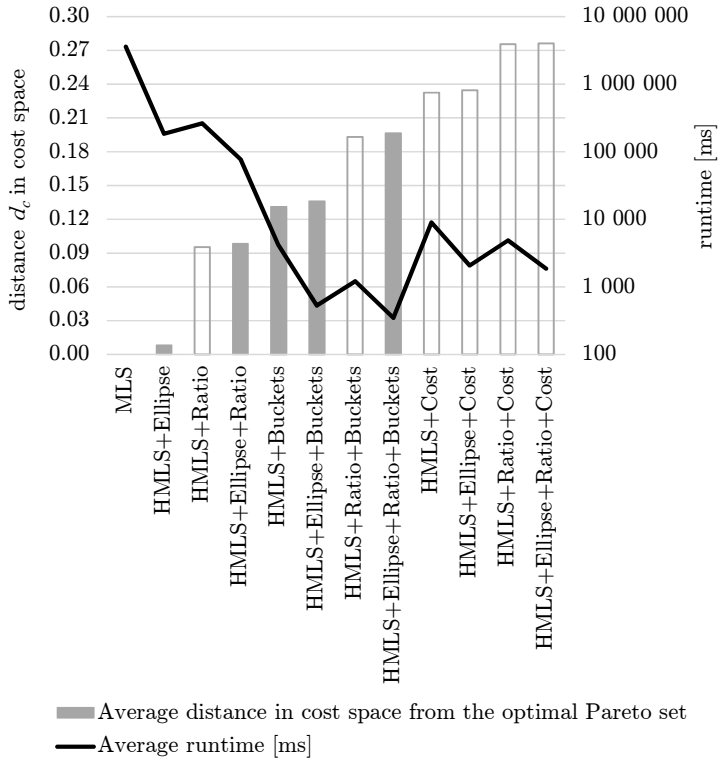


Figure 3 Speed and quality for the HMLS algorithm and all heuristic combinations sorted by the quality from the best (MLS on the left hand side) to the worst. Non-dominated heuristic combinations have grey filled in bars.

The last combination *HMLS+Ellipse+Ratio+Buckets* performs best in terms of average runtime which is approximately 350 ms, i.e., it has four orders of magnitude speedup over the pure MLS algorithm. The quality of this combination is reflected by higher $d_c(\Pi^*, \Pi) = 0.196$, still over 65% of the routes in the heuristic Pareto set Π are equal to the ones in the optimal Pareto set Π^*.

To provide a deeper insight in search runtimes, we show in Figure 4 how the runtime of the *HMLS+Ellipse+Buckets* heuristic depends on the direct origin-destination distance. Although the runtime increases with the origin-destination distance, the rate of increase slows down as the origin-destination distance grows. This behaviour was confirmed in our initial scale-up experiments that resulted in less than 10 second response times even for 20 times larger cycleway graph covering the whole city of Prague (approx. 200 km^2). Finally, in Figure 5 we illustrate the route distribution from the optimal Pareto set of routes in the physical space on an example of a route around a hilly area in Zizkov, Prague 3.

To summarize, we have evaluated 11 different combinations of heuristics from which 5 combinations dominated the others in terms of quality and speed. The heuristics offer significant *one to four orders of magnitude speedup* over the pure MLS algorithm in terms of average runtime. The speedup is achieved by lowering the number of iterations and also the number of dominance checks in each iteration. *HMLS+Ellipse* is the best heuristic in terms of quality of the produced Pareto set while *HMLS+Ellipse+Ratio+Buckets* is the best heuristic in terms of average runtime. Taking into the account the trade-off between the quality of a solution and the provided speedup, we consider *HMLS+Ellipse+Buckets* heuristic to have the best ratio between the quality and speed.

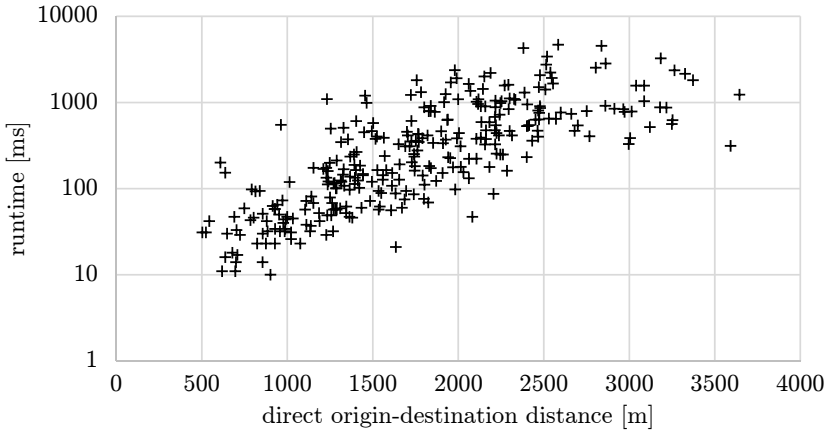


Figure 4 The runtime of *HMLS+Ellipse+Buckets* in milliseconds in dependency on the direct origin-destination distance.

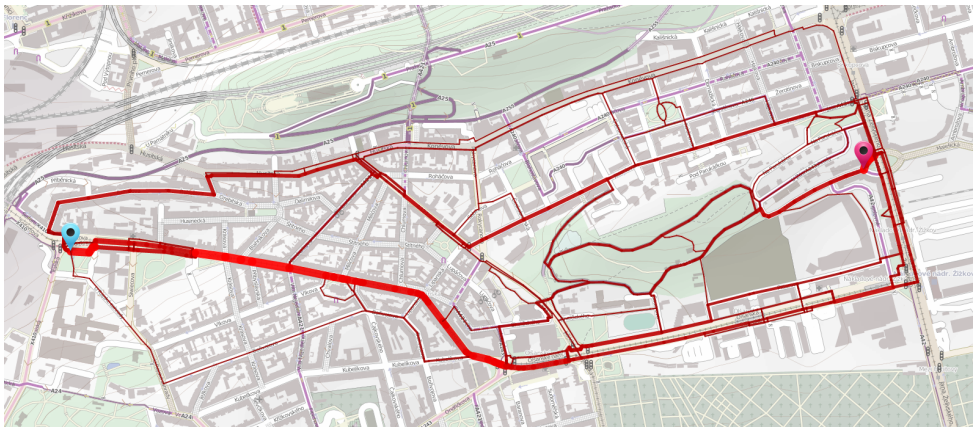


Figure 5 Distribution of 503 routes from the optimal Pareto set of routes. The more routes use a given cycleway network segment, the wider is the depicted line.

7 Conclusions

We have made bicycle routing that properly considers multiple realistic route choice criteria fast enough for practical, interactive use. We have achieved so by employing four heuristic speedup techniques for multi-criteria shortest path search. The speedup heuristics provide a variable trade-off between the search time and the completeness and quality of the suggested routes and they enable fast response times without severely compromising the quality of the results.

The multi-criteria search produces often large Pareto sets with many similar routes. As a future work, we plan to provide a filtering method (e.g., based on our initial clustering method [24]) that would extract several representative routes from a potentially very large set of Pareto routes. Furthermore, we plan to extend the underlying cycleway graph model to consider additional features such as detailed junction models with traffic lights and penalisation of turns.

Acknowledgements. Supported by the European social fund within the framework of realising the project "Support of inter-sectoral mobility and quality enhancement of research teams at Czech Technical University in Prague" (grant no. CZ.1.07/2.3.00/30.0034), period of the project's realisation is 1 Dec 2012 – 30 Jun 2015. Supported by the European Union Seventh Framework Programme (grant agreement no. 609023) and by the Czech Technical University (grant no. SGS13/210/OHK3/3T/13). Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated.

References

1 H. Bast, D. Delling, A. Goldberg, M. Muller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. Werneck. Route Planning in Transportation Networks. Technical report, Microsoft Research, 2014.

2 Hannah Bast, Mirko Brodesser, and Sabine Storandt. Result Diversity for Multi-Modal Route Planning. In Daniele Frigioni and Sebastian Stiller, editors, *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 33 of *OpenAccess Series in Informatics (OASIcs)*, pages 123–136, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

3 J. Broach, J. Dill, and J. Gliebe. Where do cyclists ride? A route choice model developed with revealed preference GPS data. *Transportation Research Part A: Policy and Practice*, 46(10):1730 – 1740, 2012.

4 D. W. Corne. The good of the many outweighs the good of the one: evolutionary multi-objective optimization. *IEEE Connections Newsletter*, pages 9–13, 2003.

5 B. C. Dean. Continuous-time dynamic shortest path algorithms. Master's thesis, Massachusetts Institute of Technology, 1999.

6 D. Delling, J. Dibbelt, T. Pajor, D. Wagner, and R. F. Werneck. Computing and Evaluating Multimodal Journeys. Technical Report 2012-20, Faculty of Informatics, Karlsruhe Institut of Technology, 2012.

7 D. Delling, J. Dibbelt, T. Pajor, D. Wagner, and R. F. Werneck. Computing multimodal journeys in practice. In *SEA*, pages 260–271, 2013.

8 D. Delling and D. Wagner. Pareto paths with sharc. In *Proceedings of the 8th International Symposium on Experimental Algorithms, volume 5526 of LNCS*, pages 125–136. Springer, 2009.

9 D. Delling and D. Wagner. Time-dependent route planning. In *Robust and Online Large-Scale Optimization*, pages 207–230. Springer, 2009.

10 C. Dora and M. Phillips. *Transport, environment and health.* WHO Regional Office for Europe, Copenhagen, 2000.

11 V. Filler. (AUTO*MAT) Private communication, 2013. Why cycle route planners are important for cyclists.

12 L. Han, H. Wang, and W. Mackey Jr. Finding shortest paths under time-bandwidth constraints by using elliptical minimal search area. *Transportation Research Record: Journal of the Transportation Research Board*, No. 1977:225–233, 2006.

13 D. Herlihy. *Bicycle: the history.* Yale University Press, 2004.

14 H. H. Hochmair and J. Fu. Web Based Bicycle Trip Planning for Broward County, Florida. In *ESRI User Conference*, 2009.

15 J. Hrncir, Q. Song, P. Zilecky, M. Nemet, and M. Jakob. Bicycle route planning with route choice preferences. In *Prestigious Applications of Artificial Intelligence (PAIS)*, 2014.

16 P. L. Jacobsen. Safety in numbers: more walkers and bicyclists, safer walking and bicycling. *Injury Prevention*, 9(3):205–209, 2003.

17 E. Machuca and L. Mandow. Multiobjective heuristic search in road maps. *Expert Systems with Applications*, 39(7):6435–6445, 2012.

18 L. Mandow and J. De La Cruz. Multiobjective A* search with consistent heuristics. *J. ACM*, 57(5), June 2008.

19 E. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236 – 245, 1984.

20 M. Muller-Hannemann and M. Schnee. Finding all attractive train connections by multi-criteria pareto search. In *ATMOS*, pages 246–263, 2004.

21 M. Müller-Hannemann and K. Weihe. On the cardinality of the pareto set in bicriteria shortest path problems. *Annals of Operations Research*, 147(1):269–286, 2006.

22 P. Perny and O. Spanjaard. Near admissible algorithms for multiobjective search. In *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pages 490–494, Amsterdam, The Netherlands, 2008. IOS Press.

23 G. Sauvanet and E. Neron. Search for the best compromise solution on multiobjective shortest path problem. *Electronic Notes in Discrete Mathematics*, 36:615–622, 2010.

24 Q. Song, P. Zilecky, M. Jakob, and J. Hrncir. Exploring pareto routes in multi-criteria urban bicycle routing. In *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, pages 1781–1787, Oct 2014.

25 B. S. Stewart and Ch. C. White. Multiobjective A*. *Journal of the ACM (JACM)*, 38(4):775–814, 1991.

26 J. G. Su, M. Winters, M. Nunes, and M. Brauer. Designing a route planner to facilitate and promote cycling in Metro Vancouver, Canada. *Transportation Research Part A: Policy and Practice*, 44(7):495–505, 2010.

27 R. J. Turverey, D. D. Cheng, O. N. Blair, J. T. Roth, G. M. Lamp, and R. Cogill. Charlottesville bike route planner. In *Systems and Information Engineering Design Symposium (SIEDS)*, 2010.

28 M. Winters, G. Davidson, D. Kao, and K. Teschke. Motivators and deterrents of bicycling: comparing influences on decisions to ride. *Transportation*, 38(1):153–168, 2011.

Routing of Electric Vehicles: Constrained Shortest Path Problems with Resource Recovering Nodes*

Sören Merting[1], Christian Schwan[2], and Martin Strehler[2]

1 **Technische Universität München**
Boltzmannstr. 3, 85748 Garching near Munich, Germany
soeren.merting@in.tum.de

2 **Brandenburg University of Technology**
Platz der Deutschen Einheit 1, 03046 Cottbus, Germany
{christian.schwan, martin.strehler}@b-tu.de

Abstract

We consider a constrained shortest path problem with the possibility to refill the resource at certain nodes. This problem is motivated by routing electric vehicles with a comparatively short cruising range due to the limited battery capacity. Thus, for longer distances the battery has to be recharged on the way. Furthermore, electric vehicles can recuperate energy during downhill drive. We extend the common constrained shortest path problem to arbitrary costs on edges and we allow regaining resources at the cost of higher travel time. We show that this yields not shortest paths but shortest walks that may contain an arbitrary number of cycles. We study the structure of optimal solutions and develop approximation algorithms for finding short walks under mild assumptions on charging functions. We also address a corresponding network flow problem that generalizes these walks.

1998 ACM Subject Classification G.2.2 Graph Theory – Path and circuit problems, Network problems, G.2.3 Applications

Keywords and phrases routing of electric vehicles, constrained shortest paths, FPTAS, constrained network flow

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.29

1 Motivation

1.1 Electric Vehicles

Electric vehicles are a cornerstone towards eco-friendly mobility. Charged with renewable energy they contribute to a responsible use of our limited resources. Compared to common vehicles with combustion engines, there are still some disadvantages. The comparatively short range due to the restricted battery capacity is most likely the main reason for the poor popularization of electric vehicles up to now. Furthermore, for these cars even fast charging of the battery lasts significantly longer than traditional refueling.

This gives rise to many interesting mathematical questions concerning the routing of electric vehicles. First of all, electric vehicles can recuperate energy during downhill drive, i.e., there are road segments where energy consumption is negative. This also implies that for deciding whether a given path is feasible the remaining energy supply has to be checked

* This work was supported by the German Federal Ministry of Education and Research (BMBF), funding code 05M13ICA.

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt; pp. 29–41

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

en route. Obviously, it must never be less than zero, but we also must not store more energy than the battery can hold in every point in time. Thus, is it possible to reach the destination? When no such path through the network can be found, battery charging stations have to be visited. But where should one charge the battery, especially when different charging stations provide different charging characteristics, e.g., rather slow charging at home compared to fast charging or even battery swapping at professional charging stations. How much energy should be regained at a certain charging station when charging costs (in time equivalents) are not proportional to the charged amount as the charge rate typically decreases at higher charge levels? All in all, which is the fastest feasible route to the destination?

There are also several fields of application for fleets of electric vehicles in commerce and industry. Exemplary, electric automated guided vehicles (agv) are operated in harbor terminals, manufacturing facilities or warehouses to move materials or containers around. Whereas reachability is of minor interest, charging all those vehicles may require a sophisticated planning. Thus, also the network flow version of the electric vehicle routing with limited charging capacity is worth studying.

1.2 Related Work

In the constrained shortest path problem (CSP), we are given a graph $G = (V, E)$ with a cost function $c : E \to \mathbb{R}$ and lengths or resources $r_i : E \to \mathbb{R}$. In general, costs and resources are assumed to be non-negative. Now, one seeks for a shortest path P from a vertex s to a vertex t which obeys the resource constraints R_i, $i \in N = \{1, \dots, k\}$. That is, we want to find an s-t-path P minimizing $\sum_{e \in P} c(e)$ such that $\sum_{e \in P} r_i \leq R_i \ \forall i \in N$.

For unit edge lengths, i.e., there is only one resource with $r(e) = 1$ for all edges, this problem can be solved easily using a labeling algorithm. In other words, we want to find a shortest path using at most R edges. The easiest way to solve this problem is a modified Bellman-Ford-algorithm which stops after R iterations. Here, it is important to update the labels simultaneously which automatically yields paths with no more than R edges.

The problem becomes $\mathcal{NP}$-complete when the resource function can take arbitrary non-negative values. This can be seen by a reduction of 2-Partition [10]. Thus, approaches focussing on listing all Pareto-optimal solutions via dynamic programming suffer from pseudopolynomial running times. Fortunately, several fully polynomial time approximation schemes (FPTAS) have been found to tackle this problem. Approaches based on *rounding and scaling* were suggested by Warburton [22] who used this technique to compute Pareto-optimal solutions for the multiple-objective shortest path problem. Hassin [13] adapted and improved this concept for constrained shortest paths. Shortly after, Phillips [18] presented an FPTAS which uses a Dijkstra search in a resource-expanded graph. In a second line of research, *multi-phase algorithms* were purposed. Most commonly, good lower and upper bounds are computed in a first phase and the remaining gap is closed in a second phase. Handler and Zang [12] used a Lagrange relaxation of an edge-based integer linear programming formulation to compute the lower and upper bounds. In the second phase, the gap is closed with help of a k-shortest path algorithm. Similarly, Beasley and Christofides [5] closed the gap with a branch and bound strategy. Another approach was proposed by Mehlhorn and Ziegelmann [15, 23]. They use the dual of a relaxed path-based integer linear programming formulation for the first phase. There, the separation problem can be solved efficiently which is used to compute the lower and upper bounds. For the second phase, additional approaches like path ranking and labeling strategies are discussed to close the duality gap.

A flow version of the constrained shortest path problem dates back to 1978, when Lovász et al. [14] derived a version of Menger's theorem for node-disjoint length bounded paths. This

was independently extended to edge-disjoint paths by Exoo [7] and Niepel and Šafaříková [17]. Edge-disjoint paths can be interpreted as a 0-1-valued flow with unit capacity and unit edge length. Recently, a more general approach with fractional flow values was studied by Baier et al. [1, 2]. Such a flow is feasible if and only if there exists a path decomposition such that each flow carrying path fulfills the resource constraint. However, the authors of [1, 2] show that it is $\mathcal{NP}$-complete to decide whether a given edge-flow can be decomposed into such paths. Consequently, deciding whether there is a length-bounded flow of a certain value is also an $\mathcal{NP}$-complete problem. The authors present a fully polynomial time approximation scheme based on an approximation algorithm for constrained shortest paths. Furthermore, the authors show that the ratio of the minimum fractional length-bounded s-t-cut, i.e., edges can be chosen partially, and the minimum integral s-t-cut can be of order $\Omega(\sqrt{n})$ even for unit resources. This bound carries over to the gap between fractional and integral flow.

The routing of electric vehicles has been studied to a lesser extent in the literature. In [19] the authors cope with negative resource consumption due to recuperation by taking the potential energy into account. Thus, a new non-negative cost function is determined such that an A* algorithm can be applied to compute paths with minimum energy consumption. Baum et al. [4] present a very fast routing algorithm for electric vehicles by extending the customizable route planning approach of [6]. Additionally, they consider reducing speed to increase range in a subsequent paper [3]. However, recharging at charging stations is not considered in these papers. Recently, a constrained shortest path problem for electric vehicles with recharging stations was introduced by Storandt [20]. In this setting, it is assumed that the whole battery is swapped at each charging point. Thus, recharge time is constant and the battery is always recharged to full capacity after each visit of a charging station. Consequently, first results were also obtained for the facility location problem for charging stations [21].

1.3 Our Contribution

In this paper, we extend previous results for resource constrained shortest paths and flows to networks with recharging nodes. Due to recuperation, negative resource consumption is now possible. Further, we consider recharging nodes where the resource can be refilled by paying additional costs.

The paper is organized as follows. Firstly, we give some definitions and fix the notation. Afterwards, we will show that in our setting shortest paths can contain cycles and even a node for charging can be visited more than once. Therefore, we also introduce a new type of conservative cost functions in order to avoid one class of cycles. In Section 4, we develop an FPTAS for the shortest path problem with charging. Furthermore, we address the flow variant of this problem, i.e., we present analytic results for the min cost flow problem with length constraints and recharging in Section 5.

2 Preliminaries

Throughout this paper the underlying structure is a *finite directed graph* $G = (V, E)$ with $n = |V|$ *vertices* or *nodes* and $m = |E|$ *edges*. Given a source vertex s and a target vertex t, an s-t-path P is a sequence of edges $(e_1 = (s, v_1), \ldots, e_k = (v_{k-1}, t))$ fitting head to tail and each edge appears only once. However, as we will show, paths are too restrictive for finding the most efficient route from s to t. An s-t-walk W is again a sequence of edges $(e_1 = (s, v_1), \ldots, e_k = (v_{k-1}, t))$, but each edge may appear more than once. A cycle C is a special kind of walk, where the first node is equal to the last one.

In general, one may allow of several resources, but here we limit our study to a cost function and a single resource constraint. For each edge $e \in E$, there are two parameters, namely *cost* $c : E \to \mathbb{R}_{\geq 0}$ and *resource consumption* $r : E \to \mathbb{R}$. Since costs are related to travel time in most of our applications, we assume the cost function to be non-negative. In contrast, we explicitly allow recuperation of energy. Thus, there may be edges with negative resource consumption. Of course, in accordance to basic laws of physics it is assumed that the total energy consumption on a cycle is non-negative.

▶ **Definition 1.** A resource function r is called *conservative* if for each cycle C there is

$$\sum_{e \in C} r(e) \geq 0.$$

In the following we require the resource function to be conservative. With our application in mind, we refer to a node as *charging node* if at this node the resource value of a path can be increased at the expense of the cost value . Furthermore, our main application requires the ability to model a non-linear charging process. We describe this *charging process* with help of a *charging function* $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ which is continuous and increasing and maps from the amount of recharged resources to the resulting costs.

▶ **Definition 2.** For a subset $S \subseteq V$ of *charging nodes*, the *charging function* $f_v : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, $v \in S$, is a continuous and increasing function. If the path P arrives at vertex $v \in S$ with x units of remaining resources we recharge μ resources which increase the cost of P by $f_v(x + \mu) - f_v(x)$.

This definition via the charging function has two advantages. Bypassing a charging node, that is $\mu = 0$, causes no additional cost. Further, there is also no need to interrupt the charging process, since $f_v(x+(\mu_1+\mu_2)) - f_v(x) = (f_v(x+\mu_1) - f_v(x)) + (f_v(x+\mu_1+\mu_2) - f_v(x+\mu_1))$. That is, we can assume that the desired amount of resources is recharged in one step. However, this does not prevent the optimal walk to visit a charging node $v \in S$ twice as we will show in Section 3.

▶ **Definition 3.** An *s-t-walk W with charging* is a sequence of tuples (e_i, μ_i), $i \in \{1, \ldots, k\}$ with edges $e_i = (v_{i-1}, v_i)$ and recharged resources $\mu_i \geq 0$ at v_i such that the edges form an s-t-walk. Furthermore, we require $v_i \in S$ if $\mu_i > 0$.

Contrary to constrained shortest paths, it is not sufficient to check the resource constraint only at the target terminal. Due to negative resource consumption on some edges, feasibility has to be checked at each intermediate point, too. Let us assume an initial resource value of R at node s and a lower bound of 0. Furthermore, let $\overline{R}$ be the maximum amount of storable resources. However, this upper bound is a soft bound. We will not violate it by recharging, since this will cause additional costs. If we would exceed this bound by recuperation, excess resources is not stored.

Let $r(W, j)$ be the remaining resources on walk W after the jth edge and after charging μ_j units of resources at node v_j. Starting with $r(W, 0) = R$ we define iteratively:

$$r(W, j) := \min\{\overline{R}, r(W, j-1) - r(e_j) + \mu_j\}.$$

▶ **Definition 4.** Given initial resources R and resource bound $\overline{R}$, a walk W with charging is feasible iff $r(W, j) - \mu_j \geq 0$ for all $j \in \{1, \ldots, k_W\}$.

Now, the cost of a walk W with charging is defined by

$$c(W) = \sum_{j=1}^{k} c(e_j) + \sum_{j=1,\ \mu_j > 0}^{k} f_{v_j}\left(r(W, j)\right) - f_{v_j}\left(r(W, j) - \mu_j\right).$$

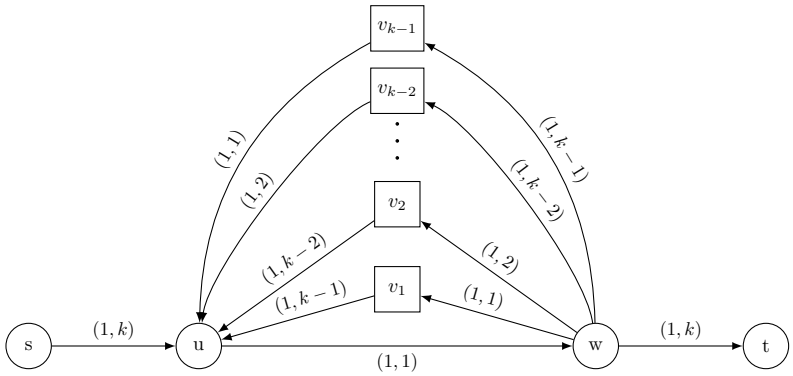


Figure 1 Network with common nodes (circle) and charging nodes (boxed). Edge labels denote *(cost, resource consumption)*. Initial resources and maximum storage are $R = \overline{R} = k + 2$. Charging is instantaneous/free, i.e., $f \equiv 0$ for all charging nodes, until reaching $\overline{R}$.

That is, the costs of a walk consist of costs for crossing the edges and of costs for recharging resources. Note that $r(W, j)$ already includes charging at the head of e_j, so resources are refilled from $r(W, j) - \mu_j$ up to $r(W, j)$. Let $r(W) = R - r(W, k_W)$ be the total resource difference of the walk W with k_W edges. A walk W_1 with charging is *dominating* a walk W_2 iff $c(W_1) < c(W_2)$ and $r(W_1) \leq r(W_2)$ or $c(W_1) \leq c(W_2)$ and $r(W_1) < r(W_2)$. Hence, a walk W_1 with charging is *Pareto-optimal* if there exists no walk W_2 which dominates W_1. A walk W_1 with charging is a *shortest walk*, if there is no other feasible walk W_2 with $c(W_2) < c(W_1)$.

3 Structure of Optimal Solutions

From an algorithmic point of view, the problem of deciding whether there is a walk with charging of cost $\leq c$ is clearly $\mathcal{NP}$-complete. On the one hand it is a decision problem and a walk given as certificate is easy to check, on the other hand the problem includes the constrained shortest path problem. Yet, in some sense it is also not harder than the constrained shortest path problem if we require an additional property of the network and its charging functions. We will derive this property in this section and we present a corresponding FPTAS in the next section.

However, the problem of finding shortest walks with charging has a richer combinatorial variety. In this section, we will also emphasize some of the most important properties of such walks. First of all, an optimal walk may contain cycles despite the positive cost function and the conservative resource consumption function.

▶ **Lemma 5.** *A shortest walk with charging may contain arbitrarily many cycles even when no charging costs exist. There are networks where an optimal walk passes a certain edge $\Omega(n)$-times.*

Proof. Consider the network in Figure 1. Charging node v_j, $j = 1, \ldots, k-1$ is reachable from charging nodes v_i with $i \geq j - 1$. Node v_j is only reachable from node v_{j-1} if the resources are completely recharged in v_{j-1}. Further, t is only reachable from v_{k-1}. Thus, even the shortest walk has to visit all charging nodes. Edge (u, w) is used k-times. ◀

The construction in Figure 1 may lead to the assumption that a shortest walk visits charging nodes at most once. However, this is not true.

▶ **Lemma 6.** *A shortest walk with charging can visit a single charging node several times.*

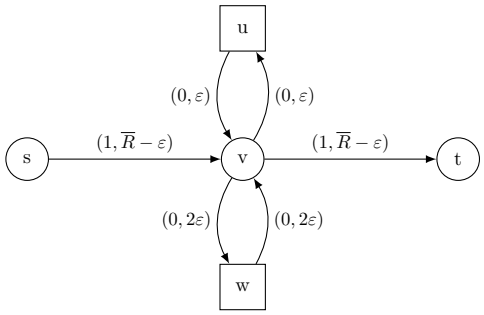


Figure 2 Network with two charging nodes. Charging is very expensive at u, that is $f_u(x) = x$. Charging at w is very fast/free, i.e., $f_w(x) = 0$. Even in this simple instance with linear charging functions, u is visited twice by the optimal path.

Proof. Again, we provide an example in Figure 2. Assume $R = \overline{R} = 1$. A walk starting in s may only reach v and u subsequently. From u, target t can only be reached by a full recharge $\mu_u = 1$. With $f_u(x) = x$, this also induces costs of 1. For small $\varepsilon > 0$, it is better to refill only 3ε. Then, w can be reached where completely charging is free. However, t is not reachable from w without visiting u and recharging 3ε again. Thus, if $\varepsilon < \frac{1}{6}$, the shortest path visits u twice. ◀

More than two visits of a charging node can easily be achieved by choosing the charging functions properly. Instead of a single free recharge node w as in Figure 2, assume several charging nodes w_i each providing free recharge only for a small subset of the domain of the charging functions and very high fees otherwise. Then, a shortest path can visit u several times. Actually, it is also possible to choose the charging function of two stations so that an optimal path would have to switch infinitely often between these two stations.

However, the network in the proof of Lemma 6 presumes somewhat unnatural charging functions. To prevent such unnatural walks, we introduce an additional constraint similar to the restrictions of conservative cost functions to prevent negative cost cycles.

▶ **Definition 7.** Let C be a cycle, let $S_C = S \cap V(C)$ be charging nodes on C, and let $v \in S_C$ be one of these nodes. Let W_C be the cyclic walk with charging, starting at node v with $r(W, 0) = R$ resources, using edges e_j, $j = 1, \ldots, k$ of C with arbitrarily recharged resources μ_j, and ending in node v. The cost of this walk is $c(W_C)$. Then, C is called a *regenerating cycle* iff there is a choice of recharged resources μ_j such that $r(W, k) > r(W, 0)$ and $c(W_C) < f_v(r(W, k)) - f_v(r(W, 0))$.

In other words, requiring a network without regenerating cycles implies that is always cheaper to recharge at a charging node immediately, instead of taking detours for recharging and coming back. Without charging, this definition would be equivalent to the definition of conservative resource functions. In accordance, we call the resource function and the charging functions *strictly conservative* if no regenerating cycles occur in the network.

Checking for such cycles is a difficult problem since there is an exponential number of cycles generally. Assuming non-linear charging functions, it is already $\mathcal{NP}$-hard to compute the optimal μ_i at each charging node on a single cycle since we may have to deal with a nonconvex nonlinear programming problem [16].

▶ **Theorem 8.** *In a network without regenerating cycles a shortest walk visits a charging node at most once.*

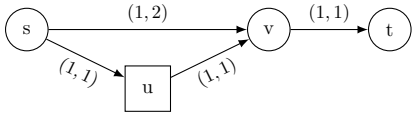


Figure 3 Assume $R = \overline{R} = 2$. Hence, an s-t-walk has to visit u for recharging. Obviously, the s-v-subwalk is not an optimal s-v-walk, since v can be reached directly from s with lower costs.

Proof. Definition 7 implies that visiting a charging node twice always causes additional costs (compared to charging at this node directly) or a loss of resources compared to the first visit. ◀

Please note that regenerating cycles are created by an interplay of resource functions and charging functions. Even in a network where all charging functions are identical, regenerating cycles may occur. Furthermore, identical charging functions do not imply either that it is optimal to completely charge the battery when a charging node is visited. Still, a path that visits several charging nodes and recharges small amounts of μ_i may be better than a path visiting fewer charging stations.

Finally, as shown next walks with charging do not have the shortest subwalk property.

▶ **Lemma 9.** *An s-v-subwalk of a shortest s-t-walk with charging is not necessarily a shortest s-v-walk.*

Proof. Figure 3 provides an instance without the subwalk property. Same claim also holds for constrained shortest paths and CSP is a subproblem of our problem. ◀

At first sight, the results in this section may seem of academic interest only. But charging functions for electric vehicles are non-linear. Consequently, such effects cannot be generally excluded in practice. Hence, each algorithm designed for this problem should be prepared to cope with cycles or one has to provide appropriate restrictions that are consistent with the corresponding application.

4 Approximating Constrained Shortest Walks with Charging

We will now develop an FPTAS for the constrained shortest walk problem with charging nodes. This will consist of two steps. Firstly, we construct an FPTAS for the common constrained shortest path problem but with conservative resource consumption and without charging at intermediate nodes. Secondly, we use a slightly modified version of the first algorithm as a sub-routine in an FPTAS connecting charging nodes in a network without regenerating cycles.

4.1 The Inner Approximation Algorithm

The inner approximation algorithm picks up the main idea of Hassin's approach, namely scaling and rounding [13], but we make some important changes to cope with negative resource consumption.

Let us fix two terminal nodes $\hat{s}$ and $\hat{t}$. Furthermore, we are given initial resources $\hat{R}$ and upper bound $\overline{R}$, a non-negative cost function $c : E \to \mathbb{R}_{\geq 0}$, and a conservative resource consumption $r : E \to \mathbb{R}$. In this inner approximation we do not consider recharging, i.e., there are no charging nodes. In consequence, no cycles can occur in an optimal path. Now, we want to find an $\hat{\varepsilon}$-approximation of the shortest s-t-path with respect to the cost function

c obeying the resource constraints. In other words, we want to find a feasible path that has cost at most $(1+\hat{\varepsilon})$-times higher than the optimal path.

To record information about subpaths, we provide a set of labels at each node. Each label consists of two values (c, r) of costs and remaining resources. Feasibility is crucial, hence, resource consumption has to be calculated exactly. Consequently, we can only round the cost values in the approximation algorithm. Here, the main difficulty is to find a suitable precision for rounding. On the one hand, the number of different values has to be bounded polynomially. On the other hand, we have to meet the approximation factor $1+\hat{\varepsilon}$.

Suppose that we would know the value OPT of the objective function of an optimal solution already. In this case, we can round up all cost values on the edges to integral multiples of $\frac{\hat{\varepsilon}\,\mathrm{OPT}}{|V|-1}$. Since the optimal path has at most $|V|-1$ edges, and the error on each edge is at most $\frac{\hat{\varepsilon}\,\mathrm{OPT}}{|V|-1}$, the total error of the path is at most $\hat{\varepsilon}\,\mathrm{OPT}$. If we round the cost values with this precision, we also get only integer multiples of $\frac{\hat{\varepsilon}\,\mathrm{OPT}}{|V|-1}$ as possible cost values for subpaths at each node. Since the length OPT of an optimal path is known and costs are non-negative, it suffices to record cost values up to this bound $(1+\hat{\varepsilon})\,\mathrm{OPT}$. Thus, we have at most $\frac{(|V|-1)(1+\hat{\varepsilon})}{\hat{\varepsilon}}$ different cost values that can occur at a node. Of course, for each node v we only store the set $Q(v)$ of Pareto-optimal labels, i.e., the highest battery charge achieved so far for each of the possible cost values.

To find a path of the desired length, we initially label node $\hat{s}$ with $Q(\hat{s}) = \{(0, \hat{R})\}$. All other label sets $Q(v)$ are initially empty. In an update step, we propagate the whole label set of each node u to all its neighbors v. Each label in $Q(u)$ is updated using cost and resource consumption of $e = (u, v)$ and added to the label set $Q(v)$ accordingly. Labels with cost values higher than $(1+\hat{\varepsilon})\,\mathrm{OPT}$ are discarded, they can never contribute to a path with the desired properties. Since an optimal path can consist of at most $|V|-1$ edges, precisely this number of update rounds suffices to find an optimal $\hat{s}$-$\hat{t}$-path according to Bellman-Ford's algorithm. Setting pointers for each label pointing to the predecessor node, the path itself can easily be reconstructed.

Thus, given a value OPT, we can check whether there is a feasible $\hat{s}$-$\hat{t}$-path of length at most $(1+\hat{\varepsilon})\,\mathrm{OPT}$ in time polynomial in $|V|$ and $\frac{1}{\hat{\varepsilon}}$. However, the OPT value is unknown at the beginning, so we apply binary search to find it. Let LB be the cost of a shortest path without considering the resource constraints. Thus, LB is a lower bound on OPT. Further, compute a shortest path with respect to resource consumption. Since resource consumption is conservative, one may use Bellman-Ford's algorithm. Obviously, the cost UB of this path is an upper bound on OPT. Our first guess on OPT is $\overline{\mathrm{OPT}} = \sqrt{\mathrm{LB}\cdot\mathrm{UB}}$, i.e., we use a logarithmic scale.

If we find a feasible path with length smaller than $\overline{\mathrm{OPT}}$, we can use this length as a new upper bound UB. If we cannot find a path of length at most $(1+\hat{\varepsilon})\overline{\mathrm{OPT}}$, we can use $\overline{\mathrm{OPT}}$ as a new lower bound LB. $\overline{\mathrm{OPT}}$ is updated accordingly and the binary search is stopped when $\frac{\mathrm{UB}}{\mathrm{LB}}$ is smaller than a pre-defined constant k, e.g., $k=2$. Now, we execute a final run with rounding precision $\frac{\hat{\varepsilon}\,\mathrm{LB}}{|V|-1}$ and maximum cost value UB. In other words, we use the precision given by the lower bound LB, but we use k-times as many labels to cover paths of length UB.

Summarizing, Algorithm 1 is a very short description of the inner approximation.

Algorithm 1 Inner Approximation.

Input: Graph $G = (V, E)$ with c, r and $\hat{\varepsilon}$, nodes $\hat{s}$ and $\hat{t}$, $Q(\hat{s}) = \{(0, \hat{R})\}$
Output: $\hat{s}$-$\hat{t}$-path with cost $\leq (1 + \hat{\varepsilon})\,\mathrm{OPT}$
compute LB, and UB, and $\overline{\mathrm{OPT}}$
while $\frac{UB}{LB} > 2$ **do**
    initialize Q, round c
    for $i = 1, \ldots, |V| - 1$ **do**
        for all nodes $v \in V$ **do**
            propagate $Q(v)$ to all neighbors
        end for
    end for
    update LB or UB, and $\overline{\mathrm{OPT}}$
end while
execute final run with double precision
if $Q(\hat{t}) \neq \emptyset$ **then**
    reconstruct $\hat{s}$-$\hat{t}$-path, return $Q(\hat{t})$
else
    no such path exists
end if

4.2 The Outer Approximation Algorithm

Whereas the inner approximation can be used for calculating costs and battery consumption between charging nodes, the outer approximation combines these paths between charging stations to create a walk from start node s to target node t.

The network for this algorithm consists only of the charging nodes S and any two nodes of S are connected if there is a feasible path between them in the original graph. However, reachability and the cost thereof depend on the initial battery charge. Thus, it is not possible to compute this reachability graph a priori. Instead, the cost for going from one charging node to another has to be computed just when needed.

Let $(1 + \varepsilon)$ be the accuracy of the outer approximation, we again determine the optimal rounding precision via a binary search on the optimal value $\mathrm{OPT}_{\mathrm{outer}}$ as described in Section 4.1. Only nodes $v \in S \cup \{s, t\}$ are labeled, initially all label sets are empty but $Q(s) = \{(0, R)\}$. To propagate a label from $u \in S \cup \{s\}$ to $w \in S \cup \{t\}$, we call the inner approximation with $\hat{s} = u$, $\hat{t} = w$, and $Q(\hat{s}) = Q(u)$.

In contrast to the pure inner approximation described above, we can omit the inner binary search by using the $\mathrm{OPT}_{\mathrm{outer}}$ value from the outer binary search and by setting $\hat{\varepsilon} = \frac{\varepsilon}{|S+1|}$. Thus, we also compute paths with higher costs but less consumption which match the required accuracy of the outer approximation. Note that this may lead to a poor actual approximation of the subpath. Especially if the subpath is very short, then an error of $\hat{\varepsilon}\,\mathrm{OPT}_{\mathrm{outer}}$ can be much larger than $\hat{\varepsilon}\,\mathrm{OPT}$. However, this relative error on subpaths does not matter as long as the total error of all subpaths is bounded.

Now, one has the option to charge the battery in node w. We use the return value $Q(\hat{t})$ of the inner approximation (where $\hat{t} = w$) and calculate all possible battery charges that match the cost discretization. Let f_w be the charging function at w, $(\hat{c}, \hat{r}) \in Q(\hat{t})$ and α the rounding precision of the outer approximation. We determine all values of x such that $0 \leq f(\hat{r} + x) - f(\hat{r}) + \hat{c} = k\alpha \leq (1 + \varepsilon)\,\mathrm{OPT}_{\mathrm{outer}}$ with $k \in \mathbb{N}$. Each label $(\hat{c}, \hat{r})$ of $Q(w)$ is shifted by all those values and is added to $Q(w)$ forming a new Pareto-optimal label set. Here, it is assumed implicitly that all operations concerning the computation of f can be done in polynomial time.

Since the resource function r is assumed to be strictly conservative, each charging node is visited at most once. We may also apply Bellman-Ford's principle here. Propagating the label set of each $u \in S \cup \{s\}$ to any other $w \in S \cup \{t\}$ in $|S| + 1$ rounds creates a

Algorithm 2 Outer Approximation.

Input: Graph $G = (V, E)$ with $S \subseteq V$, c, r and ε, nodes s and t, R
Output: s-t-path not longer than $(1 + \varepsilon)\,\mathrm{OPT}$
compute LB, and UB, and $\overline{\mathrm{OPT}}$
while $\frac{UB}{LB} > 2$ **do**
    initialize Q, $Q(s) = \{(0, R)\}$, round c
    for $i = 1, \ldots, |S| - 1$ **do**
        for all nodes $u \in V \cup \{s\}$ **do**
            for all nodes $w \in V \cup \{t\}$ **do**
                determine Pareto-optimal u-w-paths with maximum cost $\mathrm{OPT}_{\mathrm{outer}}$ and precision $\hat{\varepsilon} = \frac{\varepsilon}{|S+1|}$, and initial label set $Q(u)$
                apply recharging in w
                update $Q(w)$
            end for
        end for
    end for
    update LB or UB, and $\overline{\mathrm{OPT}}$
end while
if $Q(t) \neq \emptyset$ **then**
    reconstruct s-t-path, return $Q(\hat{t})$
else
    no such path exists
end if

correct label at t. Since we use at most $|S| + 1$ subpath and each subpath has at most error $\hat{\varepsilon}\,\mathrm{OPT} = \frac{\varepsilon}{|S+1|}\,\mathrm{OPT}$, the total error is less than $\varepsilon\,\mathrm{OPT}$.

Thus, in a very condensed form Algorithm 2 states the outer approximation scheme.

The outer approximation runs in time polynomial in $|S|$ and ε and the number of function calls to the inner approximation is also bounded polynomially. Due to the finer precision $\hat{\varepsilon} = \frac{\varepsilon}{|S+1|}$, the running time of the inner algorithm is increased by at most $\mathcal{O}(n)$ but the inner binary search is not needed. Please note that a feasible path for determining UB can be easily found by applying a full recharge at each charging node and checking reachability only for maximal resources. Due to space constraints, we have to omit a more detailed analysis but refer the reader to the journal version of this paper.

5 Flows with Recharging

Finally, we study the network flow version of constrained shortest paths in a network with charging nodes. In this setting, we are looking for a network flow that has a path decomposition such that each path is feasible with respect to the resource consumption. Hence, the constrained shortest path problem can be seen as a sub-problem. For example, the flow is zero iff no feasible path exists. Consequently, the flow problem is at least as difficult as the path problem.

We look at the unweighted version of this problem, here. That means, we only have capacities $u : E \to \mathbb{R}_{\geq 0}$ and resource consumption but no costs on the edges. The capacities limit the maximal flow on each edge. We also change the interpretation of charging nodes. A charging node now also has a capacity $c : S \to \mathbb{R}_{\geq 0} \cup \{+\infty\}$ which states the maximum amount of resources that can be provided for charging. One may also use a different kind of charging function to capture different efficiency factors during charging, i.e., the charging station has to expend more energy than arrives in the battery. In practice, this difference depends on the charge level of the battery. For simplicity, we assume a linear relation in this paper.

As seen in Section 3, cycles may occur. Since walks are the main actors, we use a walk based flow definition. Here, the parameter $\lambda_e(K)$ counts the number of occurrences of edge

e in a walk K. Further, $\mathcal{K}_{s,t}$ denotes the set of all possible s-t-walks. Note that due to the walk based formulation flow conservation is automatically implied.

▶ **Definition 10.** An *s-t-flow* in this setting is a function $f : \mathcal{K}_{s,t} \to \mathbb{R}_{\geq 0}$ assigning a *flow value* f_K to each s-t-walk K in G. The sum $\sum_{K \in \mathcal{K}_{s,t}} f_K$ is called the *s-t-flow value of f*. The flow f is *feasible* if it respects edge capacities, i.e.,

$$\sum_{K \in \mathcal{K}_{s,t} : e \in K} \lambda_e(K) f_K \leq u_e \qquad \forall e \in E.$$

Given initial resources R and resource bound $\overline{R}$, a resource constrained *s-t-flow with charging* is an s-t-flow, where every walk K with $f(K) > 0$ is a feasible s-t walk with charging with respect to R and $\overline{R}$.

An s-t-walk K with charging that charges μ units at node $v \in S$ and which is used by $f(K)$ flow units uses $f(K)\mu$ units of the capacity $c(v)$.

▶ **Definition 11.** An s-t-flow with charging is *feasible*, if the underlying s-t-flow is feasible and all walks with $f(K) > 0$ charge in total at most $c(v)$ units in node v for all $v \in S$.

▶ **Definition 12.** Given initial resources R and resource bound $\overline{R}$, an *s-t-cut with charging* is a set of edges $E' \subseteq E$ such that there is no feasible s-t-walk with charging in $G = (V, E \backslash E')$. The accumulated capacity of the edges in E' is called the *cut value* of E'.

Now, we consider maximum s-t-flows with charging, that is, flows where the flow value is maximum among all feasible s-t-flows with charging. Even in such maximum flows it can be necessary that some path visits a certain charging node more than once.

▶ **Corollary 13.** *There are instances of the maximum s-t-flows with charging problem, where a path contributing to a maximum s-t-flow with charging has to visit a charging node more than once.*

Proof. Consider the network in Figure 2. Let all edge capacities and $c(w)$ be infinite. Hence, the flow is only restricted by $c(u)$. If we visit node u only once, there is only one feasible path and at most $\frac{c(u)}{\overline{R}}$ units can be sent from s to t. If we recharge 3ε per flow unit, visit node w for a full recharge, and go back to u for another charge of 3ϵ, we can send $\frac{c(u)}{6\varepsilon}$ units from s to t. Thus, if we choose $\varepsilon < \frac{\overline{R}}{6}$, the claim follows. ◀

But even without limiting rechargeable resources, i.e., $c(v) = +\infty$ for all $v \in S$, flows with recharging significantly differ from common network flows. The network in Figure 1 implies two more corollaries for resource constrained flow with charging.

▶ **Corollary 14.** *The gap between the flow value and the cut value of resource constrained flow in a network with charging nodes and unlimited supply at charging nodes can be of order $\Omega(n)$, even for planar networks with unit capacities.*

Proof. Add unit capacities to the graph in Figure 1. Since there is only one feasible path, and this path uses one edge $\Omega(n)$-times, the claim follows. ◀

▶ **Corollary 15.** *The gap between resource constrained fractional flow and integral flow with charging in a network with unlimited supply in charging nodes is unbounded.*

Furthermore, there exist networks that do not allow for an integral flow greater than zero despite all capacities being integral. Nevertheless, the total fractional flow value can be integral and positive in those networks.

Proof. Figure 1 with unit capacities provides an instance where no positive integral flow is possible. Using several copies of this network in a parallel manner and scaling capacities would allow a fractional flow of value 1. ◀

6 Discussion

In this paper, we studied shortest paths and flows in a resource constrained setting where it is possible to refill resources at some nodes. We considered charging costs which depend on the charged amount. Thus, finding good paths is not only a question of reachability, but we have also to decide where and how much to charge. This additional combinatorial variety makes the problem significantly more difficult. Thinking of applications like routing of electric vehicles it seems very challenging to run an FPTAS on an onboard unit in admissible time.

This suggests further research into two directions. On the one hand, one may work on alternative approximation algorithms, e.g., based on routing algorithms in a condensed resource-expanded network (cf. [8]). On the other hand, the cycle constraints are crucial. Are there other ways to control regenerating cycles? Checking consumption and charging functions for being strictly conservative is difficult and expensive. Is it possible to perform this check using some kind of cycle basis instead of the whole set of cycles?

One may also extend the problem. For example, one may ask for the shortest path if no more than $(1 + \alpha)$ of the resources of the most resource efficient path can be spent. Further, not only the charging functions may depend on the initial value of the battery, but also the consumption itself may differ for full or nearly empty batteries. If this cannot be handled in a preprocessing step which parameterizes the battery linearly, this will lead to some kind of dynamic shortest path problem with recharging.

Another open question is the existence of an FPTAS for maximum flows with charging. Such an FPTAS can make use of the FPTAS for s-t-walks with charging. For example, one may try an approach like the algorithm of Garg and Könemann [11] with the extension of Fleischer [9] to approximative shortest paths. However, one will need consistent dual cost functions for the recharge capacities of the charging nodes. Thinking of fleets of vehicles, one may also extend the problem to include a time component. That is, charging stations only provide a limited number of slots for charging vehicles simultaneously. Therefore, recharging also requires scheduling.

References

1 Georg Baier. *Flows with Path Restrictions*. PhD thesis, TU Berlin, 2003.

2 Georg Baier, Thomas Erlebach, Alexander Hall, Ekkehard Köhler, Heiko Schilling, and Martin Skutella. Length-bounded cuts and flows. In *Automata, Languages and Programming*, LNCS 4051, pages 679–690. Springer Berlin Heidelberg, 2006.

3 Moritz Baum, Julian Dibbelt, Lorenz Hübschle-Schneider, Thomas Pajor, and Dorothea Wagner. Speed-consumption tradeoff for electric vehicle route planning. In *Proceedings of the 14th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'14)*, OpenAccess Series in Informatics (OASIcs), pages 138–151, 2014.

4 Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Energy-optimal routes for electric vehicles. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 54–63. ACM Press, 2013.

5 John E. Beasley and Nicos Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19(4):379–394, 1989.

6 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning. In P. M. Pardalos and S. Rebennack, editors, *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, LNCS 6630, pages 376–387. Springer, 2011.

7 Geoffrey Exoo. On line disjoint paths of bounded length. *Discrete Mathematics*, 44(3):317–318, 1983.

8 Lisa Fleischer and Martin Skutella. Quickest flows over time. *SIAM Journal on Computing*, 36(6):1600–1630, 2007.

9 Lisa K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000.

10 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

11 Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

12 Gabriel Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980.

13 Refael Hassin. Approximation schemes for the restricted shortest path problem. *Math. Oper. Res.*, 17(1):36–42, February 1992.

14 László Lovász, Víctor Neumann-Lara, and Michael Plummer. Mengerian theorems for paths of bounded length. *Periodica Mathematica Hungarica*, 9(4):269–276, 1978.

15 Kurt Mehlhorn and Mark Ziegelmann. Resource constrained shortest paths. In Mike S. Paterson, editor, *Algorithms – ESA 2000*, LNCS 1879, pages 326–337. Springer Berlin Heidelberg, 2000.

16 Katta G. Murty and Santosh N. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39(2):117–129, 1987.

17 Ludovít Niepel and Daniela Šafaříková. On a generalization of Menger's theorem. *Acta Mathematica Universitatis Comenianae*, 42:275–284, 1983.

18 Cynthia A. Phillips. The network inhibition problem. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC'93, pages 776–785, New York, NY, USA, 1993. ACM.

19 Martin Sachenbacher, Martin Leucker, Andreas Artmeier, and Julian Haselmayr. Efficient energy-optimal routing for electric vehicles. In *Conference on Artificial Intelligence, Special Track on Computational Sustainability*. AAAI, 2011.

20 Sabine Storandt. Quick and energy-efficient routes: computing constrained shortest paths for electric vehicles. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 20–25. ACM, 2012.

21 Sabine Storandt and Stefan Funke. Enabling e-mobility: Facility location for battery loading stations. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. AAAI Press, 2013.

22 Arthur Warburton. Approximation of pareto optima in multiple-objective, shortest-path problems. *Operations Research*, 35(1):70, 1987.

23 Mark Ziegelmann. *Constrained shortest paths and related problems*. Phd thesis, Universität des Saarlandes, Saarbrücken, 2001.

Heuristic Approaches to Minimize Tour Duration for the TSP with Multiple Time Windows

Niklas Paulsen[1,2], Florian Diedrich[2], and Klaus Jansen[1]

1 **Institut für Informatik, Christian-Albrechts Universität zu Kiel, Christian-Albrechts-Platz 4, 24118 Kiel, Germany**
{npau,kj}@informatik.uni-kiel.de
2 **FLS GmbH, Schlosskoppelweg 8, 24226 Heikendorf, Germany**

Abstract

We present heuristics to handle practical travelling salesman problems with multiple time windows per node, where the optimization goal is minimal tour duration, which is the time spent outside the depot node. We propose a dynamic programming approach which combines state labels by encoding intervals to handle the larger state space needed for this objective function. Our implementation is able to solve many practical instances in real-time and is used for heuristic search of near-optimal solutions for hard instances. In addition, we outline a hybrid genetic algorithm we implemented to cope with hard or unknown instances. Experimental evaluation proves the efficiency and suitability for practical use of our algorithms and even leads to improved upper bounds for yet unsolved instances from the literature.

1998 ACM Subject Classification I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases TSPTW, minimum tour duration, dynamic programming, heuristics

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.42

1 Introduction

The `Travelling Salesman Problem with Time Windows (TSPTW)` is the problem of finding a cost-minimal Hamiltonian cycle through a complete digraph on N nodes, which respects time windows given for each node. The nodes are represented by the set $V = \{0, \dots, N-1\}$, where 0 is called the *depot*, and we define $V' := V \setminus \{0\}$ to be the other nodes. Each node $v \in V'$ has a given time window $[a_v, b_v]$ in which it has to be visited. To calculate times, $c : V \times V \to \mathbb{N}$ assigns a travel time to each edge. Arriving at a node v before a_v will lead to waiting there until the time window opens. Node dependent visit times can be encoded in the travel times; distinct start and return nodes can be combined into the node 0 by adjusting travel times from and to node 0; missing arcs can be encoded by high travel times. The time windows constrain the set of feasible solutions; in general the presence of time windows makes it NP-hard to even find a feasible tour [12]. A possible generalization from TSP to TSPTW minimizes the same objective function, namely the sum of weights of the chosen edges [4, 10]. Since we have edge weights as travel times, this objective corresponds to minimization of the total travel time. However, in workforce planning, loans make a major contribution to the planned costs and thus also waiting times are expected to have an impact on the objective function. An important choice is whether a delayed start of a given tour is counted as working time or not. If not, any tour can be started at some earliest possible time. Then, minimizing the total travel time plus any waiting time along the way corresponds to finding the tour which has the earliest return to the depot node. We call this problem `Minimum Completion Time Problem (MCTP)`. In this work, however,

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt; pp. 42–55

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

we allow the start to be delayed without cost, searching a tour that can be traversed with minimal tour duration from depot departure to return. We call the according problem `Minimum Tour Duration Problem (MTDP)`. This objective function is a generalization of the completion time minimization, since the latter can be expressed by fixing the starting time at the depot [3]. While for mathematical formulations, the difference between `MCTP` and `MTDP` is just a slight change in the objective function, common heuristic approaches as well as Dynamic Programming (DP) become more complicated for the latter. Modifying a given tour with known optimal departure time, a new optimal departure time needs to be searched, as Savelsbergh [13] pointed out. Despite its relevance for workforce planning, the `MTDP` has been given only little attention until recently. Tilk et al. [15] treated the `MTDP`, using a new Dynamic Programming (DP) based approach to solve many available instances to optimality or provide bounds. Although they are more focused on optimal solution rather than real-time processing in a practical use case, their solutions can serve as a good reference. We handle an even further generalized version of the problem, allowing an arbitrary number of time windows per node, as it occurs in practice for example at machine-related maintenance tasks or simply due to opening hours with lunch breaks. This complicates the search of an optimal departure time for a given tour [1].

Our focus is the development of fast algorithms for a practical workforce planning context. Our two methods are a DP based heuristic and a strongly randomized genetic algorithm. In practical workforce planning, we see multiple applications of fast heuristics for the `MTDP`:

- When planning only a single worker,
- as a frequently called local search in advance planning of big `Vehicle Routing Problems (VRPs)`,
- real-time post-optimization of planned tours after changes in online `VRP`s, and
- to obtain upper bounds that can be used by exact methods for small instances or for evaluation purposes.

In Section 2 we discuss the state-space inflation for DP inherent with allowing multiple time windows per node and propose an approach to encode multiple states into intervals. In Section 3 we outline our Genetic Algorithm, `GA`, which allows solving diverse instances like ones with very wide time windows. We report results on instance sets from the literature and on new real-world instances in Section 4 and give a final conclusion in Section 5.

1.1 Formal Definitions

We want to formalize the timings for given tours. Be $K_v > 0$ the number of time windows for $v \in V'$, $a_{v,k}$ and $b_{v,k}$ be the opening and closing time, respectively, for the k-th time window of node $v \in V'$, $0 \leq k < K_v$. The time windows of every node are presumed to be sorted, non-overlapping, and of non-negative length ($a_{v,0} \leq b_{v,0} < a_{v,1} \leq \cdots < a_{v,K_v-1} \leq b_{v,K_v-1}$ for $v \in V'$). Define Π to be the set of `TSP`-tours, represented by permutations of V, starting in the depot ($\pi(0) = 0$ for $\pi \in \Pi$). For an arrival time $t \in \mathbb{N}$ at a node $v \in V'$, the next feasible schedule time at that node is given by:

$$T^{\rightarrow}(v,t) := \min\{x \mid x \geq t \wedge \exists k < K_v : x \in [a_{v,k}, b_{v,k}]\}$$

For $t > b_{v,K_v}$ a minimum over $\emptyset$ leads to $T^{\rightarrow}(v,t) = \infty$. For a `TSP`-tour $\pi \in \Pi$ and departure time $t_0 \in \mathbb{N}$ the scheduled departure times $t^{\pi}_{t_0} : V \to \mathbb{N}$ can be calculated as follows: For the depot 0 it is $t^{\pi}_{t_0}(0) = t_0$ and for $v \in V'$ it is, depending on the last node visited, $v^- := \pi(\pi^{-1}(v) - 1)$:

$$t^{\pi}_{t_0}(v) := T^{\rightarrow}(v, t^{\pi}_{t_0}(v^-) + c(v^-, v))$$

Define $t^{\pi}_{t_0}(N) := t^{\pi}_{t_0}(\pi(N-1)) + c(\pi(N-1), 0)$ to be the returning time at the depot. Furthermore we define $W^{\rightarrow}(v,t) := T^{\rightarrow}(v,t) - t$ for the waiting time at node $v \in V'$, when reached at time t. The optimization goal is then to find $\pi \in \Pi$ and $t_0 \in \mathbb{N}$ minimizing $t^{\pi}_{t_0}(N) - t_0$.

▶ **Lemma 1.** *For $\pi \in \Pi, i < N, \tau, \delta \in \mathbb{N}$ we have $t^{\pi}_{\tau+\delta}(\pi(i)) \geq t^{\pi}_{\tau}(\pi(i))$. (Proof in Appendix)*

2 Adaption of Dynamic Programming for Tour Duration Minimization

A common way to solve various variants of TSPs is via dynamic programming (DP), based on the formulation for the classic TSP proposed decades ago by Bellman et al. [2]. Bellman's Principle states, generically speaking, that optimal solutions of a problem (instance) are consisting of optimal solutions to smaller sub-problems. We call sub-problems *states* and their solution a *label* of the state. The proceeding of forward-labelling is to label some initial states and use recurrence relations to propagate given labels to labels for other states.

In case of the TSP, sub-problems are finding a minimum cost path originating in 0, going through a given subset of nodes, $S \subset V'$, and ending in a given node, $\ell \in V' \setminus S$. We call these paths S, ℓ-paths. The calculation can be tackled in n stages, for increasing $|S|$ according to longer paths. A minimum path through a given $S \neq \emptyset$ can only be arising from an optimal path through $S \setminus \{x\}$ to $x \in S$, but with the presence of time windows, this only holds for minimizing tour completion time (MCTP). To solve TSPTW regarding minimum travel time, a two-dimensional labelling for (S, ℓ)-states is necessary, as used by Dumas et al. [4]. This is because all S, ℓ-paths are relevant that are Pareto optimal concerning cost and completion time, since during calculation it is not known which time at ℓ can lead to a feasible completion of the tour through $V' \setminus (S \cup \{\ell\})$. Recently, DP was adapted for the MTDP (with single time windows per node) by Tilk et al. [15]. They use labels containing 3 *resources* for the earliest possible time to complete an S, ℓ-path, the tour duration so far, and a time slack. Labelling all (non-dominated) S, ℓ-paths, Bellman's Principle holds. In the generalized case of an arbitrary number of time windows for each node, every S, ℓ-path extended to an $S \cup \{\ell\}, \ell'$-path must distinguish the times at which it is travelled: Compared to the time leading to a minimal duration of the S, ℓ-path, an earlier or later traversal with a *longer* duration may bring along a smaller waiting time at node ℓ', if a different time window of ℓ' can be taken possibly leading to a smaller tour duration. As a consequence, more labels can arise for every S, ℓ-path, corresponding to different choices of time windows for visited nodes. We show that the number of labels for each tour is growing at most linearly with the overall number of time windows. For a fixed tour $\pi \in \Pi$, the following definition is used to model a specific choice of time windows for a prefix of π.

Define a *time window path of length* $k \leq |V'| = N - 1$ to be a tuple $(s_1, \ldots, s_k)$ with $s_i < K_{\pi(i)}$ for $0 < i \leq k$ choosing time window indices for the first k nodes visited by π after the depot; we call it *schedule*, iff $k = |V'|$ and we call it *reachable*, iff a start time τ exists such that all nodes are visited within their chosen time window:

$$t^{\pi}_{\tau}(\pi(i)) \in [a_{\pi(i),s_i}, b_{\pi(i),s_i}] \quad \text{for } 0 < i \leq k. \tag{1}$$

Note that reachable time window paths are only met by delaying the departure at the depot while every other node is visited as early as possible by definition of t^{π}_{τ}.

▶ **Theorem 2.** *With a given TSP-Tour $\pi \in \Pi$ there are at most $1 + \sum_{v \in V'}(K_v - 1)$ reachable schedules.*

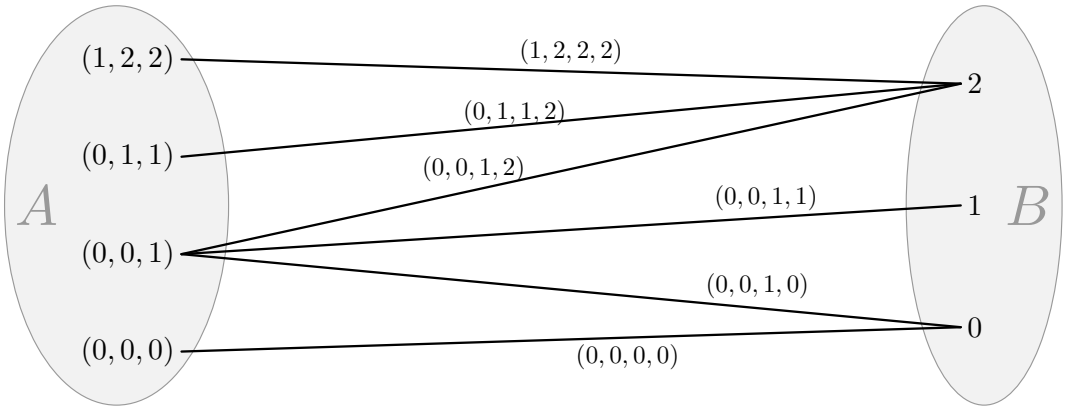


Figure 1 Illustration of an example bipartite graph H for $i = 3$.

Proof. We show via induction over $0 < i \leq |V'|$ that there are at most

$$1 + \sum_{j=1}^{i} (K_{\pi(j)} - 1) \tag{I}$$

reachable time window paths of length i. The induction base is $i = 1$, with the visit of node $\pi(1)$ in one of its $K_{\pi(1)}$ time windows.
Now assume (I) holds for an $0 < i < |V'|$. To prove that (I) also holds for $i + 1$ we need to show that only up to $K_{\pi(i+1)} - 1$ more time window paths of length $i + 1$ arise than for i. Define a bipartite graph $H = (A, B, F \subset A \times B)$ with $A \subseteq \mathbb{N}_{<K_{\pi(1)}} \times \cdots \times \mathbb{N}_{<K_{\pi(i)}}$ being the reachable time window paths of length i and $B = \mathbb{N}_{<K_{\pi(i+1)}}$ being the time window indices of the next node, $\pi(i + 1)$. An edge $f = ((c_1, \ldots, c_i), k) \in F$ shall exist, if and only if the time window path $c' := (c_1, \ldots, c_i, k)$ is reachable. Since every reachable time window path of length $i + 1$ contains a reachable time window path of length i, the edges correspond to the reachable time window paths of length $i + 1$. An example for H is shown in Figure 1.

We show that H can be drawn without crossings in the sense of Eades et al. [5]. We use the lexicographic order as $\prec_A$ on A, and the order $\prec_B$ on B which is given for the time windows by their definition. Suppose an edge (a, w) exists. Then, an edge (a', w') with $a \prec a'$ but $w' \prec w$ cannot exist, since this would mean that starting later[1] leads to reaching an earlier time window at waypoint $i + 1$, contradicting Lemma 1. This implies a crossing-free drawing of H, and thereby absence of cycles in H [5]. Being cycle-free, H is a forest and has at most $|A| + |B| - 1$ edges. With the induction hypothesis, $|A| \leq 1 + \sum_{j=1}^{i} (K_{\pi(j)} - 1)$, and with $|B| = K_{\pi(i+1)}$, (I) also holds for $i + 1$. ◀

Consider the state space $2^{V'} \times V' \times \mathbb{T}$ with $\mathbb{T} \subset \mathbb{N}$, where each state (S, ℓ, t) gets a scalar label expressing the minimum S, ℓ-path duration when ℓ is visited at time t. Even with a bounding of $\mathbb{T}$ to actually relevant times, the size of this state-space is wasteful, especially with the inherent growth with increasing temporal resolution of time encoding. Our approach is to encode for each S and ℓ the function assigning the minimal S, ℓ-path duration to each departure time t. An example of this function is shown in Figure 2 in the lower left. The

[1] With $a \prec a'$, at some point a later time window (bigger index) is taken with a', therefore starting times leading to reaching a' cannot be smaller than any starting time leading to reaching a (contraposition of Lemma 1).

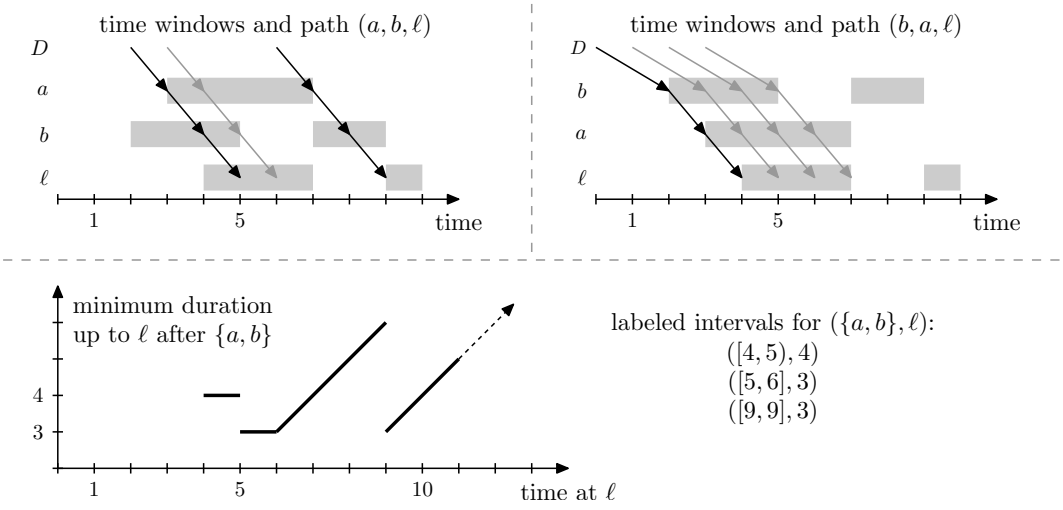


Figure 2 Example for labelled intervals, for State $(S = \{a, b\}, \ell)$. D is the depot, travelling time between the depot and b is two hours, all others one hour.

notion is a shift of the $\mathbb{T}$ factor from the state space into the labels of the (S, ℓ)-states. If an S, ℓ-path $\mathcal{P}$ can lead to a visit of ℓ at time t with (minimal) duration T, only the following cases apply for the minimal duration T' for the "next" time $t + 1$:
(C1) $\mathcal{P}$ can be traversed later without waiting times, leading to $T' = T$, (C2a) another path leads to minimal duration when visiting ℓ at $t + 1$, (C2b) $\mathcal{P}$ with another time window combination leads to minimal duration when visiting ℓ at $t + 1$, or (C3) $\mathcal{P}$ traversed later is optimal for $t + 1$ but leads to increased waiting times along the path, with $T' = T + 1$.

Therefore the function consists (except for undefined values of t, before the arrival of the first S, ℓ-path) only of piecewise constant parts (starting with cases C2a or C2b, continued with case C1) and piecewise linear parts with a slope of 1 (case C3). Our idea is to store only the interval and assigned tour duration of constant parts to implicitly encode the function. We can then handle multiple (S, ℓ, t)-states by working with the encoded intervals.

A labelled interval $I = ([t_s, t_e], T)$ is a non-empty interval $[t_s, t_e]$ $(t_e \geq t_s)$ of $\mathbb{N}$ and an assigned tour duration. For a given (S, ℓ)-state we use

$$\mathit{Ints} : S, \ell \longmapsto \text{set of } \textit{labelled intervals} \text{ encoding labels of } (S, \ell, \cdot)$$

to label (S, ℓ)-states. An interval corresponds to a constant part of the function of minimal S, ℓ-path duration at different times. With Bellman's Principle we can demand for $S \subsetneq V', \ell \in V' \setminus S$:

$$[t_s, t_e] \cap [t'_s, t'_e] = \emptyset \quad \text{f.a. } ([t_s, t_e], T) \neq ([t'_s, t'_e], T') \in \mathit{Ints}(S, \ell) \tag{2}$$

$$t'_s > t_e \Rightarrow t'_s > t_e + (T' - T) \quad \text{f.a. } ([t_s, t_e], T), ([t'_s, t'_e], T') \in \mathit{Ints}(S, \ell). \tag{3}$$

Clearly, disjoint intervals suffice: For state (S, ℓ) only the *minimum* tour duration to reach ℓ at a time t after having visited the nodes in S is needed. To evince (3) we show:

▶ **Lemma 3.** *Suppose $\ell \in V'$ can be visited at time t after all nodes in $S \subset V'$ with a tour duration of T_1, but also such that it is left until $t + \delta$ with tour duration $T_2 \geq T_1 + \delta$, for a $\delta > 0$. Then the latter is dominated by the former (it cannot lead to a tour with a smaller duration).*

No matter how the rest of the tour is constructed through $V' \setminus (S \cup \{\ell\})$, the forward propagation of the first state is able to reach the same time windows as the forward propagation of the second. Since the waiting times can only be larger by the lead δ, this will conduct at most the same tour duration (formal proof in Appendix).

To read out the label, i.e. the minimal tour duration, for a state (S, ℓ, t), we use a function *Cost* to interpret the set of labelled intervals $\mathcal{I} = Ints(S, \ell)$ at the time t:

$$Cost(\mathcal{I}, t) := \min_{\substack{([t_s,t_e],T)\in\mathcal{I} \\ t_s \le t}} T + \max\{0, t - t_e\} \tag{4}$$

With $\mathcal{I}$ satisfying Equations (2) and (3) we can write (proof in Appendix):

$$Cost(\mathcal{I}, t) = T + \max\{0, t - t_e\} \text{ for } ([t_s, t_e], T) = \operatorname*{arg\,max}_{\substack{([t_s,t_e],T)\in\mathcal{I} \\ t_s \le t}} t_e \tag{5}$$

This means, to evaluate the minimum tour duration at time t, only the last labelled interval starting before t needs to be considered, which can be retrieved efficiently when the labelled intervals are stored in suitable data structures.

The labelled intervals can be initialized by

$$Ints(\emptyset, \ell) = \{([a_{\ell,k}, b_{\ell,k}], c(0, \ell)) : k < K_\ell\} \tag{6}$$

Forward propagation can be done for aggregated times in intervals and time windows (a pseudocode can be seen in the Appendix). When multiple labels (as labelled intervals) occur for a state (S, ℓ), the intervals can be merged, choosing for each time t the interval with the best label and respecting Equation (3), as can be seen in Figure 2.

Heuristic Adaption. To heuristically reduce the search space for larger instances, Malandraki et al. [9] used a cutoff on the number of states to keep track of after each stage in their DP heuristic for the time dependent `TSP`. By retaining only the most promising H labels after each step, the run time can be minimized drastically. For $H = 1$ it resembles a *Nearest Neighbour Heuristic*, for $H = \infty$ the Dynamic Programming for an optimal solution is not affected. We call this approach `DPH` in the following. Note that we retain H labels, containing generally more than H intervals. In our implementation the labels are simply ranked by the minimal possible duration for each (S, ℓ)-State: $\min_t Cost(Ints(S, \ell), t)$.

Adapted Cost Function. It is an easy step to generalize the `DPH` to minimize a more generic objective function, being a weighted sum of working time and travelled distance.

Preprocessing and Trimming the Search Space. The search space can be trimmed by preprocessing the instance, see [3]. Also, when calculating the labelled intervals for a state (S, ℓ), only times need to be considered, which allow to reach all unvisited nodes $v \in V' \setminus (S \cup \{\ell\})$ before the end of their last time windows, b_{v,K_v-1}. Assuming the triangle-inequality (which holds often, especially with visit times present), an easy bound for relevant departure times at ℓ is:

$$B_{S,\ell} := \min\{b_{v,K_v-1} - c(\ell, v) : v \in V' \setminus (S \cup \{\ell\})\}. \tag{7}$$

Algorithm 1: Dynamic Programming for tour duration minimization.

1 $\mathcal{H} \leftarrow$ empty hashtable for labels assigned to S, ℓ-states;
2 label $(\emptyset, \ell)$ with $\{([a_{\ell,k}, b_{\ell,k}], c(0,\ell)) : k < K_\ell\}$ for $\ell \in V'$;
3 **for** *stage from* 1 *to* $N-2$ **do**
4 **for** *state* (S,ℓ) *with label* $\mathcal{I}$ *and* $|S| = stage - 1$ **do**
5 **for** $\ell' \in V' \setminus (S \cup \{\ell\})$ **do**
6 calculate new interval set $\mathcal{I}'$ by propagating $\mathcal{I}$ towards ℓ';
7 Trim interval ranges to be $\leq B_{S\cup\{\ell\},\ell'}$;
8 **if** $\mathcal{H}$ *contains label* $\mathcal{I}''$ *for* $(S \cup \{\ell\}, \ell')$ **then**
9 $\mathcal{H}(S \cup \{\ell\}, \ell') \leftarrow$ Merged intervals of $\mathcal{I}'$ and $\mathcal{I}''$;
10 **else**
11 $\mathcal{H}(S \cup \{\ell\}, \ell') \leftarrow \mathcal{I}'$;
12 retain only best H labels with $|S| =$ stage in $\mathcal{H}$;
13 **return** $\min_{\ell \in V'} \min_{([t_s,t_e],T) \in \mathcal{H}(V'\setminus\{\ell\},\ell)} T + c(\ell, 0)$;

2.1 Pseudocode

Algorithm 1 illustrates the principal `DPH` flow. States are expressed by a combined binary representation of S and ℓ. Order constraints between nodes are also saved in a binarily represented set of nodes that have to be visited before a given node. It can be checked with little computation whether all required nodes have been visited when extending toward a node ℓ' (not shown, line 5). The hashtable lookup in line 8 can be done very efficiently. The merge step in line 9 only takes time linear in the number of intervals to be merged. By iterating all times t_s, t_e for $([t_s, t_e], \cdot) \in \mathcal{I} \cup \mathcal{I}'$ in ascending order, one simply has to chose the minimal intervals between the times and trim them to fit Equation (3). Note that with merging labels each state gets at most one label. Backtracking information is included for every labelled interval.

3 A Genetic Algorithm

To find high-quality solutions for instances with arbitrary or unknown properties in real-time, we developed a genetic algorithm that builds and refines a set of solutions, called the population. It builds on the general concepts of genetic algorithms, like the one of Sengoku et al. [14]. In iterations called generations, *mutation* is trying to bring some randomly chosen solutions to near local optima, *selection* focuses the search by removing the least promising solutions from the population, and *multiplication* makes up for deletions by combining existing solutions into crossovers, in hope of finding new local optima. An initial population is generated with randomized `Insertion` heuristics inserting nodes iteratively at a position which is chosen with higher probabilities towards positions that lead to lower overall cost. For an additional start solution, the `DPH` is run with $H = 200$. Mutation of the population makes use of local search strategies on one third of the solutions picked randomly. The main local search is a repeated search in randomly chosen fixed-size subsets of 3-Opt [8] neighbourhoods. We experienced a randomized 3-Opt to be more effective than searching in the full neighbourhood of weaker local searches like 2-Opt. The fixed number of checked neighbours leads to execution times growing only about linearly in the number of nodes, for relevant instance sizes. The basic crossover operation is `CommonEdgesCrossover`, which

chooses randomly three parent solutions from the population and constructs a new tour by choosing randomly the edges to traverse. Edges occurring in more parent solutions are chosen with a higher probability and infeasible tours are prohibited if possible. Selection removes the worst fifth of the population, but is also allowed to eliminate solutions based on their affinity to the other solutions in the population to encourage diversity. The said affinity is valued by computing the longest common subsequence shared with some randomly picked solutions from the rest of the population. The population size and the number of generations can be set with a single aggregated parameter, γ, which controls the overall number of performed mutations, with $\gamma = 0$ leading to 750 mutations. For instances with more nodes, those mutations are deployed over more generations but with a smaller population, which we found to be more efficient. We suppose that this is due to bigger neighbourhoods with possibly more potential for bigger instances. The execution of the genetic algorithm can generally be stopped any time leading to the return of the best solution so far, which allows for interruption by users or timing. Infeasible solutions are tolerated but highly penalized: If for a tour $\pi \in \Pi$ and start time τ, a node $v \in V'$ is reached after its last time window, we correct adjust the timing to be ($v^- := \pi(\pi^{-1}(v) - 1)$):

$$t^\pi_\tau(v) = t^\pi_\tau(v^-) + c(v^-, v) + P_\infty,$$

where P_∞ is a *soft infinity* penalty, higher than any feasible tour duration. As a consequence, the tour duration increases by the number of nodes not yet visited ($N - \pi^{-1}(v)$) times P_∞. This allows to improve infeasible tours while always favouring tours that (feasibly) visit more nodes.

4 Experimental Results

The following experimental analysis was conducted based on the rationale of [7]. The test system is a Dell OptiPlex 980 equipped with 16 GB RAM and an Intel Core i7-880 CPU (8MB Cache, 3.06 GHz clock rate) running Windows 7. The algorithms were implemented in Microsoft C# 4.0 and compiled with Microsoft Visual Studio 2010. To evaluate the computation times and solution quality and provide comparable results, we use available instances from the literature and additional real-world instances to rate suitability for use.

We use the instances from `Gendreau` [6] and `Potvin+Bengio` [11][2] and processed them according to Tilk et al. [15]. The former are 120 instances with different parameters for node count (21–101) and width of time windows; the latter are 30 instances with 4 to 46 nodes. All instances have only single time windows per node. The other instances treated by Tilk et al. [15] were omitted since one set originates from a stacker crane context and the other one has instances with more than 126 nodes, for which the current `DPH` implementation is not capable (and which may arise for mobile workforce day trips only in very special circumstances).

Being interested in practicability of the algorithms for real-world scenarios, we adduce another test set consisting of 332 stops assigned to 17 tours with 16 to 24 nodes. The data originates from a logistics company bringing goods from and to collecting points within an urban area. Time windows in most cases resemble a full workday, a half workday, or a full one with a midday closure, with varying times. Around 30% of the nodes have a midday closure leading to two time windows. A fixed-time lunch break for the drivers has already

[2] Both sets downloaded from `http://iridia.ulb.ac.be/~manuel/tsptw-instances`.

Table 1 Aggregated results for different instance groups. (*)-marked averages are only taken over the found solutions. Stats for `GA` are reported as averages of 5 runs.

Instances	Program		#Feas.	#UB	#Imp	UBGAP [%] ∅	UBGAP [%] max	Time [s] ∅	Time [s] max
Gendreau small	DPH	H=1500	75	52	2	1.1	16.0	0.4	1.4
(75 instances)	DPH	H=5k	75	62	3	0.5	8.3	1.6	5.3
	DPH	H=15k	75	64	3	0.2	6.4	4.7	17.2
	GA	γ= -10	75	44	3	0.2	2.7	0.6	1.0
	GA	$\gamma = 0$	75	53	3	0.0	1.1	2.1	3.4
Gendreau big	DPH	H=1500	45	19	1	2.6	10.8	1.9	3.5
(45 instances)	DPH	H=5k	45	28	2	1.7	10.6	7.5	13.3
	DPH	H=15k	45	29	3	1.2	7.7	23.5	40.0
	GA	γ = -10	45	17	1	0.8	5.8	1.9	2.8
	GA	$\gamma = 0$	45	25	1	0.2	3.0	6.5	9.4
Potvin+Bengio	DPH	H=1500	29	19		1.6*	15.5	0.2	1.3
(30 instances)	DPH	H=5k	29	21		1.3*	13.7	0.8	5.3
	DPH	H=15k	28	21	1	0.8*	13.4	2.5	13.9
	GA	γ = -10	30	15	1	0.2	2.3	0.3	0.6
	GA	$\gamma = 0$	30	18	2	0.0	1.0	1.3	1.8
Real data	DPH	H=1500	17	5		3.4	11.2	0.2	0.3
(17 instances)	DPH	H=5k	17	9		1.6	10.9	0.7	1.5
	DPH	H=15k	17	10		1.0	9.6	2.6	5.5
	GA	γ = -10	17	15		0.1	0.8	0.6	1.1
	GA	$\gamma = 0$	17	17		0.0	0.0	2.6	4.5

been incorporated into the time windows. Compared to the other instances, time windows are rather broad (7:53 hours open spread over the day on average). Other properties differ from the simulated instances, like that 200 of 332 nodes have their first time window starting exactly at 8 o'clock, instead of time windows more randomly scattered around the day or simulated around a reference tour. The sum of travel costs assigned to edges and working time loans plus overtime fees are to be minimized.

Table 1 shows results for the Genetic Algorithm and the `DPH`, with different parameters H aggregating the instances according to Tilk et al. [15] *#Feas.* is the number of instances, for which a solution was found; *#UB* and *#Imp* the count of upper bounds (including optima) reported by Tilk et al. [15] which were hit exactly or improved, respectively. For the new instance set, optima were instead previously calculated using the `DPH` with $H = \infty$. Since, for real-time post-processing of tours, we are interested in good solutions rather than guaranteed optimality, the solution qualities for yet unsolved instances are reported in relation to the upper bounds from Tilk et al. [15], which we consider a good reference due to the overall quality of their algorithm (and effort expended for calculation). *UBGAP* for a given upper bound u and a solution value v is defined as $\frac{v-u}{u}$. Averages of *UBGAP* are over all respective instances, including those, where the upper bound was hit (zero gap) or improved (negative gap). Although the `DPH` leads to satisfying solution quality, it is outperformed by the `GA` on these instance sets, especially the real-world one with very wide time windows. However, this also depends on the composition of the test sets: Table 2 shows results on the `Gendreau`

Table 2 Aggregated results for `Gendreau` instances with narrow time windows. `GA*` is `GA` without the `DPH` start solution. Stats for `GA(*)` are reported as averages of 5 runs.

Instances	Program		#UB	#Imp	UBGAP [%]		Time [s]	
					∅	max	∅	max
Gendreau	DPH	H=1500	38	1	0.40	5.71	0.9	2.5
w120 + w140	DPH	H=5000	46	2	0.01	0.77	3.3	9.4
(50 instances)	GA	γ= -10	28		0.36	2.85	1.2	2.9
	GA	$\gamma = 0$	34	2	0.13	1.73	4.3	10.0
	GA*	γ = -10	22		0.54	2.94	1.0	2.7
	GA*	$\gamma = 0$	31	1	0.16	1.69	4.2	9.8

instances with rather tight time windows, for which the `DPH` solves the instances very close to optimality within seconds. The `GA` is noticeably weaker, and when ran without its additional start solution from `DPH` (marked with `*`), even more.
Regarding execution times, not only for instances presented here, we found that the `GA` has a running time roughly linear in the number of nodes (tested up to 200) that grows by about 15% when incrementing the parameter γ by one. The running times of `DPH` are varying stronger and also expectedly depend on the time window width.
We conclude that a combination of both algorithms is promising, for example by running `DPH` on visibly easier (few nodes and/or tight time windows) instances.

Improving Upper Bounds of Unsolved Instances

We ran the `DPH` on the instances with no more than 101 nodes that have not been solved to optimality yet. These are 17 instances with 36 to 101 nodes from the `Gendreau` and `Potvin+Bengio` instance sets. Running the `DPH` with increasing parameter $H \in \{10^3, 10^5, 10^6\}$ (stopping, if the lower bound was reached) we tried to improve the upper bounds reported by Tilk et al. [15]. Detailed results are shown in Table 3. *LB* and *UB* are the bounds previously reported. *Time* is the sum of the execution times in case multiple parameters were run. For the two `Potvin+Bengio` instances we also ran the `GA`, since the `DPH` seemed less effective. For five instances, the upper bound was met exactly and for ten it was improved (values underlined in Table 3), with an average improvement of 2.44%. In five cases our upper bound equals the known lower bound, so that those instances are now solved to optimality. The other five reduced the gap of the best upper bound to the best lower bound by more than half, on average.

5 Conclusion

We presented two algorithmic concepts to treat `TSP`s with (multiple) time windows for which the tour duration is to be minimized, like it is common in many areas of mobile workforce planning. The DP approach is based on aggregating state labels into efficient data structures by encoding intervals of times. It can be used to seek (and prove) optimal solutions, but also as a heuristic for harder instances. Our genetic algorithm applies local searches in a strongly randomized manner leading to good solution qualities, even with very broad time windows. It has a simple parameter to balance running time and (expected) solution quality and can be interrupted, e.g. for online problems, if the input needs to be modified. Both

Table 3 Results on open instances with up to 101 nodes, all times in seconds.

Instance	n	[LB,UB]	Result for $H =$ 1K	100K	1M	for GA	Time [s]	Improved UB [%]
n40w200.5	41	[347,350]	347				1	0.9
n60w180.5	61	[466,486]	501	466			150	4.1
n60w200.3	61	[497,525]	497				1	5.3
n80w140.2	81	[588,591]	592	589	589		765	0.3
n80w140.3	81	[615,617]	632	617	617		912	
n80w140.4	81	[549,561]	583	550	550		962	2.0
n80w160.2	81	[603,609]	637	629	629		1659	
n80w160.3	81	[633,638]	676	651	633		2061	0.8
n80w160.5	81	[583,584]	627	584	584		1771	
n80w180.2	81	[564,570]	615	591	570		2059	
n80w180.5	81	[570,571]	573	571	571		2210	
n80w200.1	81	[559,584]	620	566	564		2809	3.4
n80w200.2	81	[549,550]	603	582	560		2901	
n100w120.2	101	[843,846]	843				2	0.4
n100w140.2	101	[948,949]	954	949	949		1722	
rc_208.1	38	[73432,79904]	-	-	83683	79348	524	0.7
rc_208.3	36	[61302,67902]	84723	64499	64123	63436	1604	6.6

implementations are applicable for practical real-time optimization and post-processing. Computational results showed that very satisfying solutions can be found with minimal computing times. We even improved the best solution reported so far for 10 out of the 17 unsolved instances from the `Gendreau` and `Potvin+Bengio` benchmarks.

Acknowledgements. The authors would like to thank Thomas Brechtel for many fruitful discussions. Furthermore, the authors would like to thank the anonymous referees for many helpful suggestions which led to the improvement of the presentation.

References

1 Slim Belhaiza, Pierre Hansen, and Gilbert Laporte. A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows. *Computers & Operations Research*, 52:269–281, 2014.

2 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962.

3 Jacques Desrosiers, Yvan Dumas, Marius M Solomon, and François Soumis. Time constrained routing and scheduling. *Handbooks in operations research and management science*, 8:35–139, 1995.

4 Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations research*, 43(2):367–371, 1995.

5 Peter Eades, Brendan D McKay, and Nicholas C Wormald. On an edge crossing problem. In *Proc. 9th Australian Computer Science Conference*, volume 327, page 334, 1986.

6 Michel Gendreau, Alain Hertz, Gilbert Laporte, and Mihnea Stan. A generalized insertion heuristic for the traveling salesman problem with time windows. *Operations Research*, 46(3):330–335, 1998.

7 David S Johnson. A theoretician's guide to the experimental analysis of algorithms. *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges*, 59:215–250, 2002.

8 Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal, The*, 44(10):2245–2269, 1965.

9 Chryssi Malandraki and Robert B Dial. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research*, 90(1):45–55, 1996.

10 Gilles Pesant, Michel Gendreau, Jean-Yves Potvin, and Jean-Marc Rousseau. On the flexibility of constraint programming models: From single to multiple time windows for the traveling salesman problem. *European Journal of Operational Research*, 117(2):253–263, 1999.

11 Jean-Yves Potvin and Samy Bengio. The vehicle routing problem with time windows part II: genetic search. *INFORMS journal on Computing*, 8(2):165–172, 1996.

12 Martin WP Savelsbergh. Local search in routing problems with time windows. *Annals of Operations research*, 4(1):285–305, 1985.

13 Martin WP Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA journal on computing*, 4(2):146–154, 1992.

14 Hiroaki Sengoku and Ikuo Yoshihara. A fast TSP solver using GA on Java. In *Third International Symposium on Artificial Life, and Robotics (AROB III'98)*, pages 283–288, 1998.

15 Christian Tilk and Stefan Irnich. Dynamic programming for the minimum tour duration problem. Technical Report LM-2014-04, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany, 2014.

A Appendix: Proofs

Proof of Lemma 1. We show the claim by induction over i; fix $\tau, \delta \in \mathbb{N}$.
The equality $t^\pi_{\tau+\delta}(\pi(0)) = \tau + \delta \geq \tau = t^\pi_\tau(\pi(0))$ yields the induction base.
For the induction step, let $0 < i < N$, $v := \pi(i)$, $v^- = \pi(i-1)$ and assume

$$t^\pi_{\tau+\delta}(v^-) \geq t^\pi_\tau(v^-). \tag{H}$$

Then it follows, by definitions of $T^{\rightarrow}$ and $t^\pi_{t_0}$ and (H):

$$\begin{aligned}
t^\pi_{\tau+\delta}(v) &= T^{\rightarrow}(v, t^\pi_{\tau+\delta}(v^-) + c(v^-, v)) \\
&= \min\{x \mid x \geq t^\pi_{\tau+\delta}(v^-) + c(v^-, v) \wedge \exists k < K_v : x \in [a_{v,k}, b_{v,k}]\} && (T^{\rightarrow}) \\
&\geq \min\{x \mid x \geq t^\pi_{\tau} \quad (v^-) + c(v^-, v) \wedge \exists k < K_v : x \in [a_{v,k}, b_{v,k}]\} && (\mathrm{H}) \\
&= T^{\rightarrow}(v, t^\pi_{\tau}(v^-) + c(v^-, v)) && (T^{\rightarrow}) \\
&= t^\pi_\tau(v) && \blacktriangleleft
\end{aligned}$$

Proof of Lemma 3. Construct states $s1 := (S, \ell, t)$ and $s2 := (S, \ell, t + \delta)$ according to the supposition.

Case 1). It is clear that for $S \cup \{\ell\} = V'$, state $s1$ leads, by extension towards the depot, to a TSP-Tour with tour duration $T_1 + c(\ell, 0)$ which dominates the extension of $s2$ to the depot, concluding a TSP-Tour with duration $T_2 + c(\ell, 0) \geq T_1 + c(\ell, 0)$.

Case 2). For $S \cup \{\ell\} \subsetneq V'$, fix an arbitrary $\ell' \in V' \setminus (S \cup \{\ell\})$ and regard the forward propagation of $s1$ and $s2$ towards ℓ', leading to labelling of states $s1'$ and $s2'$, respectively.
State $s1' = (S \cup \{\ell\}, \ell', T^{\rightarrow}(\ell', t + c(\ell, \ell')))$ is labelled with $T_1' = T_1 + c(\ell, \ell') + W^{\rightarrow}(\ell', t + c(\ell, \ell'))$.
$s2' = (S \cup \{\ell\}, \ell', T^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')))$ is labelled with $T_2' = T_2 + c(\ell, \ell') + W^{\rightarrow}(\ell', t + \delta + c(\ell, \ell'))$.
Set $\delta' := T^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')) - T^{\rightarrow}(\ell', t + c(\ell, \ell'))$. Then:

$$\begin{aligned}
T_2' - T_1' &= T_2 + c(\ell, \ell') + W^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')) \\
&\quad - (T_1 + c(\ell, \ell') + W^{\rightarrow}(\ell', t + c(\ell, \ell'))) \\
&= T_2 - T_1 + W^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')) - W^{\rightarrow}(\ell', t + c(\ell, \ell')) \\
&\geq \delta + W^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')) - W^{\rightarrow}(\ell', t + c(\ell, \ell')) \\
&= \delta + T^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')) - (t + \delta + c(\ell, \ell')) \\
&\quad - T^{\rightarrow}(\ell', t + c(\ell, \ell')) + (t + c(\ell, \ell')) \\
&= T^{\rightarrow}(\ell', t + \delta + c(\ell, \ell')) - T^{\rightarrow}(\ell', t + c(\ell, \ell')) = \delta'
\end{aligned}$$

The initial situation is reiterated. Since S is of increasing cardinality this iteration converges to Case 1). ◀

Proof of Equation 5. We prove that Equation 5 follows from Equation 4, if (2),(3) hold. It is to be shown that of the labelled intervals from $\mathcal{I}$ with $t_s \leq t$, the one with maximal t_e (uniquely defined with (2) holding) also maximizes $T + \max\{0, t - t_e\}$. This is clear, if there is only one labelled interval in $\mathcal{I}$ with $t_s \leq t$. Otherwise, fix two distinct labelled intervals $([t_s, t_e], T), ([t_s', t_e'], T') \in \mathcal{I}$ with $t_s, t_s' \leq t$. With (2), one of them is earlier, say $t_e < t_e'$ and $t_e < t_s'$. With (3) we have $t_s' > t_e + (T' - T)$. This leads to:

$$\begin{aligned}
T' + \max\{0, t - t_e'\} &\leq T' + \max\{0, t - t_s'\} && (t_e' \geq t_s') \\
&= T' + t - t_s' && (t_s' \leq t) \\
&< T + t - t_e && (t_s' > t_e + (T' - T)) \\
&= T + \max\{0, t - t_e\} && (t_e < t_s' \leq t)
\end{aligned}$$

◀

B Appendix: Additional Pseudocode

Propagation of labelled intervals. The forward propagation of labels is shown in Algorithm 2. Adjusting the intervals to conform to Equation 3 is omitted here. We write $\mathcal{I}[i]$ for the i-th labelled interval of a sorted set $\mathcal{I}$ of labelled intervals, and write a labelled interval i as $([i.t_s, i.t_e], i.T)$.

Algorithm 2: Propagation of labelled intervals.

Data: Labelled intervals $\mathcal{I}$ for (S, ℓ) satisfying equations (2) and (3),
Travel time $c = c(\ell, \ell')$ from node ℓ to next node $\ell' \in V' \setminus (S \cup \{\ell\})$.
Result: $\mathcal{I}'$: Propagated intervals $\mathcal{I}$ towards node ℓ'.

$i \leftarrow 0$;
for k *from 0 to* $K_{\ell'} - 1$ **do**
  while $i < |\mathcal{I}| - 1$ *and* $\mathcal{I}[i+1].t_s + c \leq a_{\ell',k}$ **do**
    i++;
  if $i \geq |\mathcal{I}|$ **then**
    break;
  if $\mathcal{I}[i].t_e + c < a_{\ell',k}$ **then**
    Add $([a_{\ell',k}, a_{\ell',k}], \mathcal{I}[i].T + a_{\ell',k} - \mathcal{I}[i].t_e)$ to $\mathcal{I}'$;
    i++;
  while $i < |\mathcal{I}|$ *and* $\mathcal{I}[i].t_s + c \leq b_{\ell',k}$ **do**
    $t'_s \leftarrow \max(\mathcal{I}[i].t_s + c,\ a_{\ell',k})$;
    $t'_e \leftarrow \min(\mathcal{I}[i].t_e + c,\ b_{\ell',k})$;
    Add $([t'_s, t'_e], \mathcal{I}[i].T + c)$ to $\mathcal{I}'$;
    i++;

Single Source Shortest Paths for All Flows with Integer Costs*

Tadao Takaoka

Department of Computer Science, University of Canterbury
Christchurch, New Zealand
tad@cosc.canterbury.ac.nz

Abstract

We consider a shortest path problem for a directed graph with edges labeled with a cost and a capacity. The problem is to push an unsplittable flow f from a specified source to all other vertices with the minimum cost for all f values. Let $G = (V, E)$ with $|V| = n$ and $|E| = m$. If there are t different capacity values, we can solve the single source shortest path problem t times for all f in $O(tm + tn \log n)$ time, which is $O(m^2)$ when $t = m$. We improve this time to $O(\min(t, cn)m + cn^2)$, which is less than $O(cmn)$ if edge costs are non-negative integers bounded by c. Our algorithm performs better for denser graphs.

1998 ACM Subject Classification E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases information sharing, shortest path problem for all flows, priority queue, limited edge cost, transportation network

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.56

1 Introduction

We consider a network optimization problem such that each edge has two quantities associated, that is, cost and capacity. We want to maximize a flow from a specified source vertex s to a destination vertex v with minimum cost. Here we have two objectives; flow amount and path cost. Both cannot be optimized at the same time. Let us call the minimum cost path the shortest path. We need to compute the shortest path for the given flow value f for all possible f. An example is the routing problem in a train network. Suppose f passengers want to travel together in a group from a station specified as the source vertex s to the destination station expressed by vertex v. On the way they may need to change trains at several stations. The capacity of an edge corresponds to the remaining number of seats on the train and the cost corresponds to the fare. Let d be the cost of a path from s to v and f be the flow (unsplittable) from s to v. The pair (d, f) is called a *df*-pair.

Another example is a computer network. Here vertices correspond to hub computers and edges correspond to the links. Capacities are band-widths and flows are packet sizes to be sent. It is regarded as better if packets are transmitted together to prevent packet loss and recovery.

If $d \leq d'$ and $f \geq f'$, (d, f) is better than or equal to (d', f'), the latter being redundant and represented by the former. Otherwise, excluding the opposite case, they are incomparable. We only need to compute incomparable *df*-pairs.

* This research was partially supported by Optali : Optimization and its Applications in Learning and Industry, funded by EU and New Zealand Government.

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt; pp. 56–67

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Similar problems in the literature are the multi (bi)-objective shortest path problem [11] and the minimum cost flow problem [1]. In the former, the two objectives are similar; additive costs over paths. In our problem, they are cost and capacity. In the latter, the flow can be split over several paths to minimize the cost. In our model, a flow cannot be divided. Unsplittable flow is studied in a few papers such as [3] and [9], in which flow amounts are considered and costs are not.

The network optimization model in this paper is simple enough to be used by network practitioners, but to the author's knowledge there is no algorithmic or theoretical analysis on this model in the literature apart from the recent [15], [16] and [14]. [15] improves the second term in the complexity of $O(tm + tn \log n)$, the first remaining $O(tm)$. This paper improves the first term. [16] and [14] deal with the all pairs shortest path for all flows (APSP-AF) problem. The complexity in this paper is better in certain combinations of t and c.

The algorithmic technique is viewed as information sharing described in [18], which solves the all pairs shortest path problem efficiently. More specifically, for a graph with n vertices, the single source shortest path problem is solved n times by changing the source n times, where they share common resources obtained in advance as preprocessing or during the course of computation. In our problem, we solve the single source shortest path problem for all flow amounts simultaneously utilizing some common data structures.

To prepare for the later development, we describe the single source shortest path problem for all flows (SSSP-AF) in the following. Let $G = (V, E)$ be a directed graph where $V = \{v_1, \cdots, v_n\}$ and $E \subseteq V \times V$. Let $|E| = m$. The cost and capacity of edge (u, v) is a non-negative real number denoted by $cost(u, v)$ and a positive real number $cap(u, v)$ respectively. We specify a vertex, s, as the source. A shortest path from s to vertex v is a path such that the sum of edge costs of this path is the minimum among all paths from s to v. The minimum cost is also called the shortest distance. The single source shortest path problem (SSSP) is to compute shortest paths from s to all other vertices.

The bottleneck (value) of a path is the minimum capacity of all edges on the path. The bottleneck of the pair of vertices (u, v) is the maximum bottleneck of all paths from u to v. Such a path is called the bottleneck path from u to v. The single source bottleneck path (SSBP) problem is to compute the bottleneck paths from s to all vertices v. The bottleneck from s to v is the maximum flow value of a simple path from s to v. Those two problems are well studied. For the bottleneck path problem the readers are referred to [13] for single source and [10] for all pairs.

If we send a smaller unsplittable flow from s to v, there may be a shorter path from s to v. Thus it makes sense to compute the shortest paths from s to v for all possible flows for all vertices v, which is called SSSP-AF. We compute a tuple of pairs (d, f), called a *df*-pair, for each v where d is the shortest distance of a path that can push f to v.

▶ **Example 1.** An example graph with solutions is given in Figure 1 with v_1 as the source. A pair of cost and capacity is attached to each edge. Also *df*-pairs are attached to each vertex as worked solutions with d's and f's in increasing order. For example, we have (3, 1)(6, 3)(7, 4) at vertex v_4. This means if we want to carry the flow amount of 4, we need to take the path of (v_1, v_3, v_4) at the cost of 7. If the flow is 3, we have a cheaper path of (v_1, v_2, v_3, v_4). If we push just 1, the path (v_1, v_2, v_4) costs us 3. If we want to push 2, this flow amount is missing from the *df*-pairs at v_4. The *df*-pair (6, 3) covers this case, meaning there is no cheaper route than for flow 3. *df*-pairs are worked out for other vertices as well. The *df*-pair (6, 3) at v_4 is obtained from (4, 3) at v_3 and the label on edge (v_3, v_4), which is $(2, 5)$, that is, 6=4+2 and 3=min{3, 5}.

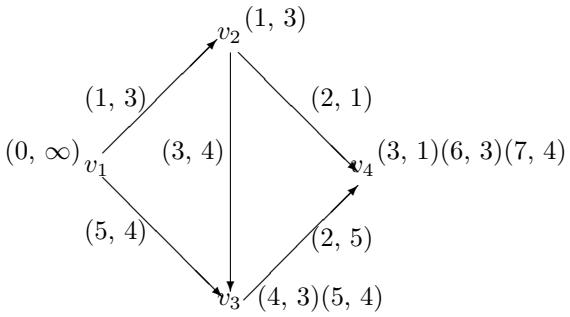


Figure 1 An example graph with incomparable *df*-pairs.

The problem can be solved by removing edges one by one. Suppose there are t different capacity values $cap_1, \ldots, cap_t$ in this order. The simplest algorithm looks like:

---

for $i = 1$ to t do begin
  Remove edges whose capacity is less than cap_i
  For the flow f such that $cap_i \leq f < cap_{i+1}$,
    solve the single source shortest path problem for the resulting graph.
end

If we use a Fibonacci heap [12] for Dijkstra's algorithm [8], the complexity of this algorithm becomes $O(tm + tn \log n)$, including the time for sorting capacities. This trivial upper bound is our starting point. The above algorithm works for edge costs of non-negative real numbers. In this paper, we improve the complexity when edge costs are non-negative integers bounded by a small positive constant c, achieving $O(\min\{t, cn\}m + cn^2) \leq O(cmn)$. The trivial complexity above is strongly polynomial, while our complexity is pseudo-polynomial. The point here is that we have a speed-up when c is small. When $t = O(m)$ and $m = O(n^2)$, the trivial complexity hits $O(n^4)$, called quartic, while our complexity $O(cn^3)$ can stay sub-quartic when $c = o(n)$. Similar studies are done on the all pairs shortest path problem (APSP) with integer edge costs such as [2], [17] and [20], who investigated up to what value of c we can stay in sub-cubic for the APSP complexity. The best bound for such c is $O(n^{0.624})$ if we use the Coppersmith-Winograd matrix multiplication algorithm. Recent studies improve this bound slightly.

The rest of the paper is as follows: In Sections 2 and 3, SSSP and SSBP are described in a pedagogical way so that we can see how they can be combined to solve the SSSP-AF problem. In Section 4, SSSP-AF is solved with the data structure of a one dimensional bucket system. The computational complexity of the approach described in this section is already known [15]. In Section 5, we improve the complexity in Section 4 by introducing another data structure and enhancing the one-dimensional bucket system. This section is the major contribution of the paper. In Section 6, we define the single source bottleneck path for all costs (SSBP-AC) problem. Although we can design an algorithm for this problem on its own, we show the problem can be solved as a by-product of the algorithm in Section 5. Section 7 concludes the paper. We use up-right fonts for some long names of variables and functions for readability.

2 Single source shortest path problem

We describe Dijkstra's algorithm [8] below in our style. The set S, called the solution set, is the set of vertices to which the shortest distances have been finalized by the algorithm. The

set F, called the frontier set, is the set of vertices which is outside S and can be reached from S by a single edge. We note that the distances to vertices in F can be limited to a small band when edge costs are bounded by a small integer.

Let $OUT(v) = \{w|(v,w) \in E\}$. The solution (the shortest distances from s) is in the array d at the end of the computation. To simplify presentation, only the shortest path distances are calculated, not the shortest paths. We assume all vertices are reachable from the source. Paths are given by a sequence of vertices such that for two successive vertices u and v, there is an edge (u,v). We list two invariants maintained by Algorithm 1 below.

(1) S is the set of vertices v to which shortest distances are worked out in $dist[v]$.
(2) If v is in F, $dist[v]$ is the distance of the shortest path that lies in S except for the end point v itself.

▶ **Lemma 2.** *The invariants* (1) *and* (2) *are kept through Algorithm 1.*

Proof. Lemma is true before while. Suppose (1) is true immediately after line 4. If there is a shorter path to v after line 5 via another vertex, say u, in F, which must exist to reach v, then $dist[u]$ is shorter than $dist[v]$, which is a contradiction. After v is included in S, all w in F or in $V - S - F$ are updated with the smallest possible $dist[w]$. Thus (2) is preserved. ◀

Throughout the paper, comments are given in the pseudo codes of the algorithms by the double slash for readability.

Algorithm 1

```
1.  S = ∅
2.  dist[s] = 0; dist[v] = ∞ for all v ≠ s
3.  F = {s}
4.  while F is not empty do begin
5.    v =delete-min(F) // with key dist[v]
6.    S = S ∪ {v}
7.    for w ∈ OUT(v) do
8.      if w ∉ S then
9.        if w ∈ F then dist[w] = min{dist[w], dist[v] + cost(v,w)} //decrease-key
10.         else begin dist[w] = dist[v] + cost(v,w); F = F ∪ {w} end // insert
11. end
```

At the end of computation F becomes empty and S becomes V, giving the solution in $dist$. We use a simple data structure of one-dimensional bucket system with array Q. $Q[i]$ is a list of items whose key value is i. Items in our case are vertices. We observe delete-min or delete-max operations can be done in $O(cn)$ time in total where cn is the size of Q, and decrease-key or increase-key, and insert can be done in $O(1)$ time per operation.

▶ **Theorem 3.** *Algorithm 1 solves the SSSP in* $O(m + cn)$ *time* [7].

Proof. We use a one-dimensional bucket system for the priority queue following Dial's idea [7], where total delete-min takes $O(cn)$ time and each insertion and decrease-key takes $O(1)$ time. Suppose there are m_i edges from vertex v_i. Summation of $m_i O(1)$ gives the result, where $m = m_1 + \ldots + m_n$. ◀

3 Single source bottleneck path problem

We modify Algorithm 1 slightly for the single source bottleneck path problem. Note that we can push flow f from s to v through a path whose bottleneck value is f. The bottleneck path is sometimes called the widest path, where the capacity of an edge is viewed as the width. The solution set S and frontier set F are similarly defined.

(1) S is the set of vertices v to which maximum flows are worked out in $flow[v]$.
(2) If v is in F, $flow[v]$ is the flow of the path with the maximum flow to v that lies in S except for the end point v itself.

We use array "$flow$" instead of "$dist$" in the following. Capacities, which are non-negative real numbers, are sorted and normalized to integers $1, \ldots, t, t+1$, where there are t different capacity values in the graph and $t+1$ represents ∞. Note that $t \leq m$.

Algorithm 2

```
1.  S = ∅
2.  flow[s] = t + 1; flow[v] = 0 for all v ≠ s
3.  F = {s}
4.  while F is not empty do begin
5.    v =delete-max(F) // with key flow[v]
6.    S = S ∪ {v}
7.    for w in OUT(v) do
8.      if w ∉ S then
9.        if w ∈ F then flow[w] = max{flow[w], min{flow[v], cap(v, w)}} //increase-key
10.         else begin flow[w] = min{flow[v], cap(v, w)}; F = F ∪ {w} end // insert
11. end
```

▶ **Lemma 4.** *Invariants* (1) *and* (2) *are kept through the iteration in the while loop.*

Proof. Omitted. ◀

▶ **Theorem 5.** *After normalization of capacities, Algorithm 2 solves the SSBP in* $O(m+t) = O(m)$ *time.*

Proof. Omitted. ◀

4 Single source shortest paths for all flows

We parameterize Dijkstra's algorithm with the flow value f. Array $dist[v]$ is extended to $dist[v, f]$ whose intuitive meaning is the distance of the shortest path that can push f to v. The solution set S is extended to $S(f)$, meaning the solution set for the SSSP for the flow value f. Data structure Q is used for F such that items (v, f) are kept in the list at $Q[dist[v, f]]$, that is, $dist[v, f]$ is the key. The idea is to solve $t+1$ SSSP's in parallel with the shared data structure Q.

Algorithm 3

Main data structures

$dist[v, f]$: currently shortest distance of path from source s to vertex v that can push flow f.

Q : a one-dimensional array of lists of items (v, f). If $Q[d]$ includes (v, f), $dist[v, f]$ is d. In each list the same vertex may appear more than once with different f. This part will be improved in Algorithm 4.

Pointer array indexed by (v, f) : $pointer[v, f]$ points to item (v, f) in Q. Capacities are normalized to $1, \ldots, t, t + 1$.

```
1.  S[f] = ∅ for f = 1,...,t,t+1 // t+1 is for infinity
2.  dist[s,f] = 0 for f = 1,...,t,t+1; Other dist are initialized to ∞
3.  Q[0] = {(s,t+1)}
4.  while Q is not empty do begin
5.    (v,f) =delete-min(Q) // with key dist[v,f]
6.    S[f] = S[f] ∪ {v}
7.    for w in OUT(v) do begin
8.      d* = dist[v,f] + cost(v,w) // candidate distance for w
9.      f* = min{f, cap(v,w)} // candidate flow for w
10.     if w is not in S[f*] then
11.       if (w,f*) ∈ Q then dist[w,f*] = min{dist[w,f*],d*} //decrease-key
12.         else begin dist[w,f*] = d*; Q = Q ∪ {(w,f*)} end // insert
13.   end
14. end
```

The following lemma is obvious.

▶ **Lemma 6.** *In Algorithm 3, we have* $f \leq f' \Rightarrow S(f) \supseteq S(f')$

We establish two assertions similar to those in the previous sections.

(1) For all v in $S[f]$, $dist[v, f]$ is the distance of the shortest path from s to v that can push flow f.

(2) For all (v, f) in Q, $dist[v, f]$ is the distance of the shortest path from s to v that can push flow f whose vertices are in $S[f]$ except for the end point v.

▶ **Lemma 7.** *The above invariants* (1) *and* (2) *are kept through iterations by the while-loop.*

Proof. The proof is based on induction overthe while-loop. Before the while-loop (1) and (2) are obviously true. Suppose there is a shorter path to v via u in Q that can push flow f at line 5. This means $dist[u, f] < dist[v, f]$, which is a contradiction to line 5 that chose $dist[v, f]$ as minimum. To push flow f^* via v after v is included in $S[f]$, we update all (w, f^*) in Q with possible shorter distances via v, or include (w, f^*) if it was outside Q with the new distance via v. Note that $S(f^*) \supseteq S(f)$ from Lemma 6, meaning the path in $S(f)$ for (w, f^*) is included in $S(f^*)$ except for w. Thus at the end of one iteration (2) is preserved. ◀

▶ **Theorem 8.** *Algorithm 3 solves SSSP-AF in* $O(tm + cn)$ *time* [15].

Proof. The correctness is seen from the fact that at the end of the algorithm the set $S(f)$ includes all v to which flow f can be pushed. The time is analysed from delete-min and decrease-key/insert. The former takes $O(cn)$. The latter takes $O(tm)$, because each vertex v_i joins $S(f)$'s at most t times and decrease-key/insert takes $O(tm_i)$ for each v_i, where

$|OUT(v_i)| = m_i$, resulting in $O(tm)$ over summation on i. Note that all (v, f) in Q are distinct so that we have at most t such (v, f)'s in Q for each v. ◀

The following monotone property is obvious and can be used for obtaining the solution, i.e., incomparable *df*-pairs in increasing order for each vertex.

▶ **Lemma 9.** *It holds for finalized distances that* $f \leq f' \Rightarrow dist[v, f] \leq dist[v, f']$.

From this lemma, we can list up incomparable *df*-pairs in increasing order for each v.

// $L[v]$ is the container of the solution for v. "||" is to append a pair to the list.
$L[v] = \phi; dist[v, 0] = \infty$ for all v
for each v do
  for $f = 1$ to t do
    if $dist[v, f] < \infty$ and $dist[v, f] \neq dist[v, f-1]$ then $L[v] = L[v]||(dist[v, f], f)$

In [4] and [6], the simple one-dimensional bucket system is generalized to the k-level cascading bucket system for SSSP. The following is a very brief sketch of the data structure. Readers interested in algorithm structures may skip to the end of this section.

▶ **Example 10.** An example of a 3-level radix-10 bucket system is given below. The initial list of keys is (7, 31, 34, 38, 56, 78, 113, 456, 477, 812, 1256, 1279).
$Base = 0, a_0 = 7, Q[0] = (\phi, \phi, \phi, \phi, \phi, \phi, \phi, (7), \phi, \phi)$
$Base = 0, a_1 = 3, Q[1] = (\phi, \phi, \phi, (31, 34, 38), \phi, (56), \phi, (78), \phi, \phi)$
$Base = 0, a_2 = 1, Q[2] = (\phi, (113), \phi, \phi, (456, 477), \phi, \phi, \phi, (812), \phi, \phi, \phi, (1256, 1279))$
After delete-min, key 7 at level 0 is deleted and for the next next delete-min, list (31, 34, 38) is re-distributed to level 0, resulting in
$Base = 30, a_0 = 1, Q[0] = (\phi, (31), \phi, \phi, (34), \phi, \phi, \phi(38), \phi)$
$Base = 0, \ \ a_1 = 5, Q[1] = (\phi, \phi, \phi, \phi, \phi, (56), \phi, (78), \phi, \phi)$
$Base = 0, \ \ a_2 = 1, Q[2] = (\phi, (113), \phi, \phi, (456, 477), \phi, \phi, \phi, (812), \phi, \phi, \phi, (1256, 1279))$
After deleting 31 for delete-min suppose we decrease key 477 to 59, that will go out of level 2 and join key 56 at level 1. Due to the nature of Dijkstra's algorithm, it will not go to a lower level than $Q[0]$.

Now the initial key value $d[v] = cost(s, v)$ is given like a radix-p number, where only x_{k-1} may exceed $p - 1$.

$$d[v] = x_{k-1}p^{k-1} + \ldots + x_1 p + x_0 \quad (0 \leq x_0, x_1, \ldots, x_{k-2} \leq p-1, \\ 0 \leq x_{k-1} \leq \lceil c/p^{k-1} \rceil - 1) \text{ for some } k.$$

The data structure has k levels of buckets. At the i-th level for each i, there are p buckets. Let i be the largest index of non-zero x_i. Item v is inserted into the x_i-th bucket at level i for all v in the frontier. During the computation, we maintain the items in the appropriate buckets based on the current value of $d[v]$. Let a_i, called the active pointer, be the smallest index of a non-empty bucket in level i. The role of a_i is to skip many empty buckets at level i. The base for level i, B_i, and the range for the j-th bucket at level i, R_j, are defined by

$$B_i = a_{k-1}p^{k-1} + \ldots + a_{i+1}p^{i+1}, R_j = [B_i + jp^i, B_i + (j+1)p^i - 1]$$

If item v is in level i for $d[v] = x_{k-1}p^{k-1} + \ldots + x_1 p + x_0$, i is the largest index such that $a_{k-1} = x_{k-1}, \ldots, a_{i+1} = x_{i+1}$ and $a_i \neq x_i$. Items move from a higher level to a lower

level and from a higher bucket to a lower bucket in the same level. The minimum can be found by scanning for a non-empty level and then the first non-empty bucket. For delete-min, the keys in this bucket are re-distributedto lower levels, finally creating a non-empty bucket at level 0.. Decrease-key can be done by moving the item in the data structure. Insert can be done by putting the item at the largest level and follow decrease-key. Suppose we solve t SSSP's. In [18], it is shown that t SSSP's can be solved in $O(tm + tn \log(c/t))$ time with this data structure. In [18], the data structure is used for the all pairs shortest path problem, where $t = n$, achieving the complexity of $O(mn + n^2 \log(c/n))$. We can use the data structure for the SSSP-AF problem where t SSSP's are solved in $O(tm + tn \log(c/t))$ time. This complexity is good when c is large with a better second term, but when t is large, the first term of $O(tm)$ is outstanding. The same thing can be said of Thorup's data structure [19] that spends $O(tm + tn \log \log c)$ time when applied to our problem. We try to improve the first term in the next section.

We note at this stage that in the list $Q[d]$ for some d in the one-dimensional system, there might be items (v, f) and (v, f') such that $f \neq f'$ for some v. The following section is to prevent this duplication of items for the same v with more formalism.

5 A faster algorithm for SSSP-AF

▶ **Definition 11.** Natural order $\leq_n$ is defined on df-pairs, (d, f) and (d', f'), by

$$(d, f) \leq_n (d', f') \Rightarrow d \leq d' \wedge f \leq f'$$

Merit order $\leq_m$ is defined on (d, f) and (d', f') by

$$(d, f) \leq_m (d', f') \Rightarrow d' \leq d \wedge f \leq f'$$

The natural order represents a numerical order while the merit order specifies which is better for our objective. Note that both are partial orders.

▶ **Definition 12.** For v in $S[f]$, pair $(dist[v, f], f)$ is said to be Pareto optimal at v if there is no pair $(dist[v, f'], f')$ such that v is in $S(f')$ and $(dist[v, f'], f') >_m (dist[v, f], f)$. In other words, $(dist[v, f], f)$ is Pareto optimal if there is no better df-pair so far at v.

The priority queue Q is augmented by array $flow$, which is initialized to all 0. $flow[v, d]$ is the maximum flow so far from s to v with cost d. We maintain each list $Q[d]$ such that each v appears at most once in the list. If (v, f) is to be inserted to list $Q[d]$, where $d = dist[v, f]$, $flow[v, d]$ is consulted. If $f \leq flow[v, d]$, this insertion is ignored. If not, $(v, flow[v, d])$ is deleted from Q, (v, f) is inserted and $flow[v, d]$ is updated to f. Decrease-key(v, f) is to perform $delete(v, f)$ and $insert(v, f)$ with the new distance. We maintain pointers for each pair (v, f) to locate (v, f) in Q in $O(1)$ time. Delete-min takes $O(cn)$ time in total.

Algorithm 4

Main data structures

$dist[v, f]$: same as Algorithm 3

Q : a one-dimensional array of lists (buckets) of items (v, f). If $Q[d]$ includes (v, f), $dist[v, f]$ is d. In each list every vertex appears at most once.

$flow$: $flow[v, d]$ gives the maximum flow that can be pushed from s to v through a path in $S(f)$ except v with cost d. The size of array $flow$ is $O(cn^2)$.

Pointer array indexed by (v, f) : same as Algorithm 3

0. $dist[v, f]$ are initialized to ∞ for all $v \neq s$ and f
1. $S[f] = \emptyset$ for $f = 1, \ldots, t, t+1$; // $t+1$ is for infinity
2. $dist[s, f] = 0$ for $f = 1, \ldots, t, t+1$; $flow[v, d] = 0$ for all v and d
3. $Q[0] = \{(s, t+1)\}$
4. while Q is not empty do begin
5. (v, f) =delete-min(Q) // with key $dist[v, f]$
6. $S[f] = S[f] \cup \{v\}$
7. for w in $OUT(v)$ do begin
8. $d^* = dist[v, f] + cost(v, w)$ // candidate distance for w via v
9. $f^* = \min\{f, cap(v, w)\}$ // candidate flow for w via v
10. if w is not in $S[f^*]$ then
11. if (w, f^*) is in Q then begin
12. $dist[w, f^*] = \min\{dist[w, f^*], d^*\}$
13. decrease-key(w, f^*)
14. $flow[w, d^*] = \max\{flow[w, d^*], f^*\}$
15. end
16. else begin
17. $dist[w, f^*] = d^*$
18. insert(w, f^*)
19. $flow[w, d^*] = f^*$
20. end // if-else
21. end // for
22. end // while
23. procedure insert(w, f^*)
24. begin
25. if $f^* > flow[w, d^*]$ then begin
26. if $(w, flow[w, d^*])$ is in Q then delete$(w, flow[w, d^*])$
27. $Q = Q \cup (w, f^*)$ // insert with key $dist[w, f^*]$
28. end
29. end
30. procedure decrease-key(w, f^*)
31. begin delete(w, f^*); insert(w, f^*) end

The loop invariants (1) and (2) in the previous section hold for Algorithm 4 as well. In addition we have the following lemma, which is similar to (2).

▶ **Lemma 13.** *For all v and d, let $f = flow[v, d]$. If (v, f) is in Q, f is the maximum flow of the path with cost d that can push f from s to v whose vertices are in $S[f]$ except for the end point v.*

Proof. Suppose this invariant holds at the beginning of the while loop. After v is included

in $S[f]$, $flow[w, d^*]$ is updated at lines 14 and 19. Note that we have $f^* \leq f$ and thus $S[f^*] \supseteq S[f]$ from Lemma 6. Thus the path is in $S(f^*)$ except for the end point and the lemma holds for $f^* = flow[w, d^*]$ as well. ◀

▶ **Lemma 14.** *At each iteration of while loop, pair* $(dist[v, f], f)$ *is Pareto optimal for any* $(v, f) \in S[f]$ *at line 5.*

Proof. Suppose the statement is true at the beginning of each iteration. We perform one more iteration. Suppose $(dist[v, f], f)$ is not Pareto optimal for some v and f at the end of the iteration. Then for some $(dist[v, f'], f')$ we have $(dist[v, f'], f') >_m (dist[v, f], f)$, which means.

$$(dist[v, f'] < dist[v, f] \wedge f' \geq f) \vee (dist[v, f'] \leq dist[v, f] \wedge f' > f).$$

By a simple calculation, this is equivalent to

$$(dist[v, f'] < dist[v, f] \wedge f' \geq f) \vee (dist[v, f'] = dist[v, f] \wedge f' > f).$$

This contradicts the fact that $dist[v, f]$ is the distance of the shortest path that can push f to v, or the fact that (v, f) is updated (in the form of (w, f^*)) with the maximum possible f by consulting $flow[v, dist[v, f]]$. Lemma 13 guarantees f is the maximum flow to v with cost $dist[v, f]$. ◀

In the following lemma we abbreviate $(dist[v, f], f)$ as (d, f).

▶ **Lemma 15.** *All Pareto optimal df-pairs at any* v *can be sorted in increasing natural order. Furthermore if* $(d, f) \leq_n (d', f')$*, we have* $d < d'$ *and* $f < f'$*.*

Proof. If not sorted in natural order, there must be (d, f) and (d', f') at v such that $d > d'$ and $f \leq f'$ or $d \leq d'$ and $f > f'$. Then $(d, f) <_m (d', f')$ or $(d, f) >_m (d', f')$, a contradiction to Pareto optimal. The latter half can be seen as follows: Suppose there are (d, f) and (d', f') at v such that $d = d'$ or $f = f'$, which is a contradiction to Pareto optimal. ◀

▶ **Theorem 16.** *Algorithm 4 solves SSSP-AF in* $O(min\{t, cn\}m + cn^2)$ *time.*

Proof. Correctness is similar to that of Theorem 8. We measure the complexity by the number of accesses to major data structures. If a one-dimensional bucket system is used for Q, the total time for scanning the array for delete-min is $O(cn)$. For edge inspection at line 7, we observe pair (v, f) at line 5 brings Pareto optimal $(dist[v, f], f)$ at v. The size of the Pareto optimal solution at each v is bounded by $\min\{t, cn\}$ from the previous lemma. Thus the number of edge inspections for decrease-key and insert at line 7 is bounded by $\min\{t, cn\}m_i$ for vertex v_i. Summation over i can give us the time for decrease-key and insert being $O(\min\{t, cn\}m)$. The initialization for array $dist$, array $flow$ and Boolean arrays for membership of $S[f^*]$ and Q used at lines 10 and 11 takes $O(cn^2 + tn)$. Thus the total time is given by $O(\min\{t, cn\}m + cn + (cn + t)n) = O(\min\{t, cn\}m + (cn + t)n) = O(\min\{t, cn\}m + cn^2)$. ◀

▶ **Corollary 17.** *The SSSP-AF problem with edge costs bounded by* c *can be solved in* $O(cmn)$ *time. If the cost is a unit, it can be solved in* $O(mn)$ *time.*

Note. We could use the cascading bucket system to improve delete-min operations to $O(tn \log(c/t))$, but cannot improve the complexity of $O(cn^2)$ for the initialization of $flow$. It is open whether we can improve this time for initialization. In a way we improved the complexity of the first term at the higher cost of the second term, resulting in a better overall complexity for $c = o(n)$.

6 Single source bottleneck paths for all costs problem (SSBP-AC)

For every given cost d, we work out maximum flow $flow[v, d]$ that can be pushed from s to v for all v and d. Although we can design an algorithm for this problem by swapping the roles of *distance* and $flow$ in Algorithm 4, Algorithm 4 already solves this problem in array $flow$. Following the monotone property of $flow$ similar to Lemma 9, the solution can be obtained by

---

$L[v] = \phi; flow[v, 0] = 0$ for all v
for each v do
  for $d = 1$ to cn do if $flow[v, d] > 0$ then
    if $flow[v, d] \neq flow[v, d-1]$ then $L[v] = L[v]||(d, flow[v, d])$

---

7 Concluding remarks

Let us analyze our complexity as compared with the best known result in [15], which is $O(m^2 + mn \log(c/m))$. If $t = m$, we can say the degree of variety of the graph is high, that is, every edge has a distinct capacity. We take this case of $t = m$ following [15]. Let us define the density of the graph by $e = m/(n(n-1))$ [5], which is approximated by $e = m/n^2$. The complexity in [15] becomes $O(m^2) = O((en^2)^2)$ as the second term of the complexity becomes minor. Our complexity becomes $O(cen^3)$. Thus our complexity is better when $c \leq en$. We can say our algorithm performs well for denser graphs, i.e., with e close to 1.

When $c = O(n)$ or costs are real numbers for a dense graph, i.e., $m = O(n^2)$, the complexity is standing at $O(n^4)$ with only n as a complexity parameter. It is open whether sub-quartic is possible. There are some possibilities to extend our idea to improve time complexities for the all pairs shortest paths for all flows (APSP-AF) problem.

Acknowledgments. The author is thankful to the referees, whose careful reading and constructive comments greatly improved the quality of the paper. He also acknowledges he was greatly inspired by Tong-Wook Shinn on the subject of the research.

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.
2 N. Alon, Z. Galil, and O. Margalit. *On the exponent of the all pairs shortest path problems*. JCSS 54, 255-262, 1997.
3 Amit Chakrabarti, Chandra Chekuri, Anupam Gupta, and Amit Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica 47(1): 53-78*, 2007.
4 B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming 73, 129-174*, 1996.
5 Thomas F. Coleman and Jorge J. More. Estimation of sparse jacobian matrices and graph coloring problems. *SIAM Journal on Numerical Analysis 20 (1): 187-209*, 1983.
6 E. V. Denardo and B. L. Fox. Shortest-route methods: I. reaching, pruning, and buckets. *Operations Research 27, 161-186*, 1979.
7 R. B. Dial. Algorithm 360: Shortest path forest with topological ordering. *CACM 12, 632-633*, 1969.

8 E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math. 1, 269-271*, 1959.

9 Y. N. Dinits, N. Garg Y, and N. Goemans. On the single-source unsplittable flow problem. *Combinatorica, Springer, 19(1), 17-41*, 1999.

10 Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '09), pp. 384–391*, 2009.

11 Matthias Ehrgott. *Multicriteria Optimization.* Springer-Verlag, 2005.

12 M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Jour. ACM 34, 596-615*, 1987.

13 Harold N. Gabow and Robert E. Tarjan. Algorithms for two bottleneck optimization problems. *Journal of Algorithms 9 (3): 411–417*, 1988.

14 Tong-Wook Shinn and Tadao Takaoka. Combining all pairs shortest paths and all pairs bottleneck paths problems. *LATIN 2014: 226-237*, 2014.

15 Tong-Wook Shinn and Tadao Takaoka. Combining the shortest paths and the bottleneck paths problems. *ACSC 2014: 13-18*, 2014.

16 Tong-Wook Shinn and Tadao Takaoka. Some extensions of the bottleneck paths problem. *WALCOM 2014: 176-187*, 2014.

17 Tadao Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica 20(3): 309-318*, 1998.

18 Tadao Takaoka. Sharing information for the all pairs shortest path problem. *Theor. Comput. Sci. 520: 43-50*, 2014.

19 M. Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. *STOC03, 149-158*, 2003.

20 U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Jour. ACM, 49, 3, 289–317*, 2002.

Robust Routing in Urban Public Transportation: Evaluating Strategies that Learn From the Past*

Kateřina Böhmová[1], Matúš Mihalák[2], Peggy Neubert[3], Tobias Pröger[1], and Peter Widmayer[1]

1 **Department of Computer Science, ETH Zürich**
Universitätstrasse 6, 8092 Zürich, Switzerland
{katerina.boehmova,tobias.proeger,widmayer}@inf.ethz.ch

2 **Department of Knowledge Engineering, Maastricht University**
Postbus 616, 6200MD Maastricht, The Netherlands
matus.mihalak@maastrichtuniversity.nl

3 **Verkehrsbetriebe Zürich**
Luggwegstrasse 65, 8048 Zürich, Switzerland
Peggy.Neubert@vbz.ch

Abstract

Given an urban public transportation network and historic delay information, we consider the problem of computing reliable journeys. We propose new algorithms based on our recently presented solution concept (Böhmová et al., ATMOS 2013), and perform an experimental evaluation using real-world delay data from Zürich, Switzerland. We compare these methods to natural approaches as well as to our recently proposed method which can also be used to measure typicality of past observations. Moreover, we demonstrate how this measure relates to the predictive quality of the individual methods. In particular, if the past observations are typical, then the learning-based methods are able to produce solutions that perform well on typical days, even in the presence of large delays.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, F.2.2 Nonnumerical Algorithms and Problems, I.2.6 Learning

Keywords and phrases public transportation, route planning, robustness, optimization, experiments

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.68

1 Introduction

Motivation. When using public transportation to travel from a stop s to a stop t, we may want to arrive at t no later than at time t_A. Determining the right moment to leave s is nontrivial: We want to reach t at time t_A at the latest, but we don't want to leave s much too early. In an ideal situation, every bus and every tram is on time, and it is sufficient to compute a journey that is planned to leave s as late as possible but still reaches t at the latest at t_A. However, in reality, traffic can be congested and we should expect delays. Thus, we are looking for a *robust* journey from s to t that arrives before time t_A, but still leaves s

* This work has been partially supported by the Swiss National Science Foundation (SNF) under the grant number 200021 138117/1, and by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities and Sustainability), under grant agreement no. 288094 (project eCOMPASS). Kateřina Böhmová is a recipient of a Google Europe Fellowship in Optimization Algorithms, and this research is supported in part by this Google Fellowship.

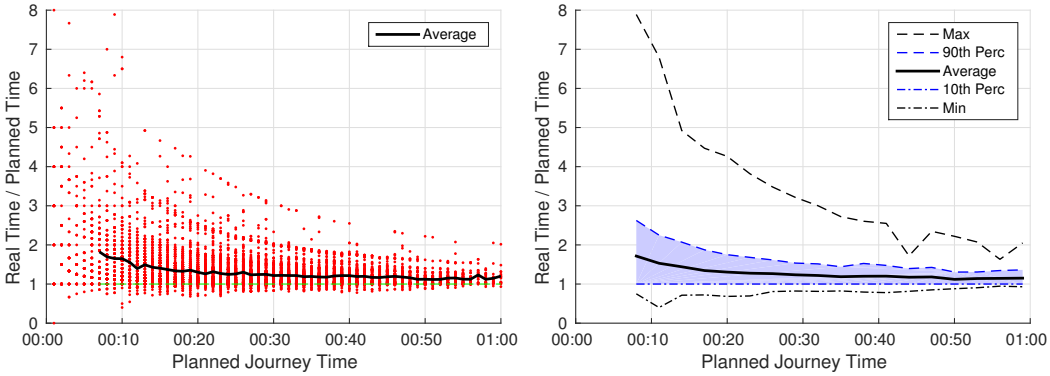


Figure 1 Distribution of the error coefficients grouped by planned travel time (a), and the average, minimum, maximum, 10th and 90th percentiles of the distribution of the error coefficients in each time slot (b). Times were measured in minutes. For the sake of clarity, (a) does not show journeys with an error coefficient greater than 8, because there are only few ($< 0.1\%$). For the same reason journeys with more than one hour planned travel time are not shown in the figures.

at a "reasonable" time. In real applications one may have additional preferences, such as low travel costs, which we don't consider for the sake of simplicity.

Firmani et al. [9] observed in an experimental study on the transportation network of Rome that the timetable information and the real movement of the vehicles (based on GPS data) are only mildly correlated. They conclude that an "important issue to investigate is how to compute robust routes" that are "less vulnerable to unexpected events". Our goal is provide methods for finding such robust routes. Since we only have historic delay data from the public transportation network of Zürich, as a preliminary step we investigated whether our network exhibits a behaviour similar to the Rome network. To make our results comparable to the results of Firmani et al., the methodology and notation of our preliminary study are similar to the methodology and notation in their original article [9].

We selected 10,000 departure and target stops s, t uniformly at random, set the latest allowed arrival time t_A to 8:30, and computed the st-journeys j that are optimal according to the planned timetable. For each journey j, we measured the planned travel times $t_p(j)$ as well as the actual travel times $t_a(j)$ (on 23 May 2013), and computed the error coefficient $t_a(j)/t_p(j)$. Figure 1(a) shows the distribution of the error coefficients grouped by the planned travel times $t_p(j)$. High error coefficients occur easily if the planned travel time is small and the vehicle of the planned journey leaves s a bit too early so that one has to wait for the next vehicle (which may, depending on the line, take up to half an hour in Zürich).

As in [9], we grouped the journeys into 3-minute time slots such that the k-th slot contains all journeys j with $t_p(j) \in (3(k-1), 3k]$. Figure 1(b) shows the average, minimum, maximum as well as the 10th and the 90th percentile of the distribution of the error coefficients of the journeys in each time slot. Since short journeys sometimes have high error coefficients, for simplicity Figure 1(b) does not incorporate the first two slots. The average error coefficient of the journeys in the remaining slots lies between 1.12 and 1.71 which means that in average a journey may take up to 71% longer than planned. Also, observe that the 90th percentile of the error coefficients of the journeys with 15 minutes travel time is roughly 2. Thus, 10% of the 15-minute journeys take in reality at least twice as long as planned. In overall, we observed that the behaviour in Zürich is comparable to the one in Rome.

One way out might be to integrate real-time information into the computation of routes. However, we believe that this is not enough, especially if the journey is planned some time

in advance. For example, a trip to the airport is usually planned a few hours earlier, and the right departure time needs to be computed before the start of the journey. Moreover in reality it often happens that delays occur suddenly and cannot be foreseen in advance, especially not at the time when the journey is planned. For example, consider an st-journey that consists of two lines l_1 and l_2, and imagine that the transfer time between the lines is 2 minutes. Even if l_1 leaves s on time, every upcoming delay of more than 1 minute (which might always occur) leads to a late arrival at t.

Our Contribution. In [3], we introduced a novel approach for finding robust journeys that uses recorded observations from the past as input—we look for journeys that performed well in the given past observations. Since this approach requires journeys to be comparable in different past days, classical solutions concepts, such as a path in the time-expanded or the time-dependent graph, are not suitable.

In the present paper, we first shortly describe our solution concept and the above-mentioned approach for finding robust routes. Since this approach was originally restricted to learn from historic data of only two different days, we show how it can be generalized to consider historic data from multiple days. We also describe how a stochastic method by Lim et al. [13] for private transportation can be adapted to compute robust journeys in public transportation. After that, we perform an extensive experimental study to evaluate these methods and to investigate different aspects related to robust routing.

Related Work. Many approaches to find a fastest journey in a given public transportation network were considered in the literature, see, e.g., a recent survey by Bast et al. [1]. One approach to account for delays is using stochastic methods—the delays are typically modeled as random variables on the edges of the network [4, 10, 15], or on each vehicle [6, 7]. For a given fixed timetable, Disser et al. [8] extended Dijkstra's algorithm for computing pareto-optimal multi-criteria journeys. Müller-Hannemann and Schnee [14] used a *dependency graph* to predict secondary delays caused by some current primary delays and gave a routing strategy with respect to these delays. Bast et al. [2] studied the robustness of transfer patterns in the presence of delays. They argue that even when delays occur, a reasonably good path is still included in the pattern. Dibbelt et al. [7] modeled the delays using stochasticity and computed a *decision graph* with all the possibly relevant nodes and vehicles instead of a single path. Goerigk et al. [12] assumed that a set of delay *scenarios* is provided, and showed how to compute a journey that arrives on time in every scenario (strict robustness) or a journey with fewest number of unreliable transfers having an almost optimal travel time (light robustness). Goerigk et al. [11] considered journeys, within the setting of delay scenarios, that can be updated if delays occur (recoverable robustness).

2 Model

Network Design. Let $\mathcal{S}$ be a set of stops. A *line* is an ordered sequence $\langle v_1, \dots, v_k \rangle$ of stops from $\mathcal{S}$, where v_i is visited directly before v_{i+1}. We explicitly distinguish two lines with the same stops but opposite directions. Given a departure stop $s \in \mathcal{S}$ and a target stop $t \in \mathcal{S}$, a sequence of lines $\langle l_1, \dots, l_{\beta+1} \rangle$ with $l_i \neq l_{i+1}$ is called an *st-route* if there exist $\beta + 2$ stops $v_0 := s, v_1, \dots, v_\beta, v_{\beta+1} := t$ where both v_{i-1} and v_i are stops on the line l_i, and the line l_i visits v_{i-1} (not necessarily directly) before v_i. We say that a *transfer* between the lines l_i and l_{i+1} occurs at v_i. Notice that there might be more than one possible transfer

between two lines. For $s, t \in \mathcal{S}$ and an integer $\beta \in \mathbb{N}_0$, let $\mathcal{R}_{st}^{\beta}$ denote the set of all st-routes with at most β transfers.

A *journey* consists of a departure time t_D, a route $\langle l_1, \ldots, l_{\alpha+1} \rangle \in \mathcal{R}_{st}^{\alpha}$ with $\alpha \leq \beta$, and a sequence of transfer stops $\langle v_1, \ldots, v_\alpha \rangle$. Its intuitive interpretation is to leave the stop s at time t_D, take the first arriving (trip of) line l_1, and for every $i \in \{1, \ldots, \alpha\}$, leave l_i at stop v_i and immediately take the next arriving trip of line l_{i+1}.

Trips and Timetables. While the only information associated with a line itself are its consecutive stops, it usually is operated multiple times per day. Each of these concrete realizations is called a *trip*. A *timetable* stores for every stop $v \in \mathcal{S}$ the arrival and departure times of every trip over a day. We have

1. a *planned* timetable T_{plan} which we assume to be periodic, i.e., every line realized by some trip τ will be realized by a later trip τ' again (not necessarily on the same day).
2. a set $\mathcal{T}$ of *recorded* timetables T_i that describe how various lines were operated during a given time period (e.g., on a concrete day). These recorded timetables are concrete executions of the planned timetable.

In the following, *timetable* refers both to the planned as well as to a recorded timetable. We assume that timetables respect the FIFO property, i.e. two buses or trams *of the same line* do not overtake each other.

Goal. Let $s, t \in \mathcal{S}$ be the departure and the target stop, and let t_A be the latest allowed arrival time. Our goal is to use the planned timetable and the recorded timetables in $\mathcal{T}$ to compute a recommendation in form of one or more (robust) journeys from s to t that will likely arrive on time (i.e., at time t_A or earlier) on a day for which the concrete travel times are not known yet.

We assume that users select one of the recommended journeys, and then travel according to it. One may argue that this assumption is rather strict, because when delays come up, users sometimes spontaneously decide to use a different journey instead. This, however, is a different situation that we do not consider in this paper for two reasons. First, one needs to know the network and the possible backup options well, which might not be the case when one is travelling in a foreign city. Second, as mentioned earlier, delays may occur suddenly, and it might be too late to choose a different journey. Consider, for example, the situation when the alternative journeys don't have any stop in common except for the departure and the target stop. In such a case one has to fix the journey already in advance.

3 Robustness

Overview. In this section we present some approaches for computing robust journeys. For this we assume that the departure stop s, the target stop t and the latest allowed arrival time t_A were specified by the user and that we already computed a reasonable upper bound β on the maximum number of transfers. Hence, s, t, t_A and β are fixed when the journey(s) are computed. We note that, given a route $r \in \mathcal{R}_{st}^{\beta}$ and a parameter $\gamma \in \mathbb{N}$, we can use the planned timetable T_{plan} to find a journey j along r that leaves s as late as possible, but not later than time $t_A - \gamma$. Thus, as soon as an algorithm identifies both a route r and a parameter γ, it can also reconstruct the corresponding journey in the planned timetable. These planned journeys will then be recommended to the user.

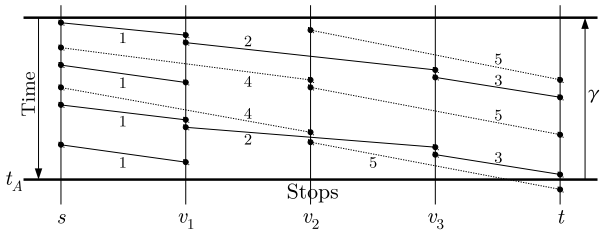


Figure 2 A timetable with five lines $\{1, \ldots, 5\}$ and two routes $r_1 = \langle 1, 2, 3 \rangle$ (solid) and $r_2 = \langle 4, 5 \rangle$ (dotted). The x-axis denotes the stops $\{s, v_1, v_2, v_3, t\}$, the y-axis the time. If a trip leaves a stop v_d at time t_d and arrives at a stop v_a at time t_a, it is indicated by a line segment from (v_d, t_d) to (v_a, t_a). $A_\gamma(T)$ contains r_1 three times and r_2 once.

Transfer Buffers. An naïve strategy to increase the reliability of a journey is to enforce an additional buffer time at each transfer or at the end of the trip. The Buffer-ξ approach uses T_{plan} to compute a journey that is planned to leave s as late as possible, arrives at t not later than at time t_A, and that has an additional time of at least ξ at each transfer of the journey. This especially implies that if a line l_i is planned to arrive at a transfer stop v_i at time t_i, then the next line l_{i+1} of the journey can only be taken at time $t_i + \xi$ or later. Buffer-0 corresponds to an optimal journey in the planned timetable, so we refer to it as Opt-TT.

A Similarity-Based Approach. In [3], we described how a general approach to robust optimization designed by Buhmann et al. [5] can be used to compute robust journeys. We briefly recall our ideas. Let $T \in \mathcal{T}$ be a timetable and $\gamma \in \mathbb{N}_0$. An *approximation set* $A_\gamma(T)$ contains all routes $r \in \mathcal{R}^\beta_{st}$ for which T contains a journey along r that leaves s at time $t_A - \gamma$ or later, and that arrives at t at time t_A or earlier. We assume that $A_\gamma(T)$ is a *multiset*: a route r is contained as often as it is realized by a journey starting at time $t_A - \gamma$ or later, and arriving at time t_A or earlier (see Figure 2 for an example). The parameter γ can be interpreted as the maximal time that we depart before t_A. In general we have $A_0(T) = \emptyset$, and the size of $A_\gamma(T)$ grows with increasing γ. If we consider the approximation sets $A_\gamma(T_1), \ldots, A_\gamma(T_k)$ for the timetables $T_1, \ldots, T_k \in \mathcal{T}$, every approximation set contains only routes that are realized (by a journey) in the same time period $[t_A - \gamma, t_A]$, and that are therefore comparable among different approximation sets.

The approach in [3, 5] expects that exactly two timetables $T_1, T_2 \in \mathcal{T}$ are given. To compute a *robust* route when only two timetables are available, we consider $A_\gamma(T_1) \cap A_\gamma(T_2)$: the only chance to find a route that is likely to be good in the future is a route that performed well in *both* recorded timetables. The parameter γ determines the size of the intersection: if γ is too small, the intersection will be empty. If γ is too large, the intersection contains many (and maybe all) st-routes, and not all of them will be a good choice. Assuming that we knew the "optimal" parameter γ_{OPT}, we could pick a route from $A_{\gamma_{OPT}}(T_1) \cap A_{\gamma_{OPT}}(T_2)$. Buhmann et al. [5] suggest to set γ_{OPT} to the value γ that maximizes

$$S_\gamma = \frac{|\mathcal{R}^\beta_{st}||A_\gamma(T_1) \cap A_\gamma(T_2)|}{|A_\gamma(T_1)||A_\gamma(T_2)|}. \tag{1}$$

The value $S_{\gamma_{OPT}}$ measures how similar the timetables T_1 and T_2 are, so Buhmann et al. refer to this ratio as the *similarity* of T_1 and T_2. They showed that it is always at least 1, and the larger it gets, the more similar T_1 and T_2 are. Of course, if one is only interested in computing γ_{OPT} (and not measuring the similarity itself), one can simply omit the term $|\mathcal{R}^\beta_{st}|$ in equation (1) as we did in our original work [5].

After γ_{OPT} has been computed, there are two possible approaches to pick a route from $A_{\gamma_{OPT}}(T_1) \cap A_{\gamma_{OPT}}(T_2)$. The Similarity-Rand approach selects a route r from the intersection uniformly at random, while Similarity-MRR selects the most frequent route r from the intersection. For both approaches we recommend to depart at least γ_{OPT} units of time in advance. More details can be found in [3, 5].

Function-Based Approaches. Let $T_i \in \mathcal{T}$ be a recorded timetable, $r = \langle l_1, \ldots, l_{\alpha+1} \rangle \in \mathcal{R}^{\alpha}_{st}$ be a route, $\tau_1, \ldots, \tau_k$ be the trips of line l_1 in T_i and $D(\tau_j, s)$ be the departure time of the trip τ_j at s. We define δ^r_i as

$$\min_{j \in [1,k]} \left\{ t_A - D(\tau_j, s) \;\middle|\; \begin{array}{l} \tau_j \text{ can be extended to a journey along } r \text{ that} \\ \text{arrives in } T_i \text{ at stop } t \text{ at time } t_A \text{ or earlier} \end{array} \right\}, \tag{2}$$

which intuitively can be interpreted as follows: to arrive on time using route r on the day at which T_i is realized, one has to leave s at least δ^r_i units of time before the latest allowed arrival time t_A. For a given function $f : (\mathbb{R}^+)^{|\mathcal{T}|} \to \mathbb{R}$, we search for a route $r \in \mathcal{R}^{\alpha}_{st}$ that minimizes $f(\delta^r_1, \ldots, \delta^r_{|\mathcal{T}|})$. In the following, we describe some possible choices for f, and we abbreviate $f(\delta^r_1, \ldots, \delta^r_{|\mathcal{T}|})$ by $f(r)$.

For a number $p \in [1, \infty]$, the Norm-p estimator has the objective function

$$f^p_{\|\cdot\|}(r) = \left\| \left(\delta^r_1, \ldots, \delta^r_{|\mathcal{T}|}\right) \right\|_p. \tag{3}$$

It is easy to see that $f^1_{\|\cdot\|}$ selects all routes which in average (w.r.t. the recorded timetables in $\mathcal{T}$) depart as late as possible. Moreover, $f^{\infty}_{\|\cdot\|}$ selects all routes minimizing the maximum time between the departure and the latest allowed arrival time t_A. Such routes can alternatively be seen as routes maximizing the earliest departure time necessary to arrive on time in *all* timetables in $\mathcal{T}$. Thus, the Norm-∞ estimator is related to the similarity-based approach from the previous paragraph in the following way. Let $\gamma_{FI} = \min\left\{\gamma > 0 \mid \bigcap_{i=1}^{|\mathcal{T}|} A_\gamma(T_i) \neq \emptyset\right\}$ be the smallest value for γ such that the intersection of all γ-approximation sets is non-empty. One can observe that every route r contained in $\bigcap_{i=1}^{|\mathcal{T}|} A_{\gamma_{FI}}(T_i)$ minimizes $f^{\infty}_{\|\cdot\|}$ and vice versa. We note that these methods relate to strict robustness [12], but are based on a different solution concept, and learn from past observations given as daily recorded timetables (instead of specifying a set of possible delays).

Now, let $p \in [1, \infty]$ be arbitrary and let r^p_j be a route minimizing $f^p_{\|\cdot\|}$. To determine how much in advance one has to depart when using r^p_j, we use our previous observations. For $p = 1$, it is reasonable to set $\gamma^p_j = f^1(r^p_j)/|\mathcal{T}|$ since $f^1_{\|\cdot\|}$ corresponds to averaging the departure times. For $p = \infty$, it is reasonable to set $\gamma^p_j = f^{\infty}(r^p_j)$. For every other $p \in (1, \infty)$, we simply scale the time linearly with respect to $p = 1$ and $p = \infty$. More concretely, we set

$$\gamma^p_j = f^{\infty}(r^p_j) - \left(\frac{f^p(r^p_j) - f^{\infty}(r^p_j)}{f^1(r^p_j) - f^{\infty}(r^p_j)} \right) \cdot \left(f^{\infty}(r^p_j) - f^1(r^p_j)/|\mathcal{T}| \right). \tag{4}$$

A different function-based estimator comes from the *mean-risk model* which was just recently used for finding robust routes in private transportation [13]. Let $c \in \mathbb{R}^+_0$ be the *risk-aversion coefficient*, where $c = 0$ corresponds to the situation where the risk is being completely ignored. The objective function associated with the Mean-Risk-c estimator is

$$f^c_{MR}(r) = \text{Mean}\left(\delta^r_1, \ldots, \delta^r_{|\mathcal{T}|}\right) + c \cdot \sqrt{\text{Variance}\left(\delta^r_1, \ldots, \delta^r_{|\mathcal{T}|}\right)}. \tag{5}$$

For a route r_j minimizing f^c_{MR}, we simply set $\gamma_j = f^c_{MR}(r_j)$ as the time one has to depart in advance. Notice that Mean-Risk-0 is equivalent to Norm-1.

4 Experimental Results

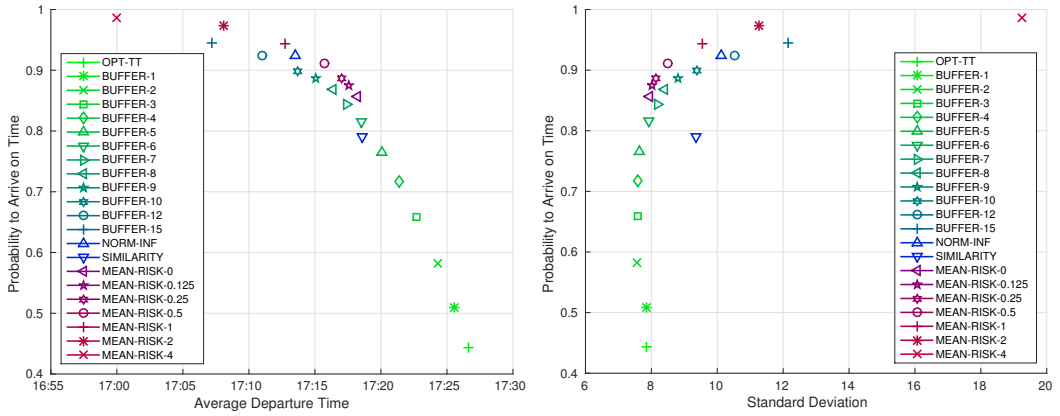
Experimental Setup. For an experimental evaluation of the methods proposed in Section 3 we used the tram and bus network of the city of Zürich, Switzerland, which has 401 stops and 292 lines. The recorded timetables $\mathcal{T} = \{T_1, \ldots, T_7\}$ were realized on seven consecutive Thursdays in the period from 4 April to 23 May 2013, ignoring 9 May (which was a public holiday and therefore had different traffic and a different planned timetable).

We observed that in reality many of the 292 lines have the same ID (such as, e.g., tram 6, bus 31, etc.). This is consequence of our modeling: not only do we distinguish lines travelling in opposite directions, but there are also special lines coming from or going to the depot, lines whose corresponding vehicle turns around in advance, and lines that do not visit certain stops in the evening. Since these special lines operate only on a low frequency and mostly only early in the morning or late in the evening, we ignored them and focused on the "standard" realizations. Hence we effectively used only 118 of the 292 lines. Although the network is rather small in comparison to the networks of other cities, it is well-suited for an experimental study on robustness for two reasons. First, the network is dense enough to provide many different routes between any two stops s and t. Second, our study in Section 1 showed that the network is affected by a considerable amount of delays, especially during the rush hours.

For each of the following experiments, we generated 10,000 (30,000 for the experiments on the number of transfers) departure/target pairs $(s, t) \in \mathcal{S}^2$ with $s \neq t$ uniformly at random. For each such pair (s, t), we computed the smallest $\beta \in \mathbb{N}_0$ such that $\mathcal{R}^{\beta}_{st} \neq \emptyset$ and used this value for the maximum allowed number of transfers. We explicitly set $\beta = 1$ if there exists a direct st-route with no transfers at all. In such a case, one might prefer to take an alternative route with only one transfer, probably leading to a shorter travel time. After computing β and $\mathcal{R}^{\beta}_{st}$, we performed the corresponding experiment. We set the target arrival time t_A to 18:00 except for the experiments that study how the behavior of the methods changes during the day. Unless otherwise stated, the buffer methods used the planned timetable T_{plan} as input, the similarity-based methods used T_5 and T_6 (recorded on 2 May and 16 May), and the function-based methods used $T_1, \ldots, T_6$ (recorded between 4 April and 16 May). Timetable T_7 (recorded on 23 May) was used to assess the quality of the proposed journeys.

In our experiments we observed that the performance of Similarity-Rand and Similarity-MRR is nearly identical, so our figures show only the behavior of the latter variant, and for simplicity we refer to both variants as Similarity. Also, Norm-2 performs similarly to Norm-Inf, so our figures mostly omit Norm-2. Furthermore we observed that it rarely happened that a journey proposed by Buffer-ξ, Similarity, Norm-Inf or Mean-Risk-1 arrived much too early or much too late in the test instance. In all of these cases this was caused either because of a highly non-typical situation in the input or the test instance (e.g., an accident), or because a line was chosen that was not realized regularly (e.g., less than once per hour). Hence we ignored all pairs (s, t) for which at least one of the methods above computed a journey arriving more than one hour too early or too late.

Our algorithms were implemented in Java 7, and the experiments were performed on one core of an Intel Core i5-3470 CPU clocked at 3.2 GHz with 4 GB of RAM running Debian Linux 7.8. For enumerating all st-routes in $\mathcal{R}^{\beta}_{st}$, we used the algorithm proposed in [3] which runs on average 35ms. After computing $\mathcal{R}^{\beta}_{st}$, the buffer strategies have an average running time 1ms or less, the similarity-based methods 8ms, and the function-based approaches 24ms. Notice that these running times are faster than the ones described in [3], because we used a smaller network (without the agglomeration).



(a) (b)

Figure 3 Comparison of various methods: arrival rate vs. average departure time (a), and arrival rate vs. standard deviation on the arrival time (b).

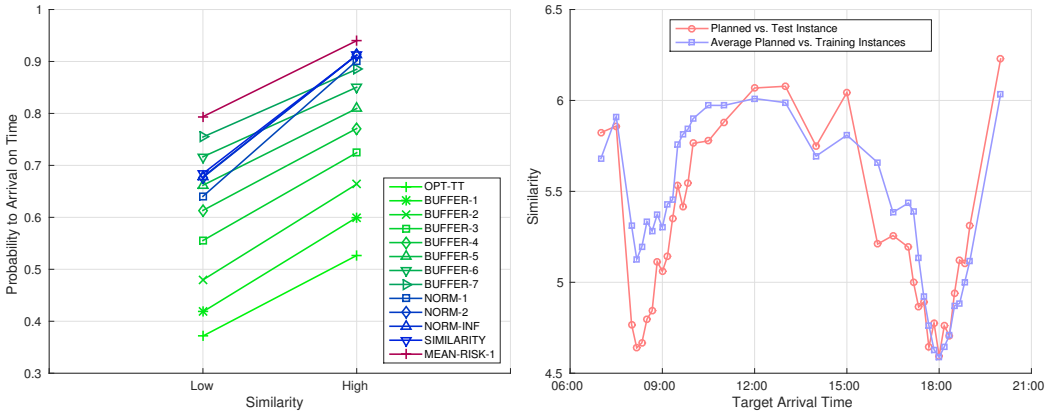
Table 1 Overview of how often the route suggestions of two methods differ.

	Opt-TT	Buffer-3	Buffer-6	Buffer-9	Buffer-12	Norm-1	Norm-Inf	Similarity	Mean-Risk-1
Opt-TT		30, 82%	24, 14%	25, 79%	29, 56%	27, 32%	40, 45%	40, 21%	32, 91%
Buffer-3	30, 82%		31, 60%	25, 97%	24, 89%	30, 31%	40, 05%	40, 72%	32, 54%
Buffer-6	24, 14%	31, 60%		28, 77%	25, 83%	30, 86%	41, 03%	42, 17%	34, 16%
Buffer-9	25, 79%	25, 97%	28, 77%		30, 37%	29, 99%	39, 59%	40, 68%	32, 03%
Buffer-12	29, 56%	24, 89%	25, 83%	30, 37%		31, 77%	40, 83%	42, 48%	33, 99%
Norm-1	27, 32%	30, 31%	30, 86%	29, 99%	31, 77%		27, 48%	31, 30%	14, 33%
Norm-Inf	40, 45%	40, 05%	41, 03%	39, 59%	40, 83%	27, 48%		32, 43%	19, 54%
Similarity	40, 21%	40, 72%	42, 17%	40, 68%	42, 48%	31, 30%	32, 43%		32, 50%
Mean-Risk-1	32, 91%	32, 54%	34, 16%	32, 03%	33, 99%	14, 33%	19, 54%	32, 50%	

Arrival Rate, Departure Time and Standard Deviation on the Arrival Time. Intuitively, an earlier departure time leads to a higher probability to arrive on time (i.e., a higher arrival rate), and achieving a higher arrival rate in a network with delays entails a higher standard deviation on the arrival time. Figure 3 compares the proposed methods with respect to these aspects. It shows that, independently of the considered method, there is a clear trade-off between the departure time and the arrival rate (a) as well as between the standard deviation of the arrival time and the arrival rate (b).

Both parameter-based methods Buffer-ξ and Mean-Risk-c, form Pareto optimal fronts in both (a) and (b). Clearly, Mean-Risk-c benefits from the additional information from the input instances $T_1, \ldots, T_6$ and it dominates Buffer-ξ in both (a) and (b). The Similarity method needs no parameter adjustment, it is based only on two past timetables, and still proposes solutions with a reasonable arrival rate that do not depart too early. Notice that Norm-Inf (the generalization of Similarity) also benefits from the knowledge of the six past timetables, and without parameter adjustment it produces a solution which gives a very reasonable trade-off between departure time, arrival rate and the standard deviation on the arrival time. Moreover, the solutions proposed by Norm-Inf performed rather well compared to all the competitors (which do require parameter adjustment).

We also investigated whether the arrival rates of different methods differ due to different departure times only, or whether the suggested route(s) also differ. In particular, for any two methods M_1 and M_2, we studied how often the suggested route(s) of M_1 and M_2 differ. For



(a) (b)

Figure 4 Influence of low/high similarity on the arrival rate: comparing various methods (a). Influence of the target arrival time on the similarity of the planned timetable and the test instance T_7, and on the average similarity between the planned timetable and each of the input instances $T_1, \ldots, T_6$ (b).

some exemplary methods, Table 1 shows that this happens in 14 to 42% of the cases. Notice that in roughly one third of the cases, the routes proposed by Similarity differ from the ones proposed by Norm-Inf (which can be seen as a generalization of Similarity). Also, there is a notable difference between the route suggestions of the different Buffer methods. Thus, for enforcing robustness there are better strategies than merely decreasing the departure time.

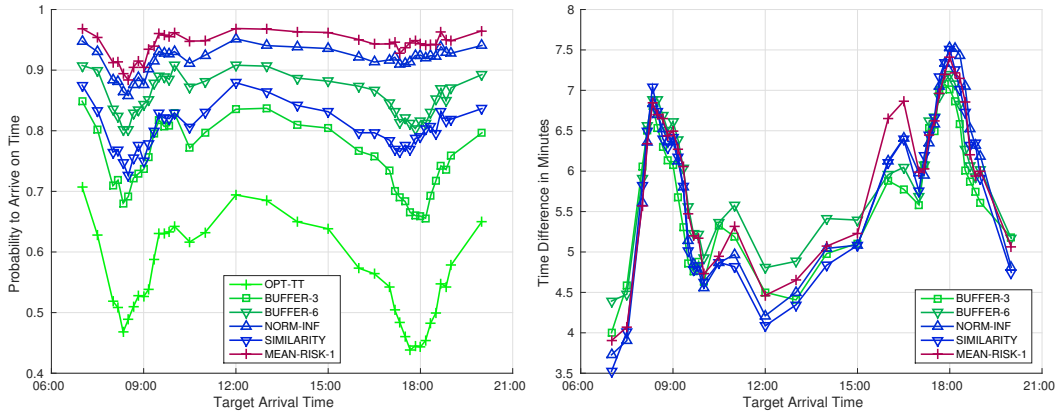
Influence of the Similarity between Input and Test Instances. We just saw that journeys proposed by the similarity-based approaches performed rather poorly, with respect to both arrival rate as well as standard deviation on the arrival time. However, we have to take into account that these methods use only two recorded timetables as input: if both differ substantially from the test instance, then in general there is very little one can do. The generic approach by Buhmann et al. [5] works well if both the input and the test instances are typical, i.e., if their mutual similarity is high. Thus we investigate the impact of high and low mutual similarities on the quality of the predictions.

First we note that the similarity $S_{\gamma_{OPT}}$ does not only depend on the two input instances but also on the origin s and the destination t, and on the target arrival time t_A. Thus, in the following experiments, we do not always use the same timetables T_5, T_6 as input and T_7 for testing, but select for every (s,t) the timetables whose mutual similarities are as high or as low as possible. Let Υ be the set of all triples of recorded timetables $(T_i, T_j, T_k) \in \mathcal{T}^3$ where i, j, k are mutually different. For a given pair (s,t) and two timetables $T_i, T_j \in \mathcal{T}$, let S^{st}_{ij} be the similarity of T_i and T_j with respect to s and t. We selected triples whose minimum (or maximum, respectively) pairwise similarity is as high or as low as possible,

$$(T_1^h, T_2^h, T_3^h) = \underset{(T_i,T_j,T_k)\in\Upsilon}{\arg\max} \min\left\{S^{st}_{ij}, S^{st}_{ik}, S^{st}_{jk}\right\} \tag{6}$$

$$(T_1^l, T_2^l, T_3^l) = \underset{(T_i,T_j,T_k)\in\Upsilon}{\arg\min} \max\left\{S^{st}_{ij}, S^{st}_{ik}, S^{st}_{jk}\right\} \tag{7}$$

and used T_1^h and T_2^h as input and T_3^h for testing, and for comparison, used T_1^l and T_2^l as input and T_3^l for testing. Even though Mean-Risk-c and Norm-p could handle more instances, they were given just the two mentioned instances.



(a) (b)

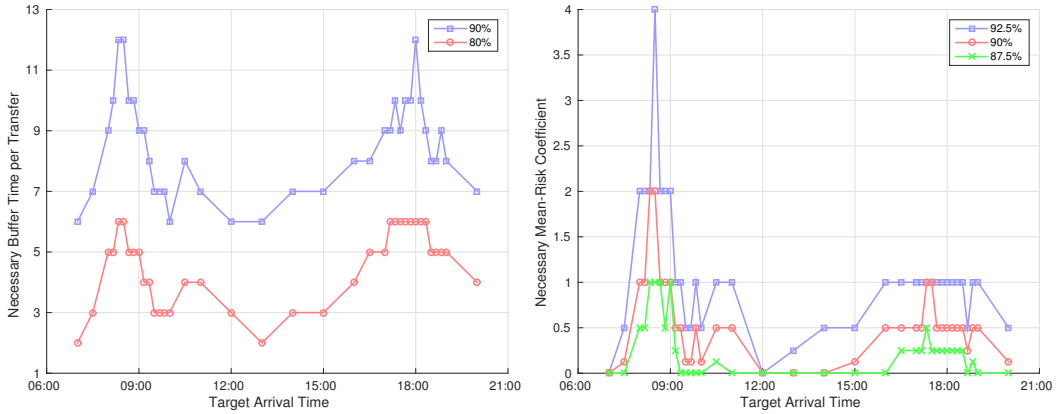
Figure 5 Comparison of various methods: Arrival rate vs. target arrival time (a), and travel time difference to the optimum travel time vs. target arrival time (b).

Figure 4(a) shows that all methods benefit when the similarity of the three instances is high. The arrival rates of both Norm-p and especially Similarity increase significantly. We observed that Similarity outperforms Norm-p when the similarity is low, which is reasonable: for a low similarity, the routes in the first intersection of the approximation sets as well as the route that maximizes the average departure time are too much influenced by the noise in the input instances. However, Similarity can still let the approximation sets grow beyond the first intersection so that more stable solutions are contained (which Norm-p cannot). On the other hand, if the similarity is high, then there is so little noise in the data that S_γ is maximized already at the first γ for which the intersection is non-empty, thus Similarity and Norm-p are nearly identical.

Of course these results cannot directly be used for designing an algorithm, since the test instance is unknown. Nevertheless we believe that the results are interesting because they demonstrate the power of the similarity-based approach.

Influence of the Target Arrival Time. Figure 5 shows how the behavior of the methods, in terms of the arrival rate (a) and travel time (b), changes over the day. In particular, we can observe a clear influence of the morning and evening rush hours. Interestingly, the two rush hours affect the arrival rates of different methods differently. Specifically, the timetable-based method Buffer-ξ is greatly affected by both rush hours while the learning strategies are less affected by the evening rush hour.

To understand this behavior, consider Figure 4(b). The red curve shows how the value of the similarity of T_{plan} and the test instance T_7 changes during the day. In particular, we see a significant drop of the similarity during rush hours. Notably, the two dips corresponding to morning and evening rush hour are of the same height. This suggests that on the day corresponding to T_7, during the morning rush hour, there was a similar amount of irregularities with respect to T_{plan}, as during the evening one. The blue curve in Figure 4(b) shows the changes during the day of the averaged value of similarity of T_{plan} and each of the training instances $T_1 - T_6$. Also there the similarity drops during rush hours, but we clearly see that the morning dip is significantly lower than the evening one. This suggests that in the recorded timetables $T_1 - T_6$ used for learning, the amount of irregularities (with respect to T_{plan}) was lower in the morning than in the evening. Thus, when comparing the two



(a) Buffer-ξ (b) Mean-Risk-c

Figure 6 Necessary parameter to achieve a specified arrival rate in T_7 depending on the target arrival time.

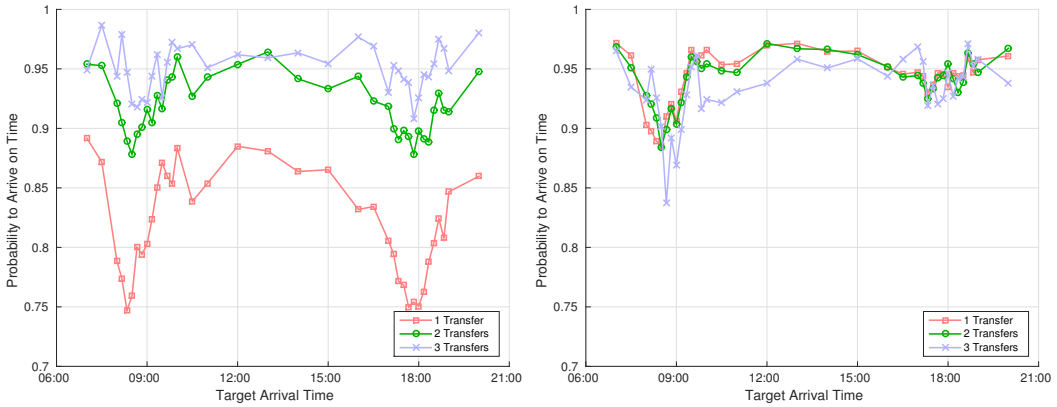
curves, we see a significant gap between them during the morning rush hour, but a relative match during the rest of the day. This suggests that the test instance T_7 contained during the day a similar amount of irregularities as it is expected on a typical day (represented by $T_1 - T_6$), with the only exception of the morning rush hour, where it was less regular.

Let us now relate what we observed in Figures 4(b) and 5(a). Since Buffer-ξ is based solely on T_{plan}, any irregularities with respect to T_{plan} occurring in T_7 (captured by the red curve in Figure 4(b)) affect its arrival rate. This explains why the arrival rate of Buffer-ξ drops both in the morning and evening rush hour and exhibits two dips of nearly the same height. On the other hand, the methods that use the information from the past observations (e.g., Mean-Risk-c) are trained to account for a certain amount of irregularities. Since the situation in T_7 in the evening is typical, the solutions proposed by these methods are prepared for it and their arrival rate is almost not affected by the evening rush hour. In contrast, morning rush hour causes their arrival rate to drop significantly and this maps to the discrepancy of the red and blue curve in Figure 4(b).

In Figure 5(b) we observe that during peak hours, the travel time increases. Interestingly, the required travel time does not depend on the method nor whether it is on time or not. Thus, to achieve higher probability to arrive on time, one has to depart earlier (as seen in Figure 3(a)), but does not need to increase the time spent traveling. We believe that this is the case because the network of Zürich is quite dense, hence there exist different alternative journeys with comparable travel times.

Choice of the Parameters for Buffer-ξ and Mean-Risk-c. Figure 6(a) displays the minimum value of the parameter ξ of Buffer-ξ that would be necessary to achieve arrival rates of 80%, and 90% of the cases in T_7, and how this value changes over the day. We see that, affected by the daily rush hours, this parameter varies significantly, suggesting that the Buffer-ξ strategy needs a non-trivial amount of parameter adjustment. We observe that the dips corresponding to morning and evening rush hours are of the same height. Again, we can directly relate this behavior with the similarity of T_{plan} and T_7 (captured by the red curve in Figure 4(b)).

Similarly, Figure 6(b) displays the value of the coefficient c of Mean-Risk-c that would be necessary to achieve arrival rates of 78.5%, 90%, and 92.5% in T_7, and its development



(a) Buffer-6 (b) Mean-Risk-1

Figure 7 Influence of the number of transfers on the arrival rate.

during the day. We observe that this value is greatly affected by the morning rush hour. On the other hand, the dip corresponding to the evening rush hour is visible, but not too significant. Again, we link this behavior of the value to the observed similarity of the training/test instances with the planned timetable—the two curves captured by Figure 4(b). Recall that in the morning rush hour there is a gap between the two curves in Figure 4(b) indicating that the situation in T_7 was not typical with respect to previous observations. As we see in Figure 6(b), the value of the coefficient c has be quite large to compensate for the unexpected irregularities. In contrast, in a situation that is typical (i.e., when the two curves in 4(b) approximately match), the Mean-Risk-c method performs well and fine-tuning of the parameters is not crucial. For instance, a coefficient c set to 1 leads to reasonably robust solutions.

Influence of the Number of Transfers. Figure 7(a) shows that the arrival rate of Buffer-ξ (for $\xi = 6$) is quite sensitive to the number of transfers. This suggests that the number of transfers is another aspect (of possibly many aspects) which has to be taken into account when searching for the best parameter for Buffer-ξ. In contrast, Figure 7(b) shows that the influence of the number of transfers on the arrival rate of Mean-Risk-c (for $c = 1$) is almost negligible. Thus, there is no need to fine-tune the coefficient c to compensate for this aspect. We remark that we generally observed that the arrival rate of the methods based on the past observations is not very sensitive to the number of transfers.

5 Conclusion

We observed a clear trade-off: to achieve a higher probability to arrive on time in a network with delays, one has to depart earlier and expect higher standard deviation on the arrival time. On the other hand, the average travel time itself does not change with robustness or the choice of a routing method.

Methods based solely on the planned timetable, where the robustness is achieved by adding buffer times, need a non-trivial parameter adjustment for which many aspects need to be considered (time of the day, number of transfers, etc.). The methods that learn from past benefit from the additional knowledge: If the test instance is typical with respect to the past observations, these strategies perform well, Mean-Risk-c does not need much fine-tuning, and Norm-Inf without parameter adjustment proposes a highly competitive solution with

reasonable trade-offs. We have seen that Similarity gives a good measure of the amount of irregularities in the network and can help to detect typical situations. Notably, it considers complex solutions (journeys), and thus it has a potential to capture behavior that cannot be observed only locally. We believe that this measure is worth further exploring, and by considering various aspects (e.g., how different approaches would benefit if Similarity was used to preselect typical instances for training) it can bring us even closer to the goal of robust routing.

The existence of equally good alternative journeys is one of the reasons why we believe that it was reasonable to choose the public transportation network of Zürich for our experiments, although the network is rather small in comparison to the public transportation networks of other cities. An interesting question is whether the algorithms are still sufficiently fast on larger networks. We believe that due to our solution concept (i.e., sequences of lines), the running time depends on the number of *lines* rather than the number of stops. In that respect, the network of Zürich is not exorbitantly small: For example, the public transportation network in Vienna has more than six times as many stops, but only less than two times as many lines. Hence, if the number of feasible *st*-routes (with a bounded number of transfers) is not too large, the algorithms should still work fast. Otherwise one could try to generate meaningful alternative routes in advance. Investigating these aspects and also whether our qualitative results hold for other cities are clearly interesting questions that we plan to investigate further.

Acknowledgements. We wish to thank the Verkehrsbetriebe Zürich (VBZ) for providing historic real-world delay data.

References

1 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato Werneck. Route planning in transportation networks. Technical Report MSR-TR-2014-4, Microsoft Research, 2014.

2 Hannah Bast, Jonas Sternisko, Sabine Storandt, et al. Delay-robustness of transfer patterns in public transportation route planning. In *ATMOS*, pages 42–54, 2013.

3 Kateřina Böhmová, Matúš Mihalák, Tobias Pröger, Rastislav Šrámek, and Peter Widmayer. Robust routing in urban public transportation: How to find reliable journeys based on past observations. In *ATMOS*, pages 27–41, 2013.

4 Justin Boyan and Michael Mitzenmacher. Improved results for route planning in stochastic transportation. In *SODA*, pages 895–902, 2001.

5 Joachim M. Buhmann, Matúš Mihalák, Rastislav Šrámek, and Peter Widmayer. Robust optimization in the presence of uncertainty. In *ITCS*, pages 505–514, 2013.

6 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In *SEA*, pages 43–54, 2013.

7 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Delay-robust journeys in timetable networks with minimum expected arrival time. In *ATMOS*, pages 1–14, 2014.

8 Yann Disser, Matthias Müller-Hannemann, and Mathias Schnee. Multi-criteria shortest paths in time-dependent train networks. In *WEA*, pages 347–361, 2008.

9 Donatella Firmani, Giuseppe F. Italiano, Luigi Laura, and Federico Santaroni. Is timetabling routing always reliable for public transport? In *ATMOS*, pages 15–26, 2013.

10 H Frank. Shortest paths in probabilistic graphs. *Operations Research*, 17(4):583–599, 1969.

11 Marc Goerigk, Sacha Heße, Matthias Müller-Hannemann, Marie Schmidt, and Anita Schöbel. Recoverable robust timetable information. In *ATMOS*, pages 1–14, 2013.

12 Marc Goerigk, Martin Knoth, Matthias Müller-Hannemann, Marie Schmidt, and Anita Schöbel. The price of robustness in timetable information. In *ATMOS*, pages 76–87, 2011.

13 Sejoon Lim, Christian Sommer, Evdokia Nikolova, and Daniela Rus. Practical route planning under delay uncertainty: Stochastic shortest path queries. In *Robotics: Science and Systems VIII*, 2012.

14 Matthias Müller-Hannemann and Mathias Schnee. Efficient timetable information in the presence of delays. In *Robust and Online Large-Scale Optimization*, pages 249–272. Springer, 2009.

15 Evdokia Nikolova, Jonathan A Kelner, Matthew Brand, and Michael Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *ESA*, pages 552–563, 2006.

Bi-directional Search for Robust Routes in Time-dependent Bi-criteria Road Networks*

Matúš Mihalák[1] and Sandro Montanari[2]

1 Department of Knowledge Engineering, Maastricht University, The Netherlands
2 Department of Computer Science, ETH Zurich, Switzerland

Abstract

Based on time-dependent travel times for N past days, we consider the computation of robust routes according to the min-max relative regret criterion. For this method we seek a path minimizing its maximum weight in any one of the N days, normalized by the weight of an optimum for the respective day. In order to speed-up this computationally demanding approach, we observe that its output belongs to the Pareto front of the network with time-dependent multi-criteria edge weights. We adapt a well-known algorithm for computing Pareto fronts in time-dependent graphs and apply the bi-directional search technique to it. We also show how to parametrize this algorithm by a value K to compute a K-approximate Pareto front. An experimental evaluation for the cases $N = 2$ and $N = 3$ indicates a considerable speed-up of the bi-directional search over the uni-directional.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases shortest path, time-dependent, bi-criteria, bi-directional search, min-max relative regret

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.82

1 Introduction

The standard goal of *route planning* is the computation of a quickest route from a location s to a location t when departing at a time τ. Road networks are modeled as directed graphs with time-dependent edge weights representing travel times at a given period of time and a quickest route corresponds to a shortest s-t path in the time-dependent graph. The edge weights are usually an aggregation (e.g., average) of measured travel times of many individual cars over many similar time points. These aggregated values provide a good estimate of the expected travel time, yet only over a large amount of past days. They say little about deviations of the actual travel times and about per-day nuances of the traffic situations. In fact, a quickest path computed from such aggregated values can perform substantially bad on one particular day. In such situations a more appropriate goal is the computation of a *robust route* [22], that is a path offering guarantees on its travel time in the various situations we can encounter.

In this paper we consider the computation of robust routes in time-dependent road networks. Our focus is on speeding-up a particular method called *min-max relative regret*.

* This work was supported by the EU FP7/2007-2013 (DG CONNECT.H5-Smart Cities and Sustainability), under grant agreement no. 288094 (project eCOMPASS) and by the Swiss National Science Foundation (SNF) under the grant number 200021 138117/1.

The method looks for a route minimizing the maximum *normalized* travel time in the worst-case scenario. In our case, there are N scenarios (N past days), and the normalization is done with respect to the travel time of a quickest route in the respective day.

In a straightforward implementation, we enumerate all s-t paths in order of increasing travel times, alternatively for each day, until the desired s-t path is found. This implementation is however computationally extremely demanding and impractical. Thus, we observe that its output lies on the Pareto front of the road network with appropriate *time-dependent* and *multi-criteria* edge weights induced by the N instances. We adapt a known algorithm of Hansen/Martins [19, 23] for computing the Pareto front of such a graph and apply the speed-up technique of *bi-directional* search to it. To the best of our knowledge, our paper is the first one to consider speeding-up the computation of shortest paths in a setting where every component of the edge weights is time-dependent *and* multi-criteria.

2 Quickest Paths and Min-max Optimization

We consider a directed graph $G = (V, E)$ with time-dependent edge weights $w : E \times T \to \mathbb{N}$ defined over a given time horizon T. In our road network application $w(e, \tau)$ expresses the travel time needed to traverse the edge $e \in E$ when the vehicle enters e at time $\tau \in T$. This information for an edge $e \in E$ is obtained from a piecewise-linear function $f_e : T \to \mathbb{N}$ with period of one week, represented succinctly by a number of breakpoints. In our data-set the week is divided into 2016 breakpoints, one every 5 minutes. We can therefore compute $w(e, \tau)$ for any $\tau \in T$ in time $O(1)$.

A *path* P is a sequence $\langle v_1, \ldots, v_k \rangle$ of vertices where $(v_i, v_{i+1}) \in E$ for $i = 1, ..., k-1$. We overload the function w to express the travel time of $P = \langle v_1, ..., v_k \rangle$ departing at $\tau \in T$ as

$$w(P, \tau) = \begin{cases} 0 & \text{if } k = 1 \\ w\left((v_1, v_2), \tau\right) + w\left(\langle v_2, \ldots, v_k \rangle, \tau + w\left((v_1, v_2), \tau\right)\right) & \text{otherwise.} \end{cases}$$

A *quickest path* for given source $s \in V$, target $t \in V$, and time $\tau \in T$, is an s-t-path P minimizing $w(P, \tau)$. Note that in the above definition we do not allow waiting at vertices; this is clearly not beneficial if waiting at a vertex on an s-t path does not result in arriving at t earlier, a property formally defined as follows.

▶ **Definition 1** (FIFO Property)**.** A weight function $w : E \times T \to \mathbb{N}$ satisfies the *FIFO property* if for all $e \in E$ and all $\tau, \tau' \in T$ with $\tau \leq \tau'$ it holds $\tau + w(e, \tau) \leq \tau' + w(e, \tau')$.

If the edge weights satisfy the FIFO property, a quickest path can be computed efficiently using a generalization of Dijkstra's algorithm [11]. Since road networks are known to satisfy this property, we therefore focus on algorithms not waiting at nodes.

Given a departure time $\tau \in T$, vertices $s, t \in V$, and a finite discrete set of *scenarios*, the problem we consider is that of computing an optimum s-t path according to the *min-max relative regret* criterion. A scenario or instance I_i is specified by an edge weight function $w_i : E \times T \to \mathbb{N}$. The *relative regret* of an s-t path P in instance I_i is the ratio between its weight and the weight OPT_i of a quickest path in I_i. Given N different instances $I_1, \ldots, I_N$, we want to compute a path minimizing its maximum relative regret. In other words, we look for a path

$$\arg\min_P \max_i \left\{ \frac{w_i(P, \tau)}{OPT_i} \right\}. \tag{1}$$

The motivation for the study of this problem comes from its relation to robust optimization. For example, Buhmann et al. [4] define a general framework for robust optimization according

to several different criteria. Among these criteria, the one denoted by the authors as "First intersection" corresponds to the min-max relative regret.

Problem (1) is similar in nature to the problems of optimizing according to the *min-max absolute* or the *min-max deviation* criteria [25], the difference lying only in the normalization factor. In the former we look for a path $\arg\min_P \max_i \{w_i(P,\tau)\}$, while in the latter the goal is $\arg\min_P \max_i \{w_i(P,\tau) - OPT_i\}$. All these problems are NP-hard even for 2 instances; hardness for the min-max absolute and the min-max deviation criteria was proven by Yu and Yang [25], while for the min-max relative regret it follows straightforwardly as a reduction from the constrained shortest path problem [13].

Since a worst-case efficient algorithm is unlikely to be found, we consider efficiency only from a practical perspective. Even though our theoretical results can be generalized for any number N of scenarios, for practical reasons we focus on the particular case $N = 2$. As additional motivation for considering a small number of scenarios, we observe that approaches based on the min-max or min-max (relative) regret criteria are of more interest in these cases. If the number of scenarios is large, approaches based on statistical analysis are typically more efficient and produce results of similar if not better quality. In Section 3 we prove that there always exists an optimum solution lying on the Pareto front of all s-t paths. In Section 5 we design a bi-directional algorithm for computing such a front that can be parametrized by a factor K in order to compute a K-approximate solution. In Section 6 we experimentally show that, on a road network of the Berlin and Brandenburg area, the speed-up of the bi-directional search over uni-directional is considerable. A preliminary evaluation of the algorithm for the case $N = 3$ seems to indicate that this speed-up scales with the number of instances. In Section 7 we propose a simple modification to the bi-directional algorithm exploiting the correlation between the travel times of the different instances.

3 Relation to Bi-criteria Quickest Paths

For the case of 2 scenarios, the instances I_1 and I_2 with edge-weight functions respectively w_1 and w_2 induce a bi-criteria weight function $w : E \times T \to \mathbb{N}^2$ defined as

$$w(e,\tau) = \begin{pmatrix} w_1(e,\tau) \\ w_2(e,\tau) \end{pmatrix}. \tag{2}$$

Again, we overload the definition of w to express the weight of a path P as

$$w(P,\tau) = w(P',\tau) + \begin{pmatrix} w_1(e, \tau + w_1(P',\tau)) \\ w_2(e, \tau + w_2(P',\tau)) \end{pmatrix}, \tag{3}$$

where P' is the path obtained from P without the last hop e. Given two s-t paths P and P', we say that P *dominates* P' if $w_i(P,\tau) \leq w_i(P',\tau)$ for all $i \in \{1,2\}$, with the inequality being strict for some i. If two paths have the same weight in both components, they are said to be *equivalent*. The *Pareto front* of a set of paths $\mathcal{P}$ is the subset of all paths in $\mathcal{P}$ that are not dominated by another path in $\mathcal{P}$. For the sake of readability, in the following we will assume that no two paths are equivalent. It is well known [1] that an optimum path for Problem (1) lies in the Pareto front $\mathcal{F}$ of all s-t paths departing at $\tau \in T$. The following theorem proves a slightly stronger statement.

▶ **Theorem 2.** *Let $\mathcal{F}_\rho$ be the Pareto front of all optimum paths of Problem (1). Then, $\mathcal{F}_\rho \subseteq \mathcal{F}$.*

Proof. Assume towards contradiction that there exists a path $P \in \mathcal{F}_\rho \setminus \mathcal{F}$. Then, there is a path $P' \notin \mathcal{F}_\rho$ dominating P. For $i \in \{1, 2\}$, we let

$$\rho'_i = \frac{w_i(P', \tau)}{OPT_i} \leq \frac{w_i(P, \tau)}{OPT_i} = \rho_i,$$

Note that the relative regret of an optimum path is $\rho^* = \max\{\rho_1, \rho_2\}$ and that $\max\{\rho'_1, \rho'_2\} > \rho^*$. If $\max\{\rho'_1, \rho'_2\} = \rho'_i$ for some $i \in \{1, 2\}$ we get a contradiction, because

$$\rho'_i \leq \frac{w_i(P, \tau)}{OPT_i} \leq \rho^* < \rho'_i.$$ ◀

Theorem 2 implies that an optimum path for Problem (1) can be computed by enumerating all paths in $\mathcal{F}$ and picking one with smallest relative regret. Note that there may exist paths in $\mathcal{F}$ that are not optima, typically the quickest paths in either of the two instances. It is straightforward to prove Theorem 2 also for the min-max absolute and the min-max deviation criteria, implying that the bi-directional search algorithm proposed in the second half of this paper can be applied for those criteria as well. We further observe that the paths in $\mathcal{F}_\rho$ might not be extreme points of the convex hull of $\mathcal{F}$. This observation rules out the possibility of adopting known algorithms for the computation of such points [6, 12, 14].

▶ Remark. The definition in eq. (3) might appear unusual to a reader familiar with bi-criteria quickest path problems. In the literature it is more typically assumed that one of the two criteria of the weight of a path is its travel time while the other one is a cost depending on the travel time (for example, fuel consumption). Such a weight function can be written as

$$w(P, \tau) = w(P', \tau) + \begin{pmatrix} w_1(e, \tau + w_1(P', \tau)) \\ w_2(e, \tau + w_1(P', \tau)) \end{pmatrix}. \tag{4}$$

Note the difference in the time at which the second component is evaluated. Since our target application is robust routing, we however need to consider different travel times for the same path and hence use the definition in eq. (3). Under similar assumptions on the FIFO property of the edge weights, our results can be generalized for eq. (4) as well.

4 Related Work

We now consider the computation of an optimum path for Problem (1) by means of time-dependent multi-criteria optimization. Our aim is to apply the speed-up technique *bi-directional search* to an algorithm by Martins for computing Pareto fronts and experimentally investigate the improvements to its running time on road networks. In spite of its relevance, the literature about the problem is scant, and not many practical algorithms are known. The most closely related work is by Batz and Sanders [3] that consider the computation of shortest paths in a graph with multi-criteria edge weights where only one of the components is time-dependent. A great amount of work has been however invested by the community into the speed-up of routing algorithms in settings where edge weights are either only time-dependent [2, 7, 21] or only multi-criteria [8, 10].

Hansen [19] introduced several variants of bi-criteria shortest path problems and a pseudo-polynomial time algorithm computing the Pareto front of a graph with static non-negative bi-criteria edge weights. Martins [23] generalized this algorithm to static edge weights with more than two criteria. His algorithm keeps a priority queue of temporary labels Q and a set of permanent labels π_u for every vertex $u \in V$. Each label $(u, \boldsymbol{\omega})$ represents a path from s to u with weight $\boldsymbol{\omega} \in \mathbb{N}^k$ (for k criteria); we write $P \in \pi_u$ to indicate that the label

Listing 1 Time-dependent Martins' algorithm.

```
∀v ∈ V : π_v := ∅
Q.insert(s, (0 0))
{Compute front}
while Q ≠ ∅ do
 (u, ω) := Q.extract_min()
 for e = (u, v) ∈ E do
  ν := (v, ω + (w1(e,τ+ω1) w2(e,τ+ω2)))
  if ¬π_v.dominates(ν) and ¬π_t.dominates(ν) then Q.insert(ν)
```

representing P is in π_u. At the beginning every π_u is empty, and a label $(s, \mathbf{0})$ is created and put into Q. At each iteration the algorithm extracts from Q the smallest label $(u, \boldsymbol{\omega})$ in lexicographical order and puts it into π_u. A new label $(v, \boldsymbol{\nu})$ is then generated for each vertex v that can be reached from u, with $\boldsymbol{\nu} = \boldsymbol{\omega} + w(u, v)$. If no label in π_v or π_t dominates the new one, it is inserted into Q and all labels corresponding to s-v paths that are dominated by $(v, \boldsymbol{\nu})$ are removed from Q. The algorithm ends when Q is empty; at this point, π_t contains labels representing all paths in the Pareto front $\mathcal{F}$. By storing labels in Q and in all π_v in lexicographical order, we can implement the operations of extract minimum, insertion, and dominance checking to run, for the bi-criteria case, in logarithmic time. For a number of criteria larger than 2 it is currently not known how to efficiently implement these operations.

Gräbener et al. [17] provide an experimental evaluation of a straightforward time-dependent extension of Martins' algorithm, shown in Listing 1, on some publicly accessible networks. The correctness of this extension crucially depends on the FIFO property. Hamacher et al. [18] consider the setting where the FIFO property does not hold, and provide algorithms computing the Pareto front for a given s-t pair as well as for the all-to-all variant.

Dijkstra's algorithm for finding shortest s-t paths in the static single-criteria case gradually grows a shortest-path tree from s. At any step, each vertex is in one of the following states: UNREACHED, SETTLED, or DISCOVERED. A vertex is UNREACHED if its distance from s is not known, it is SETTLED if its distance from s is known exactly, and it is DISCOVERED if only an upper bound on the distance is known. At every iteration the algorithm introduces a new edge in the shortest path tree and sets its tail vertex as SETTLED. The algorithm terminates when t is SETTLED. In the worst-case the tree contains all vertices, even though we are only interested in those on the shortest s-t path that is returned.

The idea behind the *bi-directional search* [15, 16] is to grow two trees rooted at s and t using Dijkstra's algorithm alternatively from s and from t. The execution from t, called *backward search*, uses the edges of the reverse graph, i.e., the graph containing the edges of the original one in reverse direction. As soon as a vertex v is SETTLED by both the forward and the backward search the algorithm terminates and a shortest s-t path is guaranteed to lie in the union of the so-far constructed shortest path trees (such a path might however not pass through v). Any alternation works correctly; a typical choice is to balance the number of iterations of the two searches.

For static multi-criteria edge weights one can apply the bi-directional search by replacing Dijkstra's algorithm with Martins'. Since the goal is to compute the whole Pareto front of s-t paths (and not only a single path), the stopping criterion is however different. Demeyer et al. [9] show that terminating the computation when the sum of the *point-wise minima* of

the forward and backward queues is dominated by the front computed so far ensures that the Pareto front is found. The *point-wise minimum* of a queue Q, denoted as Q.p_min(), is the vector where each component is equal to the minimum among all labels in Q for the corresponding criterion.

When the edge weights are time-dependent, even in the single-criterion case, applying the bi-directional search is not straightforward anymore: the input consists of s, t, and the departure time τ. Thus, we can grow a tree from s starting at time τ, but we do not know the time τ' from which we shall start growing the tree from t – ideally, τ' is the earliest arrival time at t, but that is the number we wish to compute. A way to overcome this difficulty is to make the backward search static: for each edge (v, u) of the reverse graph $\overleftarrow{G} = (V, \overleftarrow{E})$, use a static weight defined as

$$\overleftarrow{w}((v,u)) = \min_{\tau \in T} \{w((u,v),\tau)\}. \tag{5}$$

Nannicini et al. [24] propose a bi-directional algorithm using the weights in eq. (5) working in three phases. In phase 1 the forward and backward search run alternatively until a vertex is DISCOVERED in both directions, resulting in an upper bound μ on the weight of a quickest path. In phase 2 both searches continue until the distances of all the DISCOVERED vertices in the backward queue are at least μ. In phase 3 only the forward search continues, with the constraint that only vertices that were SETTLED by the backward search are considered. In the following we show how to apply this idea to the time-dependent multi-criteria case.

5 Bi-directional Time-dependent Martins' Algorithm

A bi-directional algorithm for edge weights that are both time-dependent *and* bi-criteria can be designed by straightforwardly combining the ideas of Demeyer et al. and of Nannicini et al. This results in a three-phases search using Martins' algorithm both from s and from t, where the edge weights in the reverse graph are defined as in eq. (5) for both criteria. The termination condition of the backward search (i.e., the end of phase 2) is the stopping condition of Demeyer et al. As it turns out, however, this trivial algorithm can be improved considerably.

A critical observation to improve the straightforward algorithm is to note that in the backward direction our only interest is to identify vertices that might be on a Pareto optimal path. In other words, to determine whether or not the Pareto front of a given vertex contains at least one "promising" label. However, a label that is good for one criterion might not be good for the other one and we cannot know in advance which labels are promising. Our solution is to consider only the pointwise minima of the Pareto front π_v of each vertex v.

If the only purpose of the backward search is to compute pointwise minima, then Martins' algorithm is more than what is necessary. We can instead implement the backward search as two independent Dijkstra's runs on the reverse graph for each criterion. We modify the three phases of the bi-directional algorithm according to this observation as follows.

For phase 2, suppose we have found (in some way during phase 1) a number of non-necessarily Pareto-optimal s-t paths, and let M denote the Pareto front of these paths, while $\overrightarrow{Q}$ is the forward queue and $\overleftarrow{Q_1}, \overleftarrow{Q_2}$ are the backward queues. Suppose further that at some point during the computation, the weight of a path in M dominates

$$\beta := \overrightarrow{Q}.\text{p\_min}() + \begin{pmatrix} \overleftarrow{Q_1}.\min() \\ \overleftarrow{Q_2}.\min() \end{pmatrix}.$$

At this point, if a vertex v has not been SETTLED by both backward searches, then the weight of any s-t path through v is dominated by β and therefore by a path in M (we prove the correctness of this argument formally in the following). We can thus terminate phase 2 and the backward searches as soon as a path in M dominates β.

For phase 3 consider the situation where the forward search created a label $(v, \boldsymbol{\omega})$ to insert into $\overrightarrow{Q}$. Let the vector of distances computed by the backward searches for v be $v.\boldsymbol{d}$, the value of the second term of β (the minima of the backward queues) at the end of phase 2 be $\overleftarrow{\beta}$, and the minimum between $v.\boldsymbol{d}$ and $\overleftarrow{\beta}$ in each component be $\boldsymbol{\theta}$; that is, for $i \in \{1, 2\}$, we define $\theta_i := \min\{v.d_i, \overleftarrow{\beta_i}\}$. At the beginning of the computation $v.d_i$ is set to ∞ and at the end it holds that $v.d_i \leq \overleftarrow{\beta_i}$ if v has been SETTLED by the i-th backward search. If in phase 3 a path in M dominates $\boldsymbol{\omega} + \boldsymbol{\theta}$ then no path with the same s-v prefix as $\boldsymbol{\omega}$ can be optimal. We can thus discard all labels $(v, \boldsymbol{\omega})$ for which $\boldsymbol{\omega} + \boldsymbol{\theta}$ is dominated by a path in M.

According to the above phases, the purpose of phase 1 is the computation of a suitable tentative front M. Intuitively, a tentative front is good if the domination of β happens as early as possible, because less labels carry over to phase 3. We propose to terminate phase 1 as soon as a vertex v with $\pi_v \neq \emptyset$ is DISCOVERED by both backward searches and set M as the corresponding set of s-t paths passing through v. This strategy has the advantage of being efficient while at the same time being simple to implement. We note however that it is easy to come up with different strategies; it is an interesting open question to identify an optimum one.

To summarize, the three phases of the bi-directional algorithm are in detail as follows:

Phase 1 We let the forward and the backward search run alternatively. In the forward direction we use the time-dependent Martins' algorithm. In the backward direction we use two independent runs of Dijkstra's algorithm, one per criterion, using edge weights as in eq. (5). This phase ends as soon as a vertex v with $\pi_v \neq \emptyset$ is DISCOVERED by both backward searches. At the termination of the phase we let M be the Pareto front of the s-v paths in π_v concatenated with the v-t paths discovered in the backward direction.

Phase 2 Both the forward and the backward searches continue to run as in phase 1, until a path in M dominates β.

Phase 3 Only the forward search continues, with the constraint that labels $(v, \boldsymbol{\omega})$ for which $\boldsymbol{\omega} + \boldsymbol{\theta}$ is dominated by a path in M are ignored. This phase terminates when $\overrightarrow{Q}$ becomes empty.

The pseudocode of the algorithm under the name of BiTdMartins is illustrated by Listing 2. We use ϕ to denote the current phase and $\leftrightarrow$ to denote either the forward ($\leftrightarrow = \rightarrow$) or the backward search ($\leftrightarrow = \leftarrow$). The command $\leftrightarrow \in \{\rightarrow, \leftarrow\}$ selects the direction for the current iteration according to the alternation strategy; in our implementation we alternate between one iteration of the forward search and one iteration for each backward search. Note that to check the termination condition of phase 1 it is not necessary to search through all vertices. It is sufficient to check whether the condition holds only for the vertex extracted at the current iteration.

In the algorithm of Nannicini et al. [24] phase 2 terminates when the upper bound μ computed in phase 1 is at most the minimum of the backward queue. The authors proved that replacing this condition with one that, for a fixed parameter K, checks whether μ is at most K times the minimum of the backward queue results in an algorithm computing a K-approximate quickest path (i.e., a path with weight at most K times the weight of a quickest path). The following theorem shows that BiTdMartins satisfies a similar property. A corollary of this theorem, obtained by setting $K = 1$, implies correctness of the algorithm in the exact variant.

Listing 2 Algorithm BiTdMartins.

```
1  M := ∅, φ := 1,   ∀v ∈ V : π_v := ∅, v.d := (∞, ∞)
2  Q→.insert(s, (0, 0)), Q←_1.insert(t, 0), Q←_2.insert(t, 0)
3  while Q→ ≠ ∅ do
4   if φ = 3 then ↔ := → else ↔ ∈ {→, ←}
5   {terminate phase 1}
6   if φ = 1 and ∃v ∈ V : π_v ≠ ∅ and v.d_1 ≠ ∞ and v.d_2 ≠ ∞ then
7     M.insert(< s-t paths through v >), φ := 2
8   {terminate phase 2}
9   if φ = 2 and M.dominates(β) then φ := 3
10  {relax edges}
11  if ↔ = → then
12   (u, ω) := Q→.extract_min()
13   for e = (u, v) ∈ E→ do
14    ν := ω + (w_1(e, τ+ω_1), w_2(e, τ+ω_2))
15    if φ = 3 and M.dominates(ν + θ) then continue
16    if ¬π_v.dominates(ν) and ¬π_t.dominates(ν) then Q→.insert(v, ν)
17  else
18   for i ∈ {1, 2} do
19    u_i := Q←_i.extract_min()
20    for e = (u_i, v) ∈ E← do
21     if u_i.d_i + w←_i(e) < v.d_i then Q←_i.insert(v, u_i.d_i + w←_i(e))
```

▶ **Definition 3.** Given $\tau \in T$ and $K \geq 1$, we say that a path P is a *K-approximation* of another path P' if $w(P,\tau)$ dominates or is equivalent to $K \cdot w(P',\tau)$. Given two sets of paths $\mathcal{P}$ and $\mathcal{P}'$, we say that $\mathcal{P}$ is a *K-approximation* of $\mathcal{P}'$ if every path in $\mathcal{P}'$ is K-approximated by a path in $\mathcal{P}$.

▶ **Theorem 4.** *Given $K \geq 1$, if we replace the condition to terminate phase 2 with M.dominates$(K \cdot \beta)$ and in phase 3 we discard all labels $(v, \boldsymbol{\nu})$ such that M.dominates$(K \cdot (\boldsymbol{\nu} + \boldsymbol{\theta}))$, then* BiTdMartins *computes a K-approximation of $\mathcal{F}$.*

Proof. Assume there exists a path $P \in \mathcal{F}$ not K-approximated by a path in π_t. That is, for every $P' \in \pi_t$ there is $i \in \{1,2\}$ such that

$$K \cdot w_i(P,\tau) < w_i(P',\tau). \tag{6}$$

Let P_{sv} be the prefix of P from s to the first vertex v such that $P_{sv} \notin \pi_v$, and P_{vt} be the suffix of P from v to t. Since $P_{sv} \notin \pi_v$, there is a path in M dominating $K \cdot (w(P_{sv},\tau) + \boldsymbol{\theta})$. Let P' be either this path, if it belongs to π_t, or a path in π_t dominating it otherwise. Note that for all $i \in \{1,2\}$ it holds $\theta_i \leq \overleftarrow{w_i}(P_{vt})$ since, if v was settled by the i-th backward search, then $\theta_i = v.d_i = \overleftarrow{w_i}(P_{vt})$ while, if v was not settled by the i-th backward search, then $\theta_i = \overleftarrow{\beta_i} \leq \overleftarrow{w}(P_{vt})$. Supposing without loss of generality that eq. (6) holds for $i = 1$ we obtain a contradiction, because

$$K \cdot w_1(P,\tau) < w_1(P',\tau) \leq K \cdot (w_1(P_{sv},\tau) + \theta_i) \leq K \cdot (w_1(P_{sv},\tau) + \overleftarrow{w_1}(P_{vt})) \leq K \cdot w_1(P,\tau). \blacktriangleleft$$

Table 1 Run-time in milliseconds and average number of scanned labels.

	Max Rel Regret	Run-time (ms)	Labels Phase 1	Phase 2	Phase 3
Dijkstra	1.0711	261	–	–	220,620
Naive	1.0074	–	–	–	8,420
Uni-dir	1.0074	3,105	–	–	1,419,524
Bi-dir	1.0074	1,888	178,855	189,336	449,560
$K = 1.02$	1.0085	1,570	178,855	177,133	402,970
$K = 1.04$	1.0108	1,427	178,855	165,209	356,479
$K = 1.06$	1.0156	1,286	178,855	153,549	311,216
$K = 1.08$	1.0232	1,150	178,855	142,190	268,139
$K = 1.10$	1.0337	1,028	178,855	131,160	228,646
$K = 1.20$	1.0724	712	178,855	80,678	93,660
$K = 1.40$	1.0830	338	178,855	16,854	10,420
$K = 1.60$	1.0856	275	178,855	1,858	2,407
$K = 1.80$	1.0865	269	178,855	270	1,787
$K = 2.00$	1.0868	269	178,855	81	1,692

▶ **Corollary 5.** BiTdMartins *computes the Pareto front* $\mathcal{F}$.

Note that the converse of Theorem 4 in general does not hold. There might be paths in π_t not approximating a Pareto optimal path.

6 Computational Results

The experimental evaluation was performed on one core of an Intel Xeon E5-2697v2 processor clocked at 2.7 GHz and 64 GB main memory. The code was written in C++ and compiled using GNU C++ compiler version 4.8.2 and optimization level 3.

6.1 Input Road Network

The input data consists of a road network of the area around Berlin and Brandenburg kindly provided by TomTom within the project eCOMPASS [5]. The largest strongly connected component of the graph consists of 443,365 vertices and 1,038,284 edges. The travel times of 750,544 edges are constant, while for the remaining 287,740 edges are given by a piecewise-linear function with period of one week.

To obtain two instances (edge weight functions) I_1 and I_2 we consider departure times τ_1 and τ_2 in two consecutive days. We select uniformly at random one of the 24 hours of a day and let τ_1 be the corresponding point in time on Tuesday and τ_2 be the same time in the following Wednesday. The edge weight functions are obtained by setting the beginnings of the time horizon (in other words the departure times) of I_1 and I_2 respectively at τ_1 and at τ_2. We select 10,000 pairs of vertices s and t uniformly at random.

6.2 Results

Table 1 shows a comparison of the algorithms considered in terms of quality (i.e., the maximum relative regret of the computed path) and efficiency, averaged among the performed 10,000 tests. The efficiency of an algorithm is measured in terms of CPU time and the number of labels scanned for each phase of the algorithm. The number of labels scanned, i.e.,

Table 2 Run-time and number of scanned labels for 3 instances.

			Labels		
	Max Rel Regret	Run-time (ms)	Phase 1	Phase 2	Phase 3
Uni-dir	1.0328	954,267	-	-	7,927,858
Bi-dir	1.0328	487,993	210,374	212,609	3,678,017
$K = 1.2$	1.0861	190,960	210,374	96,108	1,161,266
$K = 1.4$	1.1013	53,154	210,374	23,114	339,865
$K = 1.6$	1.1068	6,180	210,374	4,209	76,958
$K = 1.8$	1.1095	1,670	210,374	1,159	16,383
$K = 2.0$	1.1095	975	210,374	197	2,805

the overall number of labels extracted from the forward and from the backward queues, represents a machine-independent measure of efficiency. The counter of labels scanned for the bi-directional algorithm is increased by one for each iteration of the forward search, and by 0.5 for each iteration of one of the two backward searches.

The algorithms considered for comparison are: the unidirectional search using the time-dependent implementation of Martins' algorithm, the bi-directional search of BiTdMartins, the K-approximate BiTdMartins for different values of K, and the naive algorithm for the min-max relative regret problem. As additional reference, the table also shows information on the computation of a quickest path in I_1 using the time-dependent Dijkstra's algorithm.

The naive algorithm enumerates all s-t paths alternatively for I_1 and I_2 until an optimum path is found; it is implemented as a straightforward time-dependent generalization of an algorithm by Hershberger et al. [20] for the computation of the k-shortest paths. The row corresponding to this algorithm in Table 1 does not show the number of labels scanned. Instead, we display the average number of iterations before finding a path in the intersection. Since each iteration consists of several (a number linear in n) repetitions of the time-dependent Dijkstra's algorithm, we can have an idea on the number of labels scanned by looking at the first two rows of the table.

The improvements of the bi-directional search over the uni-directional is considerable both for the run-time and for the number of scanned labels. The efficiency further increases if we allow an approximation factor K greater than 1. It appears however that there is a limit to the speed-up that can be obtained via approximation. The reason for this limit is that large values of K greatly reduce the amount of time spent by the algorithm in phases 2 and 3, but do not decrease the time in phase 1. In particular, for some value of K, say K^*, the algorithm spends no time at all in phase 2 because the termination condition is met as soon as the phase begins. All values of K greater than K^* will therefore result in similar run-time and number of scanned labels.

6.3 Results for 3 Instances

Table 2 shows experimental results for the case of 3 instances (the third day being Thursday). Since in this case the operations of extract minimum, insertion and dominance checking necessary to implement Martins' algorithm cannot be implemented efficiently, all algorithms are as a result much slower than in the previous case. For this reason, the experimental evaluation is not as thorough, and only 240 pairs of s, t vertices were considered.

We can see that the speed-up of the bi-directional search over the uni-directional is still considerable, and an even more remarkable speed-up can be obtained via approximation. By setting an approximation factor of $K = 2.0$, computations that in the exact case require in average 25 minutes to terminate can be performed in less than one second.

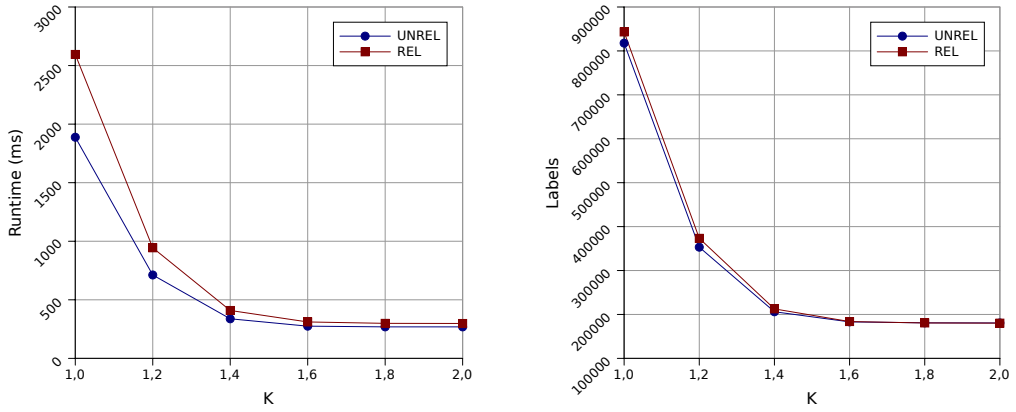


Figure 1 Comparison of bi-directional algorithms.

7 Single Backward Search

In our input data there is a strong correlation between the weights of a path in the different instances, since they represent travel times in very similar time periods. However, this correlation is not explicitly exploited by our algorithm. One might ask for a way to improve the efficiency of BiTdMartins by considering this feature more directly. For example, for the case of 2 instances we might get some improvement by replacing the two backward searches with a single one that, for each backward edge $e \in \overleftarrow{E}$, considers weights of the kind

$$\overleftarrow{w}(e) = \min_{i \in \{1,2\}} \{\overleftarrow{w_i}(e)\}\,.$$

The correctness of this algorithm and its approximated variants follows trivially from the previous proofs under the same assumptions as for BiTdMartin. The benefits of this new algorithm over the original one are however not trivial to estimate. On the one hand, if

$$\max_{e \in \overleftarrow{E}} \{|\overleftarrow{w_1}(e) - \overleftarrow{w_2}(e)|\} \tag{7}$$

is small, the modified backward search will settle almost the same vertices as before, with almost the same values, at the price of one execution of Dijkstra's algorithm instead of two. On the other hand, the lower bounds on the distances computed in the reverse graph are less accurate. As a result, the number of labels scanned by the modified algorithm is larger. The benefit of the modified algorithm is, in loose terms, inversely proportional to eq. (7).

Figure 1 shows a plot of the average run-time and the number of labels settled by the original bi-directional algorithm (unrel) and the modified one (rel) for different values K of approximation and for the same 10,000 s-t pairs. We can see that unrel is faster but it indeed settles more labels than rel. However, the difference between the two is very small and further decreases for increasing values of K until the point where the performance of the two algorithms is almost equal. It is an interesting open question to identify cases where the benefit of a single backward search takes over both the run-time and the number of labels.

8 Conclusions

We have considered the problem of computing an optimum path according to the min-max relative regret criterion and shown that there always exists one such path on the Pareto front

of a multi-criteria weight function. We have therefore engineered a bi-directional algorithm for the computation of Pareto fronts in time-dependent multi-criteria graphs and experimentally demonstrated a considerable speed-up compared to the uni-directional variant.

We observe that the presented results appear of practical interest for the application of robust routing. A peculiarity of this application is that the considered criteria correspond to travel times for different days of the week. If the days considered are correlated like, for example, working days as opposed to working days and holidays, we expect this correlation to somehow appear in the travel times as well. As a result, the number of paths in the Pareto fronts is not too big; for our experiments, the average size of the fronts is 8 (although for the case of 3 instances this number increases to 40). It is an interesting open question to assess the practical efficiency of the proposed algorithms for multi-criteria edge weights inducing fronts of larger size, such as those considered by Delling and Wagner [8]. Furthermore, an assessment of the robustness of the routes computed using the min-max relative regret criterion on the Berlin and Brandenburg data-set is planned for a follow-up paper.

References

1 H. Aissi, C. Bazgan, and D. Vanderpooten. Min-max and min-max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197(2):427–438, 2009.

2 G. V. Batz, R. Geisberger, P. Sanders, and C. Vetter. Minimum time-dependent travel times with contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 18, 2013.

3 G. V. Batz and P. Sanders. Time-dependent route planning with generalized objective functions. In *ESA*, pages 169–180, 2012.

4 J. M. Buhmann, M. Mihalák, R. Šrámek, and P. Widmayer. Robust optimization in the presence of uncertainty. In *ITCS*, pages 505–514, 2013.

5 European Commission. eCOMPASS Project. `http://www.ecompass-project.eu/`, 2011-2014.

6 C. Daskalakis, I. Diakonikolas, and M. Yannakakis. How good is the chord algorithm? *CoRR*, abs/1309.7084, 2013.

7 D. Delling. Time-dependent SHARC-routing. *Algorithmica*, 60(1):60–94, 2011.

8 D. Delling and D. Wagner. Pareto paths with SHARC. In *SEA*, pages 125–136, 2009.

9 S. Demeyer, J. Goedgebeur, P. Audenaert, M. Pickavet, and P. Demeester. Speeding up Martins' algorithm for multiple objective shortest path problems. *4OR*, 11(4):323–348, 2013.

10 S. Erb, M. Kobitzsch, and P. Sanders. Parallel bi-objective shortest paths using weight-balanced B-trees with bulk updates. In *SEA*, pages 111–122, 2014.

11 L. Foschini, J. Hershberger, and S. Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4):1075–1097, 2014.

12 S. Funke and S. Storandt. Polynomial-time construction of contraction hierarchies for multi-criteria objectives. In *ALENEX*, pages 41–54, 2013.

13 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

14 R. Geisberger, M. Kobitzsch, and P. Sanders. Route planning with flexible objective functions. In *ALENEX*, pages 124–137, 2010.

15 A. V. Goldberg and C. Harrelson. Computing the shortest path: *A** search meets graph theory. In *SODA*, pages 156–165, 2005.

16 A. V. Goldberg and R. F. Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX*, pages 26–40, 2005.

17 T. Gräbener, A. Berro, and Y. Duthen. Time dependent multiobjective best path for multimodal urban routing. *Electronic Notes in Discrete Mathematics*, 36, 2010.

18 H. W. Hamacher, S. Ruzika, and S. A. Tjandra. Algorithms for time-dependent bicriteria shortest path problems. *Discrete Optimization*, 3(3):238–254, 2006.

19 P. Hansen. Bicriterion path problems. In *Multiple Criteria Decision Making Theory and Application*, pages 109–127. Springer Berlin Heidelberg, 1980.

20 J. Hershberger, M. Maxel, and S. Suri. Finding the k shortest simple paths: A new algorithm and its implementation. *ACM Transactions on Algorithms*, 3(4), 2007.

21 S. C. Kontogiannis and C. D. Zaroliagis. Distance oracles for time-dependent networks. In *ICALP 2014*, pages 713–725, 2014.

22 P. Kouvelis and G. Yu. *Robust discrete optimization and its applications*, volume 14. Springer Science & Business Media, 2013.

23 E. Q. V. Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, 1984.

24 G. Nannicini, D. Delling, D. Schultes, and L. Liberti. Bidirectional A^* search on time-dependent road networks. *Networks*, 59(2):240–251, 2012.

25 G. Yu and J. Yang. On the robust shortest path problem. *Computers & OR*, 25(6):457–468, 1998.

A Computing the Pareto Front

We provide as reference the proof of correctness for the time-dependent Martins' algorithm. The analysis of its run-time follows trivially from the one by Hansen [19].

▶ **Theorem 6.** *Let $G = (V, E)$ be a graph with edge weights $w : E \times T \to \mathbb{N}^2$ as in eq. (2). If $w_i : E \times T \to \mathbb{N}$ satisfies the FIFO property for every $i \in \{1, 2\}$, Listing 1 computes the Pareto front $\mathcal{F}$.*

Proof. The computed front π_t is not correct if there is a path in $\mathcal{F}$ that is not in π_t, there is a path in π_t that is not in $\mathcal{F}$, or both. We consider only the first case, since the remaining two follow from the fact that if π_t contains at least the paths in $\mathcal{F}$ then all other paths are dominated by those.

Suppose towards contradiction that there exists $P \in \mathcal{F}$ such that $P \notin \pi_t$. Consider the prefix P_{sv} of P from s to the first vertex v such that $P_{sv} \notin \pi_v$, and the suffix P_{vt} of P from v to t. We can express the weight of P as

$$w(P, \tau) = w(P_{sv}, \tau) + \begin{pmatrix} w_1(P_{vt}, \tau + w_1(P_{sv}, \tau)) \\ w_2(P_{vt}, \tau + w_2(P_{sv}, \tau)) \end{pmatrix}.$$

Since $P_{sv} \notin \pi_v$, there exists another path P'_{sv} dominating it, and we can obtain an s-t path P' (not necessarily simple) by concatenating P'_{sv} and P_{vt}. The weight of P' can be written as

$$w(P', \tau) = w(P'_{sv}, \tau) + \begin{pmatrix} w_1(P_{vt}, \tau + w_1(P'_{sv}, \tau)) \\ w_2(P_{vt}, \tau + w_2(P'_{sv}, \tau)) \end{pmatrix}.$$

Since P'_{sv} dominates P_{sv}, we know that, for every $i \in \{1, 2\}$, it holds that

$$w_i(P'_{sv}, \tau) \leq w_i(P_{sv}, \tau).$$

Since both w_1 and w_2 satisfy the FIFO property, we get that $w(P', \tau)$ dominates $w(P, \tau)$. This contradicts the assumption that $P \in \mathcal{F}$. ◀

▶ **Corollary 7.** *The run-time of Listing 1 is $O(nmW \cdot \log(nW))$, where*

$$W = \min_{i \in \{1,2\}} \left\{ \max_{e \in E, \tau \in T} w_i(e, \tau) \right\}.$$

A Mixed Integer Linear Program for the Rapid Transit Network Design Problem with Static Modal Competition

Gabriel Gutiérrez-Jarpa[1], Gilbert Laporte[2], Vladimir Marianov[3], and Luigi Moccia[4]

1 **School of Industrial Engineering, Pontificia Universidad Católica de Valparaíso, Chile**
gabriel.gutierrez@ucv.cl
2 **HEC Montréal, Canada**
gilbert.laporte@cirrelt.ca
3 **Department of Electrical Engineering, Pontificia Universidad Católica de Chile, Chile**
marianov@ing.puc.cl
4 **Istituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche, Italy**
moccia@icar.cnr.it

Abstract

In recent years, several models and algorithms have been put forward for the design of metro networks (see e.g. [1], [7], [2], and [6]). Here we extend the rapid transit network design problem (RTNDP) of [4] by introducing modal competition and by enriching its multi-objective framework. In that reference an origin-destination flow is considered as captured by rapid transit if some stations are sufficiently close to both the origin and the destination of the flow. We observe that by maximizing the captured traffic using this criterion results in improving *access*, i.e. the number of commuters who could benefit from the rapid transit network for their daily trips. This is indeed a relevant goal in urban transit, but on its own it does not adequately reflect modal choices. In this talk we consider a traffic flow as captured if the travel time (or equivalently the generalized cost) by rapid transit is less than by car, i.e. an "all or nothing" criterion. This feature has been neglected in most previous discrete mathematical programs because considering origin-destination flows results in models that are too large for realistic instances. As observed by [8], considering traffic flows requires a multi-commodity formulation, where each flow is considered as a distinct commodity. This was the approach taken by [5], but it only allowed the solution of very small instances. We introduce a methodology that overcomes this difficulty by exploiting a pre-assigned topological configuration. As explained by [1], a pre-assigned topological configuration is in itself a positive feature for planners since it incorporates their knowledge of the traffic flows in cities and corresponds to what is often done in practice. We note that a metro network is typically built incrementally starting from a simple layout. Very often planners identify a few major corridors that should be privileged for an initial metro configuration or for later extensions. Geographical constraints may also limit the number options. We remark that, despite the simple layouts, the high number of location choices for each layout renders the problem hard. The precise alignment of metro lines within these corridors can be optimized by using a methodology such as the one we propose. The basic topological configurations we consider are not limitative in the sense that our approach will work with any basic layout. We note, however, that simple layouts such as stars and triangles exist in several networks (e.g. Minsk and Prague). With time the basic networks evolve into more complex, but still common configurations. [10] find that metro networks converge to a shape characterized by a core from which quasi-one-dimensional branches grow and reach out to areas of the city further from it. We discuss relevant goals of rapid transit planning, and we propose a multi-objective model conducive to a post-optimization

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt; pp. 95–96

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

analysis for effectiveness, efficiency, and equity concerns. The multi-objective framework works with two alternative measure of effectiveness under a budget constraint, and a post-optimization phase is suggested to assess efficiency and equity trade-offs, an issue rarely considered in transit planning (see for example [3] for a review of equity in location problems, and [9] for a discussion of equity in urban transit). We show that our approach can be applied to realistic situations and we illustrate it on data from Concepción, Chile.

1998 ACM Subject Classification Decision support

Keywords and phrases metro network design, multi-objective optimization, modal competition

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.95

Category Short Paper

Acknowledgements. Gabriel Gutiérrez-Jarpa was supported by FONDECYT project No. 1130878 and Becas Chile. Gilbert Laporte was supported by the Canadian Natural Sciences and Engineering Research Council under grant 39682-10. Vladimir Marianov was partially supported by the Complex Engineering Systems institute, through grants ICM-MIDEPLAN P-05-004-F and CONICYT FB016. Luigi Moccia was partially supported by the CNR (Italy) under STM-2014 and STM-2015 grants. These supports are gratefully acknowledged.

References

1 G. Bruno and G. Laporte. An interactive decision support system for the design of rapid public transit networks. *INFOR*, 40(2):111–118, 2002.

2 D. Canca, A. De-Los-Santos, G. Laporte, and J. A. Mesa. A general rapid network design, line planning and fleet investment integrated model. *Annals of Operations Research*, In Press:1–18, 2014.

3 H. A. Eiselt and G. Laporte. Objectives in location problems. In Z. Drezner, editor, *Facility Location: A Survey of Applications and Methods*, Springer Series in Operations Research and Financial Engineering, chapter 8, pages 151–180. Springer-Verlag, New York, 1995.

4 G. Gutiérrez-Jarpa, C. Obreque, G. Laporte, and V. Marianov. Rapid transit network design for optimal cost and origin-destination demand capture. *Computers & Operations Research*, 40(12):3000–3009, 2013.

5 G. Laporte, A. Marín, J. A. Mesa, and F. Perea. Designing robust rapid transit networks with alternative routes. *Journal of Advanced Transportation*, 45(1):54–65, 2011.

6 G. Laporte and J. A. Mesa. The design of rapid transit networks. In G. Laporte, S. Nickel, and F. Saldanha da Gama, editors, *Location Science*, pages 581–594. Springer, Berlin, Heidelberg, 2015.

7 G. Laporte, J. A. Mesa, F. A. Ortega, and I. Sevillano. Maximizing trip coverage in the location of a single rapid transit alignment. *Annals of Operations Research*, 136(1):49–63, 2005.

8 Á. Marín and R. García-Ródenas. Location of infrastructure in urban railway networks. *Computers & Operations Research*, 36(5):1461–1477, 5 2009.

9 A. Perugia, J.-F. Cordeau, G. Laporte, and L. Moccia. Designing a home-to-work bus service in a metropolitan area. *Transportation Research Part B: Methodological*, 45(10):1710–1726, 2011.

10 C. Roth, S. M. Kang, M. Batty, and M. Barthelemy. A long-time limit for world subway networks. *Journal of the Royal Society Interface*, 9(75):2540–2550, 2012.

Ordering Constraints in Time Expanded Networks for Train Timetabling Problems

Frank Fischer

Algorithmic Algebra and Discrete Mathematics
University of Kassel
Heinrich-Plett-Str. 40, 34132 Kassel, Germany
`frank.fischer@uni-kassel.de`

Abstract

The task of the *train timetabling problem* is to find conflict free schedules for a set of trains with predefined routes in a railway network. This kind of problem has proven to be very challenging and numerous solution approaches have been proposed. One of the most successful approaches is based on time discretized network models. However, one of the major weaknesses of these models is that fractional solutions tend to change the order of trains along some track, which is not allowed for integer solutions, leading to poor relaxations. In this paper, we present an extension for these kind of models, which aims at overcoming these problems. By exploiting a configuration based formulation, we propose to extend the model with additional *ordering constraints*. These constraints enforce compatibility of orderings along a sequence of tracks and greatly improve the quality of the relaxations. We show in some promising preliminary computational experiments that our approach indeed helps to resolve many of the invalid overtaking problems of relaxations for the standard models.

1998 ACM Subject Classification G.1.6 [Numerical Analysis] Optimization

Keywords and phrases combinatorial optimization, train timetabling, Lagrangian relaxation, ordering constraints

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.97

1 Introduction

Given a railway infrastructure network and a set of trains with fixed routes, the train timetabling problem (TTP) aims at determining schedules for each train such that certain operational restrictions like station capacities and headway times are satisfied, see, e. g., the recent surveys [14] and [5]. The possible goals for these schedules vary. Typical ones are to have the trains arrive as early as possible (often a goal for freight trains) [7] or follow a given ideal timetable with as less delay as possible (passenger trains) [4].

One major approach for large scale instances is based on time expanded networks for modeling train schedules [2, 6, 11]. These models give rise to huge integer programming formulations and cannot be solved directly by standard solvers. Recently, several mathematical techniques, e. g., dynamic graph generation [9], bundle methods [10], rapid branching [1, 17], were developed to overcome this situation.

In this paper we present an extension for time expanded models. One major drawback of these models is that they do not contain variables representing orderings of trains running on the same infrastructure arc or using the same station. This has the consequence that fractional relaxations (e. g., linear relaxations or Lagrangian relaxations) tend to find (fractional) solutions that allow overtaking of trains on tracks where it is not possible for integral solutions. This behavior is expected, but leads to very weak relaxations. Because of the

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt; pp. 97–110

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

lack of ordering variables it is not easy to strengthen the model with additional constraints that model the combinatorial properties of the infrastructure network, i. e., forbid arbitrary changes of the order of trains running on the same tracks.

Borndörfer and Schlechte [2] proposed a variant of the time expanded models, in which headway constraints on tracks are not enforced by inequality constraints, which forbid succeeding trains running in too quick succession. Instead they use so called *configuration networks*, which model sets of non-conflicting train runs on a single infrastructure arc. This means, while inequality constraints provide an "outer description" of feasible train runs, configurations networks form an "inner description", modeling all feasible points. Although both models are equivalent from a theoretical point of view, the latter has an additional advantage. Configuration networks form some kind of extended formulation and add further variables and constraints to the model. In particular, they allow easy access to a limited set of ordering variables for trains. Indeed, a configuration network provides decision variables that model whether two trains follow each other in direct succession.

In this paper we propose a model that exploits these ordering variables to forbid invalid changes of the ordering of trains along a sequence of succeeding tracks. We show that this model provides a stronger relaxation than previous models, resolving some of the nastiest weaknesses of linear relaxations for this kind of models for the TTP.

This paper is organized as follows. We give a formal description of the problem in Section 2 and state the basic time expanded model in Section 3. In particular, the new model extension with ordering constraints is described in Sections 3.2 and 3.3. Then we sketch our solution approach in Section 4 and present some promising preliminary computational experiments in Section 5. Finally, we conclude our paper in Section 6.

2 Problem Description

We briefly recall a formal description of the TTP next. The *infrastructure network* is a directed graph $G^I = (V^I, A^I)$, where the nodes V^I represent stations, junctions, and crossings and the arcs A^I represent connecting railway tracks. Arcs are directed because the running times of some trains may depend on the direction of travel and because some tracks are single line (one physical track that is used in both directions). In the latter case it is important to distinguish between trains running in the same and in opposite directions. Furthermore, we are given a set of trains R and each train $r \in R$ is associated with a path $G^r = (V^r, A^r) \subseteq G^I$ in the infrastructure network, its route, and a starting time $\hat{t}^r \in T$. Denote by R^a the set of all trains running on $a \in A^I$. Between two trains $r, r' \in R^a$ running on a there is a minimal headway time $h^a(r, r') \in \mathbb{N}$ (in discrete time steps), which is the minimal time between the two trains entering this track (we assume that $h^a(r, r') > 0$ for all $a \in A^I$, $r, r' \in R^a$, and that the triangle inequalities $h^a(r, r') + h^a(r', r'') \geq h^a(r, r'')$ for each three trains $r, r', r'' \in R^a$ are satisfied). As mentioned above, the infrastructure network may contain double line tracks (arcs with one physical track for each direction) and single line tracks (arcs with only one physical track for both directions). In the case of single line tracks we set $h^{(u,v)} = h^{(v,u)}$ and the headway times for trains running in opposite directions are appropriately high, so that no two such trains may occupy the track at the same time. Furthermore, there are capacity constraints on the nodes that state that at most a certain number of trains $c_u \in \mathbb{N}$ may be at the same station $u \in V^I$ at the same time. Note that in contrast to other works the schedules of the trains are completely free. There do not exist already fixed trains or conditions stating that schedules must not deviate too much from some ideal timetable (see, e. g., [4]).

▶ Remark. Note that the train routes are only a rough estimate of the reality. For instance, we do not consider the exact routing of trains through junctions as well as the possibility of small adjustments to the train routes, e. g., by choosing one of several parallel tracks. These aspects are often interesting in practice and can be incorporated in our model, but for sake of simplicity we decided to neglect these details.

In our experiments we consider only a simple but reasonable objective function, which aims at minimizing the delay of each train at each station. Here delay means the difference of the arrival time at some station compared with the earliest possible arrival time, if a train does not wait at some earlier station. The exact definition is given below in Section 3.1.

3 Model

In this section we present our time discretized model for the TTP. We start with the networks modeling the schedules of each train and some basic constraints in Section 3.1. Next we review the modeling of headway constraints based on configuration networks in Section 3.2. In Section 3.3 we present the new ordering constraints and finally the complete model in Section 3.4.

3.1 Basic Time Expanded Model

One of the most successful models in the literature for solving the TTP is based on time expanded networks, see, e. g., [6, 2]. Given a set of discrete time steps $T = \{1, \dots, |T|\}$ (usually minutes), we have for each train $r \in R$ a *time expanded network* $G_T^r = (V_T^r, A_T^r)$ where $V_T^r = V^r \times T$ and

$$\begin{aligned} A_T^r = &\{((u,t_u),(v,t_v)) \colon (u,v) \in A^r, t_u, t_v \in T, t_v - t_u = \bar{t}^r_{(u,v)}\} \\ &\cup \{((u,t_u),(u,t_u+1)) \colon u \in V^r_{\text{wait}}, t_u, t_u+1 \in T, t_u \geq \hat{t}^r\}, \end{aligned}$$

with $V^r_{\text{wait}} \subseteq V^r$ the nodes at which r might wait and $\bar{t}^r_{(u,v)} \in \mathbb{N}_0$ the *running time* of r over track $(u,v) \in A^r$. A feasible schedule of train r then corresponds to a path $P \subseteq G_T^r$ from the first to the last station. In particular, let $\hat{u}^r$, $\check{u}^r$ denote the first and the final station of r, respectively, then the variables $x^r \in \{0,1\}^{A_T^r}$ have to satisfy the following *flow conservation constraints*

$$\sum_{\substack{(v,t_v)\in V_T^r:\\ e=((v,t_v),(u,t_u))\in A_T^r}} x_e^r = \sum_{\substack{(v,t_v)\in V_T^r:\\ e=((u,t_u),(v,t_v))\in A_T^r}} x_e^r, \quad (u,t_u) \in V_T^r \setminus \{(\hat{u}^r,\hat{t}^r)\}, u \neq \check{u}^r, \tag{1a}$$

$$\sum_{\substack{(v,t_v)\in V_T^r:\\ e=((\hat{u}^r,\hat{t}^r),(v,t_v))\in A_T^r}} x_e^r = 1, \tag{1b}$$

$$\sum_{t\in T} \sum_{\substack{(v,t_v)\in V_T^r:\\ e=((v,t_v),(\check{u}^r,t))\in A_T^r}} x_e^r = 1. \tag{1c}$$

Constraints (1a) are the flow conservation constraints on all intermediate nodes. Constraint (1b) states that the path must start at node $(\hat{u}^r, \hat{t}^r) \in V_T^r$ and constraint (1c) enforces that the path must end at some node $(\check{u}^r, t) \in V_T^r$ corresponding to its final station. We denote the set of all feasible paths in G_T^r by

$$\mathcal{P}^r := \{x^r \in \{0,1\}^{A_T^r} \colon x^r \text{ satisfies constraints (1a)–(1c)}\}.$$

We associate a binary variable $x_a^r \in \{0,1\}$ with each $a \in A_T^r$ in each time expanded network, where $x_a^r = 1$ if and only if a is contained in the timetable of train r.

The capacity constraints in the nodes enforce that at each time instance $t \in T$ at most $c_u \in \mathbb{N}$ trains may be in $u \in V^I$ at the same time. Hence, the sum over all arcs representing a train being in u at time t

$$K(u,t) := \{(r,a) \colon a = ((u',t'),(u,t)) \in A_T^r, r \in R\}$$

must be at most c_u

$$\sum_{(r,a)\in K(u,t)} x_a^r \le c_u, \qquad u \in V^I, t \in T. \tag{2}$$

The headway restrictions impose that certain arcs must not be contained in the final timetable simultaneously if they correspond to some train runs violating a headway constraint. In particular, let $(r,a) \in A_T^r$ and $(r',a') \in A_T^{r'}$ be two arcs with $a = ((u,t_u),(v,t_v))$ and $a' = ((u,t'_u),(v,t'_v))$ with $t'_u - t_u < h^{(u,v)}(r,r')$, then those arcs must not be used both. Therefore, we have the following *headway constraints* for each pair of incompatible arcs

$$x_a^r + x_{a'}^{r'} \le 1, \qquad \{(r,a),(r',a')\} \in H, \tag{3}$$

where H is the set of pairs of incompatible train arcs.

The objective for all trains is to run as fast possible in order to minimize all delays. For this we use the simple objective function described in [7]. Let $(u,v) \in A^r$ be a track segment of train $r \in R$ and $\underline{t}_v \in \mathbb{R}_+$ the earliest possible arrival time of r at v (i. e., the arrival time if r starts at $\hat{t}^r$ at its first station and does not wait at any station before v). Then the penalty of a run arc $e = ((u,t_u),(v,t_v)) \in A_T^r$, $u \ne v$, is the quadratic delay weighted with the relative length of the track segment compared with the length of the whole train run. Let $\ell_e^r = \bar{t}_{(u,v)}^r$ denote the length (in terms of running time) of track e and $\ell^r := \sum_{a\in A^r} \bar{t}_a^r$ denote the minimal running time for the whole run. We set the weight of arc $e = ((u,t_u),(v,t_v)) \in A_T^r$, $u \ne v$, to

$$w_e^r := -\ell_e^r/\ell^r \cdot (t_v - \underline{t}_v)^2,$$

and all other weights to 0 (note that we use negative weights because we want to have a maximization problem).

Putting all together, the TTP can be formulated as integer program as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{r\in R} \langle w^r, x^r \rangle \\ \text{subject to} \quad & x^r \in \mathcal{P}^r, \qquad r \in R, \\ & (2), (3), \end{aligned}$$

i. e., we select for each train r a feasible schedule $x^r \in \mathcal{P}^r$, so that all paths satisfy the headway and capacity constraints.

However, we do not use the headway inequalities on all arcs $a \in A^I$, but we use another approach to be presented in Section 3.2.

3.2 Configuration Networks

Our modeling of headway constraints is based on an extended formulation, which has been proposed in [2]. This formulation introduces additional *configuration networks* in order to model the safety distances between succeeding trains.

The construction is as follows: Let s_a be an artificial source and t_a an artificial sink node on track $a = (u, v) \in A^I$. The set

$$\tilde{A}^a_r := \{((u, t_u), (v, t_v)) \colon ((u, t_u), (v, t_v)) \in A^r_T, (u, v) = a\}$$

denotes all running arcs of train $r \in R$ on track a. These arcs correspond to some arcs of the configuration network. For each arc $e \in \tilde{A}^a_r$, $r \in R^a$, we introduce a pair of start and end nodes

$$B^a := \{(r, \text{start}, e) \colon r \in R^a, e \in \tilde{A}^a_r\}, \qquad \text{and,} \qquad E^a := \{(r, \text{end}, e) \colon r \in R^a, e \in \tilde{A}^a_r\}.$$

For these nodes we define the following sets of arcs:

1. The *start arcs* $\tilde{A}^a_{\text{start}} := \{(s_a, u) \colon u \in B^a\}$.
2. The *end arcs* $\tilde{A}^a_{\text{end}} := \{(u, t_a) \colon u \in E^a\}$.
3. The *wait arcs*

$$\tilde{A}^a_{\text{wait}} := \{((r, \text{start}, e), (r, \text{start}, e')) \in B^a \times B^a \colon e = ((u, t_u), (v, t_v)) \in \tilde{A}^a_r, e' = ((u, t_u + 1), (v, t_v + 1)) \in \tilde{A}^a_r\},$$

   which allow to have a larger distance between two succeeding trains than the minimal headway time.
4. The run arcs, each corresponding to a possible run of one train

$$\tilde{A}^a_{\text{run}} := \{((r, \text{start}, e), (r, \text{end}, e)) \in B^a \times E^a \colon r \in R^a\}.$$

5. The *headway arcs*

$$\tilde{A}^a_{\text{hw}} := \{((r, \text{end}, e), (r', \text{start}, e')) \in E^a \times B^a \colon r, r' \in R^a, r \neq r', e = ((u, t_u), (v, t_v)), e' = ((u, t'_u), (v, t'_v)), t'_u - t_u = h^a(r, r')\},$$

   which model that r runs immediately before r' while respecting the headway time.

Then the configuration network $\tilde{G}^a = (\tilde{V}^a, \tilde{A}^a)$, $a \in A^I$, is defined by

$$\tilde{V}^a := \{s_a, t_a\} \cup B^a \cup E^a,$$
$$\tilde{A}^a := \tilde{A}^a_{\text{start}} \cup \tilde{A}^a_{\text{end}} \cup \tilde{A}^a_{\text{run}} \cup \tilde{A}^a_{\text{hw}} \cup \tilde{A}^a_{\text{wait}}.$$

Figure 1 shows an example of a configuration network for two trains.

Given that the headway times are strictly positive and transitive, it is easy to see that valid configurations on track $a \in A^I$, i. e. selections of exactly one run for each train on that arc such that the headway restrictions are satisfied, correspond to s_a-t_a-paths that contain exactly one run arc for each train, see [16]. Therefore, we define the set of feasible configuration paths in $\tilde{G}^a$, $a \in A^I$, as

$$\tilde{\mathcal{P}}^a := \{P \subseteq \tilde{G}^a \colon P \text{ is an } s_a\text{-}t_a\text{-path in } \tilde{G}^a \text{ with exactly one run arc for each train}\}.$$

▶ Remark. Note that for single line tracks, only one configuration network is constructed for both arcs $(u, v) \in A^I$ and $(v, u) \in A^I$.

As with the train graphs we associate the binary variables $\tilde{x}^a_e \in \{0, 1\}$ with the arcs $\tilde{A}^a$. The coupling between configuration networks and train graphs is simple: a train may use one of its run arcs if and only if the corresponding run arc is contained in the configuration

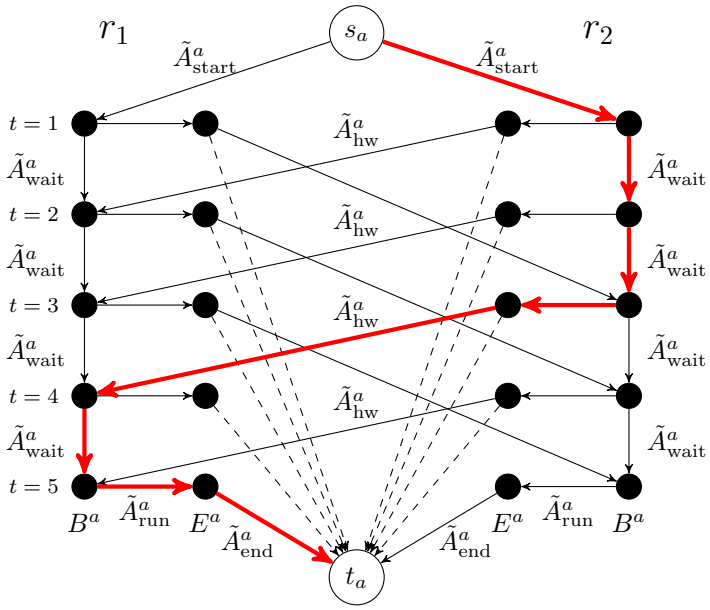


Figure 1 Example configuration network for two trains $\{r_1, r_2\} = R^a$ on some infrastructure arc $a \in A^I$. The headway times are $h^a(r_1, r_2) = 2$ and $h^a(r_2, r_1) = 1$. The red path corresponds to a configuration with r_2 running at $t = 3$ followed by r_1 at $t = 5$.

for this arc. Hence, denoting the run arc in a configuration network for some train arc $e = ((u, t_u), (v, t_v)) \in A^r_T$, $r \in R$, $(u, v) \in A^I$, by

$$\text{cfg}(e) := ((r, \text{start}, e), (r, \text{end}, e)) \in \tilde{A}^{(u,v)},$$

we have the following *configuration constraints*

$$x^r_e = \tilde{x}^{(u,v)}_{\text{cfg}(e)}, \qquad r \in R, e = ((u, t_u), (v, t_v)) \in A^r_T.$$

3.3 Ordering Constraints

The basic observation when using configuration networks is the following. The run of a train $r \in R$ on some specific infrastructure arc $a \in A^I$ is represented by the run arcs $((r, \text{start}, e), (r, \text{end}, e)) \in \tilde{A}^a_{\text{run}}$. However, we are interested in the headway arcs $((r, \text{end}, e), (r', \text{start}, e')) \in \tilde{A}^a_{\text{hw}}$. If one of these arcs equals 1, then this means that train r' is the direct successor of r on arc a. In particular, with

$$\tilde{A}^a_{\text{hw}}(r, r') := \{((r, \text{end}, e), (r', \text{start}, e')) \in \tilde{A}^a_{\text{hw}}\}$$

we define the *ordering variables*

$$s^a_{r,r'} = \sum_{a' \in \tilde{A}^a_{\text{hw}}(r,r')} \tilde{x}^a_{a'}, \qquad a \in A^I, r, r' \in R^a,$$

with the interpretation

$$s^a_{r,r'} := \begin{cases} 1, & r' \text{ succeeds } r \text{ directly on } a \in A^I, \\ 0, & \text{otherwise.} \end{cases}$$

One of the weaknesses of the standard time expanded formulation is that combinatorial properties of the network are not represented well. Figure 2 shows a typical situation: two

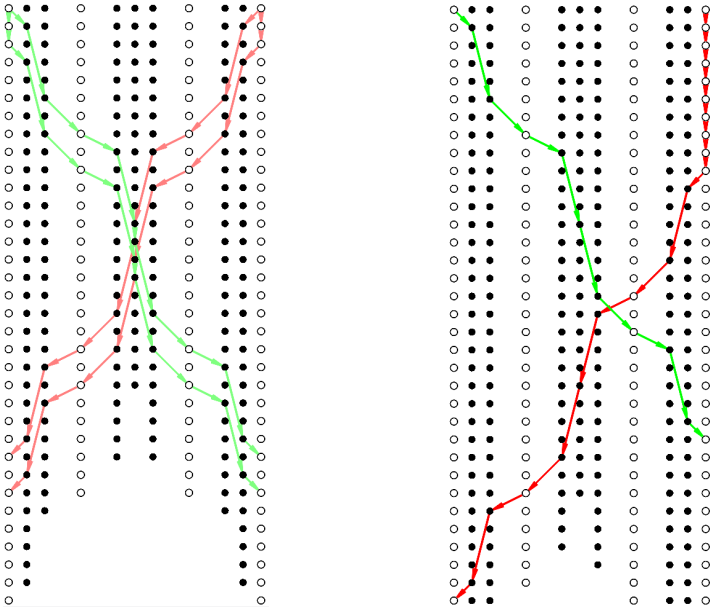


Figure 2 Tiny example with two trains running in opposite directions on a single line track with two passing possibilities. The white nodes have capacity 2 the other nodes have capacity 1. Nodes in the same row correspond to a sequence of stations at the same time step, nodes in the same column correspond to one station but at different time steps (time grows from top to bottom). All arcs between black nodes are single track, the arcs adjacent to white nodes are double track. The left picture shows an optimal solution for the standard linear relaxation. Note that the solution exploits the weak formulation allowing the two trains to meet and pass on a single line part. The right picture shows the optimal solution if ordering constraints are added to all consecutive paths of single line tracks: one train has to wait at a capacity 2 node for the other train to pass.

trains run on a sequence of single line tracks, such that no overtaking is allowed on the intermediate nodes. For instance, the intermediate nodes could be small local stations without overtaking/passing possibility or the arcs represent a sequence of blocking areas (guarded by signals) which must not be occupied by more than one train. In particular, stopping and waiting at these intermediate nodes is allowed. Obviously, because there is no overtaking possibility, in a feasible solution one train must go first through the complete sequence of tracks and then the other. However, the fractional solution can easily exploit the weak formulation: both trains can run fractionally in short succession and pass at an intermediate node, see Figure 2.

Ordering variables provide an easy way to formulate these kind of non-overtaking properties. Consider a path of nodes $(u_1, u_2, \dots, u_n)$ such that the capacity of each intermediate node is 1, i. e. $c_{u_i} = 1$, $i = 2, \dots, n-1$, and all arcs (u_i, u_{i+1}), $i = 1, \dots, n-1$, are single line tracks. Then it is clear that the order of trains running on arc (u_1, u_2) (or (u_2, u_1), which is the same because it is a single line track) must be equal to the order of trains running on arc (u_{n-1}, u_n). This can be enforced using the following *ordering constraints*

$$s_{r,r'}^{(u_1,u_2)} = s_{r,r'}^{(u_{n-1},u_n)}, \qquad r, r' \in R^{(u_1,u_2)} = R^{(u_{n-1},u_n)}.$$

Note that exactly the same constraints give rise to ordering conditions if the path consists of double line tracks. The only difference is that the configuration networks associated with (u_1, u_2) and (u_{n-1}, u_n) only model the headway conditions (and thus the ordering) of trains

running in the same direction. Thus there would be two sets of ordering constraints, one for (u_1, u_2) and (u_{n-1}, u_n), and one for (u_2, u_1) and (u_n, u_{n-1}).

▶ Remark. Of course, the ordering of the trains must not only be the same on the first and last arcs of the path but also on each intermediate arc, so one would have ordering constraints for all pairs of arcs $\{(u_i, u_{i+1}), (u_j, u_{j+1})\}$, $i, j = 1, \dots, n-1$, $i \neq j$. However, because of practical considerations (configuration networks can get large and are thus expensive from a computational point of view), we use them only on the first and last arcs of a path.

In our preliminary experiments we used only this simplest case of ordering constraints on paths, where no overtaking/passing is possible. This may sound oversimplified, but it is a typical situations in real world instances as well, where connections between different stations are made of such paths. However, it should be possible to extend this approach to situations, where the capacity of some intermediate station u_i, $i \in \{2, \dots, n-1\}$, is a small number larger than 1. For instance, if $c_{u_i} = 2$ on exactly one intermediate station, then the orders of trains on the first and last arcs may differ, but not arbitrarily. In particular, in this case the order of three trains may not be reversed (r_1 before r_2 before r_3 on the first arc but r_3 before r_2 before r_1 on the last arc). However, in order to express this kind of ordering restrictions one might need more complex configuration subproblems that do not only have variables for trains in direct succession but also for trains with an additional train in between. Solving such configuration problems (or a reasonable approximation of them) will probably be hard in itself and needs more work.

3.4 Complete Model

In this section we present the complete model. Let $\mathcal{O} \subseteq \binom{A^I}{2}$ be the set of pairs of infrastructure arcs, such that for $\{(u, u'), (v, v')\} \in \mathcal{O}$ there is a path $P = (u_1, \dots, u_n)$ of maximal length with $c_{u_i} = 1$ for all $i = 2, \dots, n-1$ such that

1. $(u, u') = (u_1, u_2)$ and $(v, v') = (u_{n-1}, u_n)$,
2. either all arcs are single line tracks or all arcs are double line tracks,
3. $R^{(u,u')} = R^{(u_i, u_{i+1})}$ for all $i = 1, \dots, n-1$.

Let $\tilde{A}^I := \bigcup_{X \in \mathcal{O}} X$ be the set of all arcs that are contained in at least one ordering constraint. Because configuration networks enlarge the model quite a bit, we use them only on those infrastructure arcs, where they are required to formulate some ordering constraints. On all other arcs $A^I \setminus \tilde{A}^I$ we use classical headway inequalities. The resulting model reads

$$
\begin{aligned}
\text{(TTP)} \qquad \text{maximize} \quad & \sum_{r \in R} \langle w^r, x^r \rangle, && && (4)\\
\text{subject to} \quad & x^r \in \mathcal{P}^r, && r \in R, && (5)\\
& \tilde{x}^a \in \tilde{\mathcal{P}}^a, && a \in \tilde{A}^I, && (6)\\
& x^r_a + x^{r'}_{a'} \leq 1, && a \in A^I \setminus \tilde{A}^I, \{(r,a),(r',a')\} \in H, && (7)\\
& x^r_e = \tilde{x}^{(u,v)}_{\mathrm{cfg}(e)}, && e = ((u,t_u),(v,t_v)) \in A^r, r \in R, && (8)\\
& \sum_{(r,a) \in K(u,t)} x^r_a \leq c_u, && u \in V^I, t \in T, && (9)\\
& s^a_{r,r'} = \sum_{a' \in \tilde{A}^a_{\mathrm{hw}}(r,r')} \tilde{x}^a_{a'}, && a \in \tilde{A}^I, r, r' \in R^a, && (10)\\
& s^a_{r,r'} = s^{a'}_{r,r'}, && r, r' \in R^a = R^{a'}, \{a, a'\} \in \mathcal{O}. && (11)
\end{aligned}
$$

We optimize a linear objective function (4) so that in each train graph and each configuration network a feasible path representing a schedule (5) or a configuration (6), respectively, is selected. The configuration networks and train graphs are coupled by (8), for infrastructure arcs $a \in \tilde{A}^I$ that have a configuration network. On the other arcs we use the usual headway inequalities (7). The capacity restrictions on the nodes are enforced by (9). Finally, constraints (10) introduce the ordering variables which are then coupled by the ordering constraints (11). Note that we write the constraints (10) only for the sake of presentation as they can easily be substituted in (11).

We use the objective function of [7], which is quite simple: all trains should run as fast as possible, so that any delays are minimized. In other words, the weights are so that early run arcs have higher weights than later arcs.

4 Solution Methods

In this section we briefly describe our solution method. The basic approach is to apply Lagrangian relaxation to (TTP), see, e. g., [3] for an early work using this approach. Indeed, we relax all coupling constraints (7)–(9) and (11) (with (10) being substituted in (11)). We collect all coupling equality and inequality constraints in

$$\sum_{r\in R} M_{1,r}x^r + \sum_{a\in\tilde{A}^I} M_{1,a}\tilde{x}^a = b_1 \qquad \text{and} \qquad \sum_{r\in R} M_{2,r}x^r + \sum_{a\in\tilde{A}^I} M_{2,a}\tilde{x}^a \le b_2,$$

respectively. The dual problem then reads

$$\begin{aligned} \text{(LR)} \qquad & \text{minimize} && \varphi(y,z), \\ & \text{subject to} && y \in \mathbb{R}^{m_1}, z \in \mathbb{R}^{m_2}_+, \end{aligned}$$

where the dual function $\varphi(y,z)$ is defined by

$$\varphi(y,z) := \sum_{r\in R} \max_{x^r\in\mathcal{P}^r} \langle w^r - M_{1,r}^T y - M_{2,r}^T z, x^r\rangle + \sum_{a\in\tilde{A}^I} \max_{\tilde{x}^a\in\tilde{\mathcal{P}}^a} \langle -M_{1,a}^T y - M_{2,a}^T z, \tilde{x}^a\rangle.$$

It is well-known that the dual problem (LR) is a non-smooth, convex optimization problem that can be solved by, e. g., bundle methods [13]. In particular, we use a special scaling variant of a bundle method based on ConicBundle [12]. Because there is a huge number of potential coupling constraints, they are separated during the solution process.

In order to solve (LR) one has to evaluate the function φ at certain trail points (y,z) provided by the bundle method. The subproblems in the train graphs G^r, $r \in R$, have the form

$$\max_{x^r\in\mathcal{P}^r} \langle w^r - M_{1,r}^T y - M_{2,r}^T z, x^r\rangle,$$

which are longest path problems in acyclic networks (which are equivalent to shortest path problems with the negated objective function because of the acyclicity). Because these networks get very large if the number of time steps increases, we use a dynamic graph generation algorithm proposed in [9], implemented in the DynG callable library [8].

The subproblems in the configuration networks $\tilde{G}^a$, $a \in \tilde{A}^I$, read similarly

$$\max_{\tilde{x}^a\in\tilde{\mathcal{P}}^a} \langle -M_{1,a}^T y - M_{2,a}^T z, \tilde{x}^a\rangle.$$

However, the set of feasible paths $\tilde{\mathcal{P}}^a$, $a \in \tilde{A}^I$, is more complicated in general than the sets $\mathcal{P}^r$, $r \in R$. The reason is that a feasible configuration corresponds to an s_a-t_a-path if and

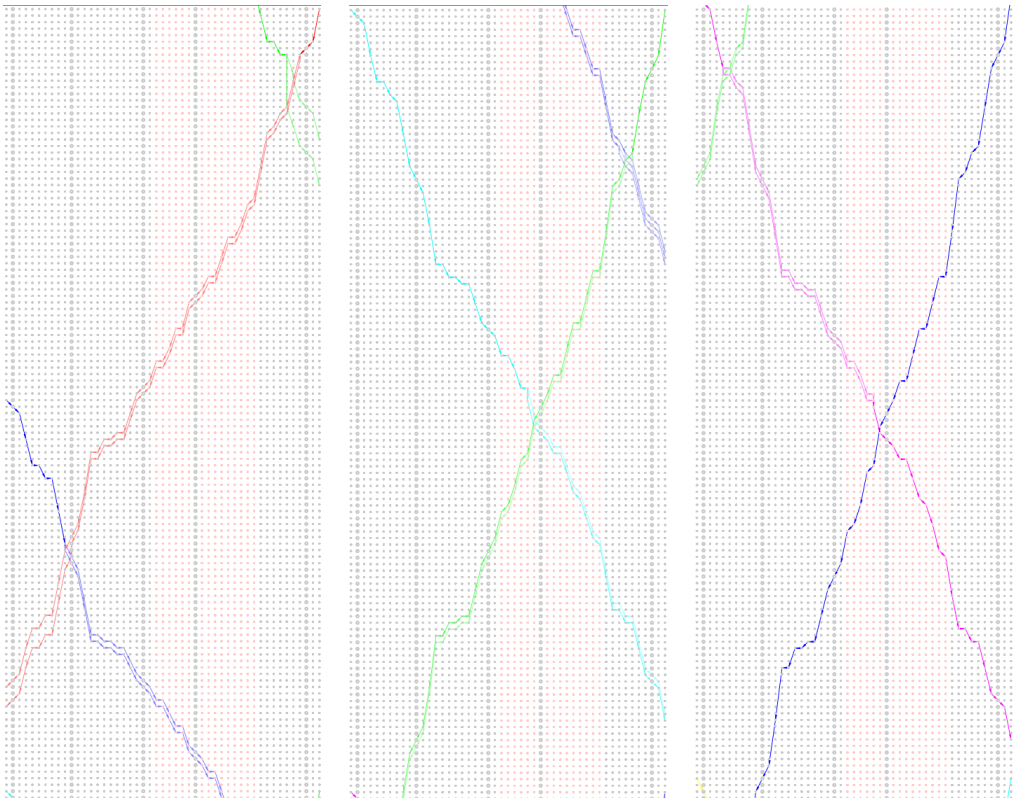


Figure 3 Part of the solution of (LR) for 12 trains *without ordering constraints* (consecutive part from left to right picture). The gray nodes form single line parts, the red nodes are a double line part. Only the thicker nodes have capacity 2, all others have capacity 1. Several passings on the single line parts are not resolved correctly because of the weak relaxation.

only if this path contains exactly one run arc for each train. These subproblems become very difficult to be solved exactly even for relatively small numbers of trains, say ten. Therefore we use the following relaxed version. Let

$$\tilde{\mathcal{P}}^a_{\text{rlx}} := \left\{ P \subseteq \tilde{G}^a : P \text{ is an } s_a\text{-}t_a \text{ containing at most } |R^a| \text{ run arcs of } \tilde{A}^a_{\text{run}} \right\} \supseteq \tilde{\mathcal{P}}^a.$$

This problem is simpler (it is a rather easy constrained shortest path problem), and we solve it using a dynamic programming algorithm, exploiting the fact that $\tilde{G}^a$ is acyclic. However, solving these subproblems only approximately can lead to worse solutions of the relaxation in practice, see Section 5.

5 Numerical Tests

We tested our approach on a small 12 train test instance from the RAS Problem Solving Competition 2012 [15]. The subproblems for the train graphs have been implemented with the DynG callable library [8], the subproblems in the configuration networks are solved approximately by a dynamic programming approach (see Section 3.4). The Lagrangian relaxation (LR) has been solved using a proximal bundle method based on ConicBundle [12]. All experiments are done on an Intel Core i7 @ 3.5 GHz with 12 GB RAM.

The test instance consists of a corridor with 49 nodes and arcs and has several single line parts and one double line part. There are only 4 passing points in the single line parts and

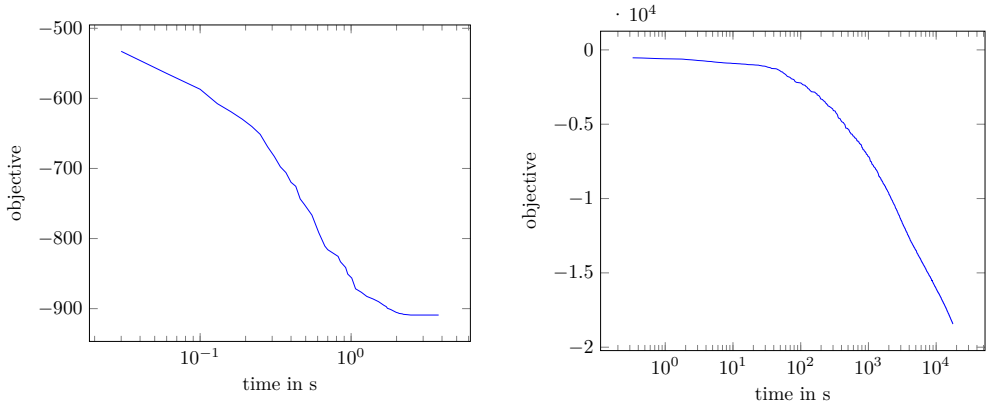


Figure 4 The left picture shows the dual function value after a certain amount of time in seconds for the model without ordering constraints. The right picture shows the same for the model with ordering constraints. Note that the time is given in a logarithmic scale. The model without ordering constraints converges very quickly, but the optimal value is quite bad. The model with ordering constraints converges much slower, but the objective value is much better. In fact, the dual bounds are better than without ordering constraints already after a few seconds.

one overtaking point in the double line part. The trains run over a period of about 9 hours, and each train requires between 1 and 2 hours to go from one end of the network to the other, depending on its speed. The model uses a time discretization of one minute.

First we look at the quality of the models in terms of the dual bounds. Figure 4 shows the development of the dual bounds after some computation time.

The pictures show that the model without ordering constraints converges very quickly to an optimal solution. In contrast, the model with ordering constraints converges very slowly but generates much better bounds. In fact, the bounds are better than the bounds without ordering constraints after 10 seconds. The slow convergence is a known disadvantage of configuration based models (also see [7], Chapter 6.4.3) when used in a Lagrangian relaxation approach with first-order optimization methods (like bundle methods).

However, in order to investigate the structure of the optimal solutions, we looked at the approximate solutions after a large computation time. A part of the resulting schedule is shown in Figure 3 for the model *without ordering constraints* and in Figure 5 for the model *with ordering constraints*.

The pictures show that without the ordering constraints the (fractional) solution of the relaxation easily exploits the weakness of the model and lets the trains meet and pass in the middle of single line parts hardly slowing down any train. In contrast, the model with the ordering constraints successfully finds appropriate waiting possibilities for some of the trains, so that meet and pass points are exactly at the nodes with capacity 2 or within the double line part.

However, the results are not always perfect. Figure 6 shows another part of the solution. Here the meet and pass decisions have not been resolved correctly. But the reason for this is not the inaccuracy of the ordering constraints. The problem is that the configuration subproblems are only solved approximately. When looking at the solutions of the subproblems during the algorithm, one sees that there are paths in $\tilde{G}^a$, $a \in \tilde{A}^I$, which do not correspond to correct configurations. In particular, the returned paths contain more than one run arc $((r, \text{start}, e), (r, \text{end}, e)) \in \tilde{A}^a_{\text{run}}$ for some train $r \in R^a$ and zero run arcs $((r', \text{start}, e'), (r', \text{end}, e')) \in \tilde{A}^a_{\text{run}}$ for some other train $r' \in R^a$ (see Section 4).

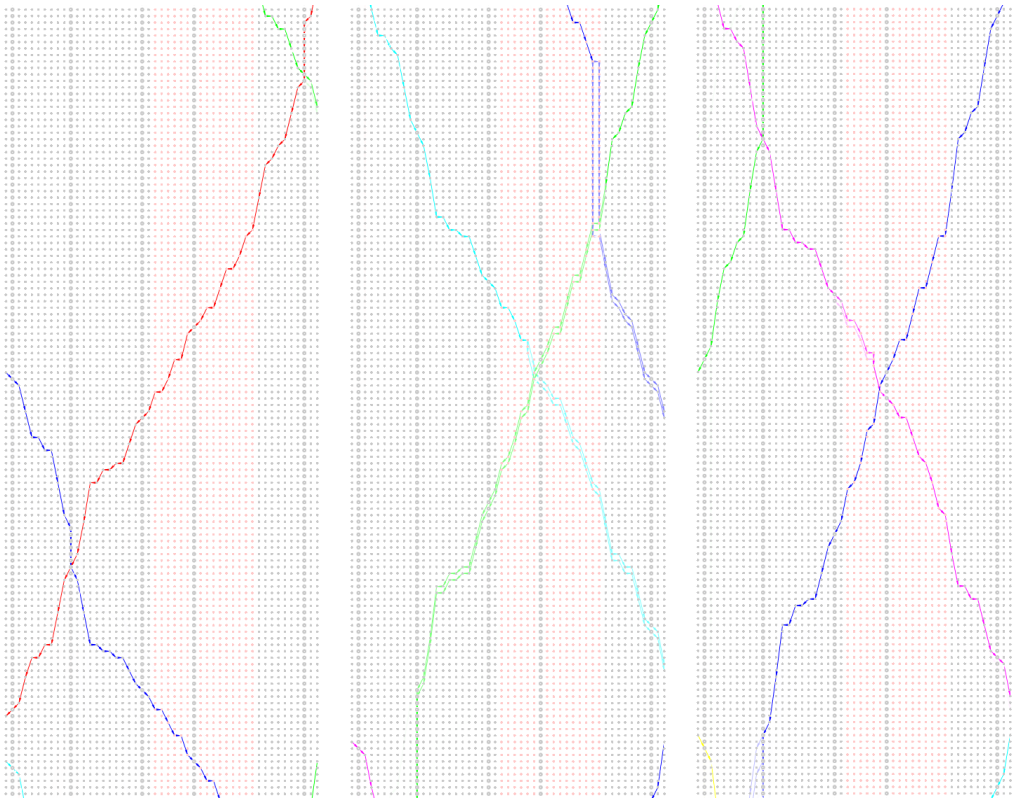


Figure 5 Part of the solution of (LR) for 12 trains *with ordering constraints* (consecutive part from left to right picture). The gray nodes form single line parts, the red nodes are a double line part. Only the thicker nodes have capacity 2, all others have capacity 1. The passings of the trains have been resolved correctly.

Note that we were not able to solve the relaxation with solving the configuration subproblems exactly. Although solving one subproblem exactly takes only few seconds, the solution method using a proximal bundle method requires many iterations and thus many subproblem evaluations for all arcs in $\tilde{A}^I$, hence the solution process did not have sufficient progress in reasonable time.

▶ Remark. Indeed, it turned out that the solution of the configuration subproblems, even if we solve them only approximately, were the main bottleneck in our approach. In contrast, the solution of the train graph subproblems was extremely fast thanks to the used dynamic graph generation technique, which ensures that only very small parts of the train graphs have to be stored and that the solutions can be found very quickly.

6 Conclusions and Future Work

In this paper we proposed an extension of a configuration network based formulation for the TTP. In particular, we use the configuration networks to formulate constraints that forbid changes in the order of trains along a path of tracks that are not possible due to the existing overtaking possibilities of the infrastructure network. We implemented the model and demonstrated how the new model greatly improves the solution of the Lagrangian relaxation of the model. Indeed, in the example the relaxation resolves many meet and pass

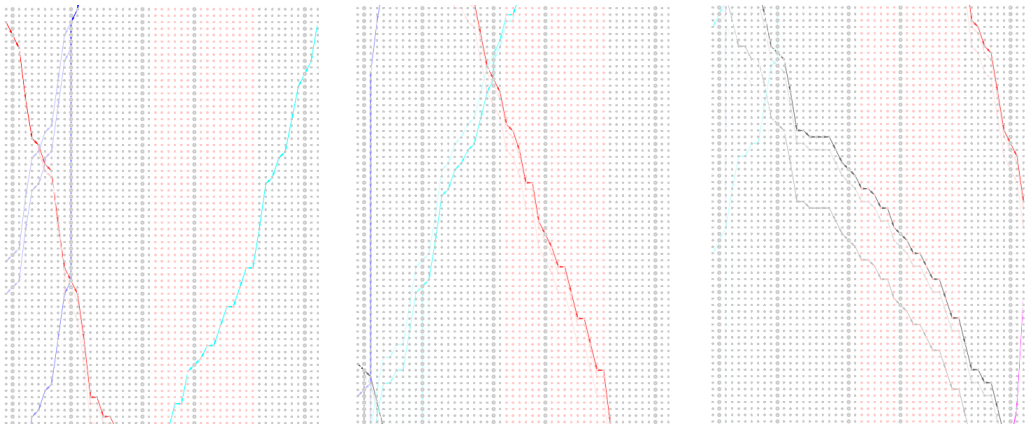


Figure 6 Another part of the solution of (LR) with ordering constraints. Because the configuration subproblems are only solved approximately, the resulting fractional solution may still contain wrong passing decisions (in particular for the blue train in this example).

decisions correctly, which does not happen without the ordering constraints. However, we have also seen that the ordering constraints might not be sufficient to ensure correct meet and pass decisions if the configuration subproblems are not solved exactly.

We can conclude from our experiments that the proposed extension with ordering constraints has great potential to improve existing models. From a theoretical point of view, several improvements of our model are possible. For instance, one could formulate configuration subproblems that represent not only the ordering of trains in direct succession but also of larger groups of trains. This would allow to formulate ordering conditions if a certain sequence of infrastructure arcs has not zero but a small number of overtaking points. Another possible improvement could be to solve the configuration subproblems exactly for small subsets of all trains running on some infrastructure arc. For instance, one could formulate configuration subproblems that ensure that each subset of 3 to 5 trains runs correctly. Given that at most points in time only few trains compete for some infrastructure arc, the exact solution of these small subproblems could be possible from a computational point of view and improve the quality of the relaxation.

Finally, our experiments shows that the current approach using standard first-order methods to solve these problems has a very bad convergence behavior. Hence, although better bounds than before can be reached after short computation times, near optimal solutions still take very long. Therefore several algorithmic developments or alternative solution methods are required in order to improve the efficiency of our approach so that it can be applied to large real world instances.

References

1 Ralf Borndörfer, Andreas Löbel, Markus Reuther, Thomas Schlechte, and Steffen Weider. Rapid branching. *Public Transport*, 5(1-2):1–21, 2013. doi:10.1007/s12469-013-0066-8.

2 Ralf Borndörfer and Thomas Schlechte. Models for railway track allocation. In Christian Liebchen, Ravindra K. Ahuja, and Juan A. Mesa, editors, *ATMOS 2007 - 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007. doi:10.4230/OASIcs.ATMOS.2007.1170.

3 U. Brännlund, P. O. Lindberg, A. Nõu, and J. E. Nilsson. Railway timetabling using Lagrangian relaxation. *Transportation Science*, 32(4):358–369, 1998.

4 Valentina Cacchiani, Fabio Furini, and Martin Philip Kidd. Approaches to a real-world train timetabling problem in a railway node. *Omega*, 58:97–110, 2016. doi:10.1016/j.omega.2015.04.006.

5 Valentina Cacchiani and Paolo 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.

6 Alberto Caprara, Matteo Fischetti, and Paolo Toth. Modeling and solving the train timetabling problem. *Operations Research*, 50(5):851–861, 2002.

7 Frank Fischer. *Dynamic Graph Generation and an Asynchronous Parallel Bundle Method Motivated by Train Timetabling*. PhD thesis, Chemnitz University of Technology, 2013. URL: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-118358.

8 Frank Fischer. *DynG Dynamic Graph Generation library*, 2014. URL: http://www.mathematik.uni-kassel.de/~fifr/fossils/dyng.

9 Frank Fischer and Christoph Helmberg. Dynamic graph generation for the shortest path problem in time expanded networks. *Mathematical Programming A*, 143(1-2):257–297, 2014. doi:10.1007/s10107-012-0610-3.

10 Frank Fischer and Christoph Helmberg. A parallel bundle framework for asynchronous subspace optimization of nonsmooth convex functions. *SIAM Journal on Optimization*, 24(2):795–822, 2014. doi:10.1137/120865987.

11 Frank Fischer, Christoph Helmberg, Jürgen Janßen, and Boris Krostitz. Towards solving very large scale train timetabling problems by Lagrangian relaxation. In Matteo Fischetti and Peter Widmayer, editors, *ATMOS 2008 - 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. doi:10.4230/OASIcs.ATMOS.2008.1585.

12 Christoph Helmberg. *ConicBundle 0.3.11*. Fakultät für Mathematik, Technische Universität Chemnitz, 2012. URL: http://www.tu-chemnitz.de/~helmberg/ConicBundle.

13 Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex Analysis and Minimization Algorithms I & II*, volume 305, 306 of *Grundlehren der mathematischen Wissenschaften*. Springer, Berlin, Heidelberg, 1993.

14 Richard M. Lusby, Jesper Larsen, Matthias Ehrgott, and David Ryan. Railway track allocation: models and methods. *OR Spectrum*, 33(4):843–883, oct 2011. doi:10.1007/s00291-009-0189-0.

15 RAS Problem Solving Competition 2012, 2012. URL: https://www.informs.org/Community/RAS/Problem-Solving-Competition/2012-RAS-Problem-Solving-Competition.

16 Thomas Schlechte. *Railway Track Allocation: Models and Algorithms*. PhD thesis, TU Berlin, 2012.

17 Steffen Weider. *Integration of Vehicle and Duty Scheduling in Public Transport*. PhD thesis, TU Berlin, 2007. URL: http://opus.kobv.de/tuberlin/volltexte/2007/1624/.

Regional Search for the Resource Constrained Assignment Problem

Ralf Borndörfer and Markus Reuther

Zuse Institute Berlin
Takustrasse 7, 14195 Berlin, Germany
⟨*surname*⟩@zib.de

Abstract

The resource constrained assignment problem (RCAP) is to find a minimal cost partition of the nodes of a directed graph into cycles such that a *resource constraint* is fulfilled. The RCAP has its roots in rolling stock rotation optimization where a railway timetable has to be covered by rotations, i.e., cycles. In that context, the resource constraint corresponds to maintenance constraints for rail vehicles. Moreover, the RCAP generalizes variants of the vehicle routing problem (VRP). The paper contributes an exact branch and bound algorithm for the RCAP and, primarily, a straightforward algorithmic concept that we call *regional search* (RS). As a symbiosis of a local and a global search algorithm, the result of an RS is a local optimum for a combinatorial optimization problem. In addition, the local optimum must be globally optimal as well if an instance of a problem relaxation is computed. In order to present the idea for a standardized setup we introduce an RS for binary programs. But the proper contribution of the paper is an RS that turns the Hungarian method into a powerful heuristic for the resource constrained assignment problem by utilizing the exact branch and bound. We present computational results for RCAP instances from an industrial cooperation with Deutsche Bahn Fernverkehr AG as well as for VRP instances from the literature. The results show that our RS provides a solution quality of 1.4 % average gap w.r.t. the best known solutions of a large test set. In addition, our branch and bound algorithm can solve many RCAP instances to proven optimality, e.g., almost all asymmetric traveling salesman and capacitated vehicle routing problems that we consider.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases assignment problem, local search, branch and bound, rolling stock rotation problem, vehicle routing problem

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.111

1 Introduction

Let $D = (V, A)$ be a directed graph with dedicated *events* taking place at every arc. We distinguish *replenishment events* from other events and call arcs with replenishment events *replenishment arcs*. Let $r : A \mapsto \mathbb{Q}_+ \times \mathbb{Q}_+$ be a *resource function* that assigns a pair of nonnegative rational numbers (r_a^1, r_a^2) to every arc denoting a resource consumption before and after the event, respectively, and define $r_a := r_a^1 + r_a^2$. A *resource path* is an elementary path in D of the form $P = (a_0, a_1, ..., a_m, a_{m+1}) \subseteq A$ such that a_0 and a_{m+1} are replenishment arcs and $a_1, ..., a_m$ are not replenishment arcs. Let $\mathbb{P}(A)$ be the set of all resource paths and $B \in \mathbb{Q}_+$ be a *resource bound*. We call a resource path $P = (a_0, a_1, ..., a_m, a_{m+1}) \in \mathbb{P}(A)$ *feasible* if the following *resource constraint* is fulfilled (otherwise P is *infeasible*):

$$r_{a_0}^2 + \sum_{i=1}^{m} r_{a_i} + r_{a_{m+1}}^1 \leq B. \tag{1}$$

Finally, let $c : A \mapsto \mathbb{Q}$ be some objective function associated with the arcs of D.

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt; pp. 111–129

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Definition 1** (Resource Constrained Assignment Problem (RCAP)). Given a directed graph $D = (V, A)$, a resource function r, an objective function c, and a resource bound B. The RCAP is to find a set of directed cycles $C_1, \ldots, C_n \subseteq A$ in D such that every node is contained in exactly one cycle, every cycle contains at least one replenishment arc, all resource paths in $\mathbb{P}(\bigcup_{i=1}^{n} C_i)$ are feasible, and $c\,(\bigcup_{i=1}^{n} C_i)$ is minimal.

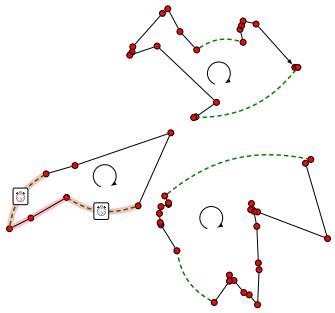


Figure 1 Cycle partition.

Figure 1 illustrates the RCAP by showing a set of nodes covered by three cycles. The dashed arcs are replenishment arcs. A resource path fulfilling constraint (1) is highlighted in red where the two stop watches indicate replenishment events. In our previous paper [16], we additionally defined all cycles $C \subseteq A$ with $\sum_{a \in A} r_a = 0$ to be feasible. In order to streamline the presentation we assume that D does not contain such cycles in this paper. We also assume that D does not contain multiple arcs between two nodes. We remark that the treatment of replenishment events "in the middle of the arcs" could be replaced by a consideration of replenishment nodes. This would blow up the RCAP instances that we are interested in. In addition, the use of replenishment nodes is no more possible if multiple resource constraint are considered which we like to keep open.

The RCAP has its roots in the *rolling stock rotation problem* (RSRP) [17], i.e., the RCAP is a specialization of the RSRP. In the RSRP the resource constraint models a maintenance constraint for rail vehicles, e.g., refueling. To model time or distance consumptions directly before or after replenishment events at the arc $a \in A$ one can use the pair (r_a^1, r_a^2). Moreover, the RCAP generalizes variants of the vehicle routing problem (VRP), see Section 3.5. In this way the RCAP provides access to different recent and classical problems.

The RCAP is a multifaceted combinatorial optimization problem in the sense that the variability in computational effort needed to solve an instance to proven optimality is huge. On the one hand, a small instance can be computational hard to solve, e.g., capacitated vehicle routing problems. On the other hand, large problem instances in which the resource constraint is less restrictive might be solved with little computational effort. We aim at utilizing this characteristic for our algorithmic design. The idea is that the algorithm should automatically allot less computation time to easy instances and more computation time to hard ones. We call this behavior *self-calibration*. Note that this desirable property is not evident for local search algorithms or meta-heuristics in general.

Our idea to implement this design is referred to as *regional search* (RS). It works as follows. Let P be a combinatorial optimization problem and let P' be a relaxation of P. Consider a feasible solution S for P, interpret S as a solution S' for P' for the moment, and consider a *local search* algorithm A' that *exactly solves* P'. In order to turn A' into an algorithm A that searches for improvements of S we "lift" the neighborhoods that are roamed by A' for S' back to the original problem P. In other words, the relaxation induces a neighborhood w.r.t. S. The lifted neighborhoods are called *regions* in order to highlight that they are exact for P', i.e., A is automatically exact if an instance of P' is considered. This algorithmic behavior is our characterization of an RS:

▶ **Definition 2** (Regional search). Let P be a combinatorial optimization problem and let A be a primal heuristic algorithm for P. Further, let P' be a relaxation of P. The algorithm A is a regional search if A is proven exact for any instance of P'.

In this way, the computational effort of A is related to the difference in tractability between P and P', i.e., A can be expected to be self-calibrating.

We proceed in Section 2 with an RS for *binary programs by using the simplex method* in order to argue that our idea is general enough to be directly used in other applications. Afterwards we present a specialized RS for the *RCAP by using the Hungarian method*. In Section 3 we describe a global search, namely a branch and bound procedure, for the RCAP. This algorithm is used as sub-routine in our RS as well as standalone exact method for the RCAP. In the last section we present computations for both the regional and global search.

2 Regional Search

In order to present our idea for a standardized setting we provide an RS for binary linear programs by using the simplex algorithm in this section. Afterwards, our proper RS for the RCAP is presented. In that algorithm a constraint integer program (CIP) for the RCAP (that we solve with a branch and bound procedure, see Section 3) and the Hungarian method take over the roles of the binary program and the simplex algorithm, respectively. In this way, we argue that the main algorithmic ingredients of our RS approach are at hand if one comes up with an (insufficient, i.e., not fast enough) exact algorithm and a linear programming relaxation for an optimization problem.

2.1 Regional search for binary programs by using the simplex algorithm

Given a rational matrix A and vectors b and c of suitable dimensions, we consider a binary program BP as

$$\min\{c^T x \,|\, Ax = b,\ x \text{ binary}\} \quad \text{with its linear relaxation} \quad \min\{c^T x \,|\, Ax = b,\ 0 \le x \le 1\}$$

that we call LP. Our RS for BP assumes that a feasible starting solution $x^\star$ is at hand, i.e., all values of $x^\star$ are binary and $Ax^\star = b$. We now interpret $x^\star$ as a basic solution of LP and try to improve $x^\star$ by using the well known primal simplex algorithm. The primal simplex algorithm iteratively improves a basic incumbent solution by searching through the *simplex neighborhood*. The simplex neighborhood of a basic solution $x^\star$ of LP is defined as the set of all basic solutions of LP that share an edge with $x^\star$ in the polytope associated with LP. We denote $\tilde{x} \sim x^\star$ if the basic solutions $\tilde{x}$ and $x^\star$ of LP share such an edge.

We now perform an improvement step of the primal simplex algorithm and end up with another basic solution $\tilde{x}$ for LP with $\tilde{x} \sim x^\star$ and $c^T\tilde{x} < c^T x^\star$ (assuming a non-degenerated simplex operation). In general, $\tilde{x}$ will not be binary, i.e., feasible for BP. In order to improve the chances to reach an improving binary vector we "lift" the simplex neighborhood as follows. If $\tilde{x} \sim x^\star$ and $c^T\tilde{x} < c^T x^\star$ we solve

$$\min\{c^T x \,|\, Ax = b,\ x \text{ binary},\ x_j = 1 \,\forall \text{ column indices } j \,:\, x^\star_j = \tilde{x}_j = 1\} \qquad (BP_{\text{REGION}})$$

Program (BP_{REGION}) is to solve BP under the additional constraint that all variables that agree to be one in both solutions of $\tilde{x} \sim x^\star$ are fixed. Note that $x^\star$ is always a feasible solution to program (BP_{REGION}) and $\tilde{x}$ is always a feasible solution to the linear relaxation of program (BP_{REGION}). The motivation behind this setup is to gain a computational compromise between the goals (1) improvement of the objective function value while (2) preserving feasibility and (3) solving small sub-problems in order to be fast. Goal (1) is promised by the simplex algorithm through $c^T\tilde{x} < c^T x^\star$ and goal (2) is meet by solving a restricted version of the original problem BP in which the current incumbent solution is

always feasible. Goal (3) is achieved if the difference of successive basic solutions within the simplex algorithm is small. In this case, a large number of variables that agree to be one lead to a huge simplification of program (BP_{REGION}) compared to the original problem.

We suggest to solve program (BP_{REGION}) whenever $\tilde{x} \sim x^\star$ and $c^T\tilde{x} < c^T x^\star$. Thus, we investigate all solutions that simplex algorithm would investigate which shows that the above algorithm is an RS for binary programs according to Definition 2. It will always exactly solve binary programs for which the linear relaxation has an integral optimal solution. In this way every global search algorithm, i.e., exact algorithm, for problems that have a linear relaxation is an RS, but not every local search algorithm is regional. Note that the proposed algorithm can also be see as an iterated variable neighborhood search algorithm, see [4] for a recent overview in the context of mixed integer non-linear programming.

2.2 Regional search for the RCAP using the Hungarian method

Denoting by $x_a \in \{0,1\}$ a variable that is equal to one if $a \in A$ belongs to a solution and zero otherwise, and using the constraint notation of Achterberg [1, Example 3.2]), the RCAP can be formulated as a CIP that serves as basis for our approach:

$$\begin{aligned} &\min \textstyle\sum_{a\in A} c_a x_a \\ &\text{s.t.} \quad \textstyle\sum_{a\in\delta^+(v)} x_a = 1, \quad \forall\, v \in V \\ &\qquad \textstyle\sum_{a\in\delta^-(v)} x_a = 1, \quad \forall\, v \in V \\ &\qquad \text{RESOURCE CONSTRAINT}(x) \\ &\qquad x_a \in \{0,1\}, \quad \forall\, a \in A \qquad \text{where} \end{aligned} \tag{RCAP_{CIP}}$$

$$\text{RESOURCE CONSTRAINT}(x) \;\Leftrightarrow\; \nexists P \in \mathbb{P}(\text{supp}(x)) \,:\, P \text{ is an infeasible path.}$$

By deleting the RESOURCE CONSTRAINT from program (RCAP_{CIP}) we obtain the *assignment relaxation* (AP): For every node $v \in V$ there must be exactly on integral incoming and outgoing arc variable which forces $x \in \mathbb{Q}^{|A|}$ to define a cycle partition of the nodes of D. The assignment relaxation is the linear programming relaxation that we use for our RS. Let π_v^t and π_v^h be two free dual variables for each node $v \in V$. The assignment problem, i.e., the assignment relaxation of the RCAP, is to solve the following dual linear programs:

$$\begin{aligned} \text{(AP)}\ \min & \textstyle\sum_{a\in A} c_a x_a \\ \text{s.t.} & \textstyle\sum_{a\in\delta^+(v)} x_a = 1, \quad \forall v \in V \\ & \textstyle\sum_{a\in\delta^-(v)} x_a = 1, \quad \forall v \in V \\ & x_a \geq 0, \quad \forall a \in A \end{aligned} \qquad \begin{aligned} \text{(AD)}\quad \max & \textstyle\sum_{v\in V} \pi_v^t + \sum_{v\in V} \pi_v^h \\ \text{s.t.} & \pi_u^t + \pi_v^h \leq c_a, \quad \forall a = (u,v) \in A \\ & \pi_v^t \in \mathbb{Q}, \quad \forall v \in V \\ & \pi_v^h \in \mathbb{Q}, \quad \forall v \in V. \end{aligned}$$

In each basic solution of (AP) the x-variables are all binary and thus the integrality constraints for them can be relaxed if one solves program (AP) with a simplex method. We do not use a simplex method for (AP) and (AD) since it needs much effort to be implemented efficiently, in particular for our purposes. Instead we use a more specialized combinatorial algorithm, namely a primal version of the Hungarian method that we briefly summarize in the following.

Let $d_a := c_a - \pi_u^t - \pi_v^h$ be the *reduced cost* of the arc $a = (u,v) \in A$. By the strong duality theorem the x- and π-variables have optimal value if and only if they are feasible for (AP) and (AD) and the reduced cost or the x-variable is zero for each arc:

$$x_a \cdot d_a = 0, \quad \forall a \in A. \tag{2}$$

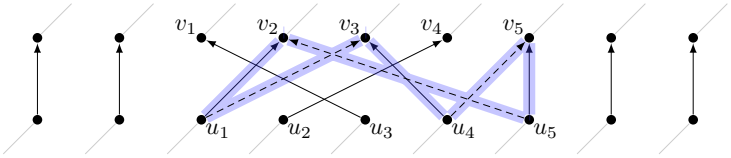


Figure 2 Alternating cycle $C = \{a_1^+ = (u_1, v_2),\ a_1^- = (u_5, v_2),\ a_2^+ = (u_5, v_5),\ a_2^- = (u_4, v_5),\ a_3^+ = (u_4, v_3),\ a_3^- = (u_1, v_3)\}$.

The primal Hungarian method of Balinski and Gomory [2] can be summarized as follows. Start with a feasible solution for (AP), i.e., a cycle partition in D and choose a configuration of the π-variables that need not be feasible for (AD) but have to satisfy (2). In each iteration of the primal Hungarian method either the cycle partition or the dual solution is improved. Thereby (2) is always preserved and the process stops if all arcs have positive reduced cost, i.e., the π-variables provide dual feasibility. The improvements found by the primal Hungarian method have a dedicated structure. In fact, they form *alternating cycles*. An alternating cycle alternates between (old) arcs that belong to the current incumbent cycle partition and (new) arcs that do not. By replacing the old arcs with the new arcs a new cycle partition appears. Figure 2 provides an example of an alternating cycle that deletes the arcs a_i^- and adds the arcs a_i^+ for $i = 1, 2, 3$. We refer to our previous paper [16] for more details about the primal Hungarian method in particular for the purpose of generating alternating cycles to be used as improvement operations. Moreover, we use exactly the same procedures to find improving alternating cycles in this paper as described in our previous paper [16].

Note that alternating cycles would also appear if we use the primal simplex algorithm because it follows exactly the same duality arguments and the symmetric difference of two vertices $\tilde{x}$ and $x^\star$ of the assignment polytope with $\tilde{x} \sim x^\star$ is exactly an alternating cycle, see [3].

Let $x^\star \in \{0,1\}^A$ be the current incumbent solution to program (RCAP_{CIP}) that is associated with the feasible cycle partition $M \subseteq A$. Further, let $\tilde{x} \in \{0,1\}^A$ be that one cycle partition that we obtain if we apply an alternating cycle $C = \{a_1^+, a_1^-, \ldots, a_n^+, a_n^-\}$ found by the primal Hungarian (or simplex) method to M. Analogous to the considerations for binary programs above it is very unlike that $\tilde{x}$ is feasible again since we did not spend any attention to the resource constraint so far. To this end, we "lift" the direct application of the alternating cycle C to the cycle partition M to the solution of the following *alternating cycle region* ($\text{RCAP}_{\text{REGION}}$):

$$
\begin{aligned}
&\min \textstyle\sum_{a \in A} c_a x_a \\
&\text{s.t.} \quad \textstyle\sum_{a \in \delta^+(v)} x_a \;\; = 1, \qquad \forall\, v \in V \\
&\qquad\;\; \textstyle\sum_{a \in \delta^-(v)} x_a \;\; = 1, \qquad \forall\, v \in V \\
&\qquad\qquad \text{RESOURCE CONSTRAINT}(x) \\
&\qquad\;\; x_a \;\; = 1 \qquad \forall\, a \in M \setminus \{a_1^-, \ldots, a_n^-\}, \\
&\qquad\;\; x_a \;\; \in \{0,1\}, \quad \forall\, a \in A.
\end{aligned}
\tag{$\text{RCAP}_{\text{REGION}}$}
$$

Solving this program increases the chances of finding an improved cycle partition under a resource constraint. An evident interpretation of solving program ($\text{RCAP}_{\text{REGION}}$) is that the primal Hungarian method suggest to apply the cycle C in order to improve the value of the objective function. But this is too naive. In order to compensate the resource constraint,

```
1  boolean isRegionallyOptimal( M ) // M is a cycle partition
2  {
3    for( a* ∈ {a ∈ A | d_a < 0} ) // pricing loop
4    {
5      C = findAlternatingCycle( a* ); // see [16]
6
7      if( C ≠ ∅ )
8      {
9        compute optimal solution M_R of (RCAP_REGION) for M and C;
10
11       if( c(M) > c(M_R) ) { return false; }
12     }
13   }
14   return true;
15 }
```

Algorithm 1 Proof of regional optimality.

we only take the arcs that the cycle proposes to delete seriously. Note that this is exactly what we describe for binary programs above, i.e., we fix all arc variables that agree to be one before and after the application of the alternating cycle. Program ($\text{RCAP}_{\text{REGION}}$) can be easily turned into a plain RCAP by replacing all constant arc variables associated with arcs of $\bigcup_{a=(u,v)\in M\setminus\{a_1^-,\ldots,a_n^-\}} \delta^+(u) \setminus \{a\}$. We solve program ($\text{RCAP}_{\text{REGION}}$) by the branch and bound algorithm presented in Section 3.

We are now ready to state Algorithm 1 that "proves regional optimality" for an instance of the RCAP. Our overall RS iteratively calls Algorithm 1 and replaces M by M_R if an improvement has been found until "regional optimality is proven". Obviously, this method is of type RS because it investigates at least all solutions, i.e., all solutions that can be reached by improving alternating cycles, that the primal Hungarian method would consider.

It turns out that it is computationally too short-sighted to always search for an optimal solution of ($\text{RCAP}_{\text{REGION}}$) because it rarely happens that the arising problem is almost as hard as the original instance if the alternating cycle is large. We resolve this issue by setting a limit of 10^3 branching nodes during depth-first-search [1] for model ($\text{RCAP}_{\text{REGION}}$).

The following insight provides the connection to our previous paper [16] that presents a local search algorithm for the RCAP.

▶ **Lemma 3.** *The algorithm proposed in our previous paper [16] is of type regional search.*

Proof. The main difference of the algorithm in [16] to the original version of the primal Hungarian method is that alternating cycles are *decomposed and recombined* before they are applied. Let $C = \{a_1^+, a_1^-, \ldots, a_n^+, a_n^-\} \subseteq A$ be the alternating cycle found. A *flip* is a 2-OPT move that is well-defined by an entering arc a_i^+, see [16]. The flips imposed by C can be applied in any sequence. Consider the cycle partition that results from any $n-1$ flips: It is exactly the same assignment that is defined by directly applying C. This is true, because after $n-1$ flips the matching clearly contains $n-1$ of the entering a_i^+ arcs and each flip inserts a closing arc that is deleted by another (because C is an alternating cycle). Thus, the matching must contain also the n-th of the a_i^+ arcs. This proofs the lemma, because one can not lose any alternating cycle, i.e., any improvement proposed by the Hungarian method. ◀

We close this section with the observation that our previous RS algorithm [16] is almost equal to our present RS with the important difference that we now exactly solve program ($\text{RCAP}_{\text{REGION}}$). This program is tackled heuristically in [16].

3 Branch and Bound for the RCAP

We present a branch and bound algorithm for the RCAP that is based on the constraint integer program (RCAP_{CIP}) already presented in Section 2.

An alternative formulation for the RCAP in terms of a pure integer program (IP) can be derived by replacing the RESOURCE CONSTRAINT in model (RCAP_{CIP}) with the infeasible path constraints

$$\sum_{a \in P} x_a \leq |P| - 1, \, \forall \, \text{infeasible paths } P \in \mathbb{P}(A). \tag{3}$$

We do, however, not expect that this integer program will produce useful results. Indeed, a vast number of papers – the most successful by now is [13] – consider much stronger formulations for the exact solution of the CVRP and the TSP, see the excellent and recent survey by Toth & Vigo [19]. In this paper we do not aim to generalize or adopt those approaches to the RCAP, even if this is an interesting research area. Instead, we pursue a much simpler approach that can solve lightly constrained easy problems fast, namely a branch and bound algorithm that does not generate any primal or dual cutting planes. We refer to [7, 19] for similar algorithms developed for the VRP.

This algorithm is based on formulation (RCAP_{CIP}) and the assignment relaxation RCAP′ for bounding. In each node, called *sub-problem*, of the branching tree the following steps are performed:

- solve the assignment relaxation of the current RCAP
- eliminate arcs using the assignment reduction, see Section 3.2
- eliminate arcs using the shortest path reduction, see Section 3.3
- eliminate arcs using the bin-packing reduction, see Section 3.4
- discard current branching node if
  - the optimal objective value of the node relaxation is not below the upper bound
  - the optimal solution of the the assignment relaxation is feasible
  - there are no further branching candidates, see Section 3.5.

In each *reduction* procedure we try to find detachable arcs of the current sub-problem that fulfill the following criterion: Any solution to the current sub-problem containing a detachable arc is definitely not better than the incumbent solution. If a reduction procedure detects an arc $a \in A$ fulfilling this criterion, we *detach* the arc from the current sub-problem, i.e., we delete the arc from the arc set A. Note that a detached arc remains detached in all child nodes of the branching tree. In the following sections, we explain our branching scheme and the three reduction procedures. We do not use a special notation to distinguish sub-problems from the original RCAP. Instead, we consider each branching node as a new RCAP instance.

3.1 Branching Scheme

Our algorithm uses the assignment relaxation of the RCAP to solve the subproblems in the branching tree. Thus, the solution of the current node relaxation is always integral. In fact, it is composed of a set of cycles $C_1, \ldots, C_k \subseteq A$. If all cycles contain at least one replenishment arc and all resource paths of $\mathbb{P}(\bigcup_{i=1}^{k} C_i)$ are feasible, we do not have to perform further branching. Otherwise, we branch on arc variables, i.e., for each branching candidate $a = (u, v) \in A$ we create two new sub-problems. The first arises from forcing $x_a = 1$ and in the other one the constraint $x_a = 0$ is imposed. The latter case is handled by detaching $a \in A$ from the current sub-problem, while the former is handled by detaching all arcs of $\delta^+(u) \setminus \{a\}$.

The following two situations lead to further branching on a certain sub-problem:

- a cycle, called *infeasible cycle*, of $\{C_1, \ldots, C_k\}$ does not contain a replenishment arc
- a path of $\mathbb{P}\left(\bigcup_{i=1}^{k} C_i\right)$ is infeasible.

Let $I = \{I_1, \ldots, I_m\}$ with $I_i \subseteq A$ for $i = 1, \ldots, m$ be the family of cycles and paths fulfilling one of these two criteria. In general it is valid to branch on each arc $a \in A$ of the current sub-problem, but it is natural to only branch on arcs $a \in \bigcup_{i=1}^{m} I_i$.

The set $\bigcup_{i=1}^{m} I_i$ can be large and the concrete choice of the branching candidate can have a huge effect on the computational performance [1]. Our expectations on a branching rule are: (1) It should remove "infeasibilities" as early as possible; (2) It should increase the lower bound as much as possible; (3) It should be computationally easy; and (4) It should be unique (i.e., break ties) in order to avoid random decisions. Many rules have been studied in the TSP, ATSP, and CVRP literature. In particular, the paper [20] provides a literature review and the ATSP case. It suggests the following two criteria to qualify arc $a \in A$ for branching:

1. Let $P \subseteq A$ that one infeasible path or cycle with $a \in P$. The criterion is $\mathrm{PL}(a) := |P|$.
2. The criterion is the optimal objective function value of the node relaxation s.t. $x_a = 0$.

The maximization of criterion 2 is known as strong branching in the literature [1]. In [20] it is suggested to lexicographically (we also always combine criteria lexicographically here) combine strong branching with minimizing criterion 1. The argumentation for this rule is conclusive and matches expectations (1) to (3). But we observed the following issue w.r.t. expectation (4). Let $a' \in A$ be an arc contained in an infeasible path or cycle. Following [20] we have to compute the *strong branching bound* $\mathrm{SB}(a')$:

$$\begin{aligned} &\mathrm{SB}(a') := \min \textstyle\sum_{a \in A \setminus \{a'\}} c_a x_a \\ &\text{s.t.} \sum_{a \in \delta^+(v) \setminus \{a'\}} x_a = 1 \text{ and } \sum_{a \in \delta^-(v) \setminus \{a'\}} x_a = 1 \; \forall\, v \in V, \quad x_a \in \{0,1\} \; \forall\, a \in A. \end{aligned} \tag{$\mathrm{RCAP}_{\mathrm{SB}}$}$$

Our observation is that the values $\mathrm{SB}(a')$ do not distinguish particular arcs, i.e., many arcs of the infeasible path or cycle give the same strong branching bound. This is comprehensible because if we force $x_a = 0$, it is unlikely that all other arcs of the corresponding path or cycle remain. Whenever at least two arcs have the same strong branching bound the choice is random and can be expected to be "wrong" in half of all cases.

Our idea to diversify the strong branching bound is to introduce an additional constraint into ($\mathrm{RCAP}_{\mathrm{SB}}$) in order to force that things change. The constraint reads:

$$\sum_{i=1}^{m} \sum_{a \in I_i} x_a \leq |V| - 2. \tag{4}$$

It forces us to change at least two arcs of the current cycle partition to end up with another cycle partition. This kind of constraints is well-known in a MIP concept that is called *local branching* [6] for a different application. Denoting the bound that is given by model ($\mathrm{RCAP}_{\mathrm{SB}}$) including inequality (4) as $\mathrm{LB}(a)$ for $a \in A$, the following lemma holds.

▶ **Lemma 4.** *$LB((u,v))$ can be computed exactly by a local search over all 2-OPT moves that insert one arc of $\delta^+(u)$ into the optimal solution of the current node relaxation.*

Proof. Inequality (4) and equality $x_a = 0$ constrain to 2-OPT moves. ◀

A natural suggestion is to consider an arc $a \in A$ maximizing $\mathrm{LB}(a)$ for branching.

We remark that LB does also not diversify completely (which is impossible, e.g., if $c_a = 0$ for all $a \in A$) but much better than SB. To break the remaining ties, we introduce another

criterion that depends on the branching history, see [4, Section 10.2] for an overview. Suppose that we just computed the optimal solution of the assignment relaxation of a branching node $j \in \mathbb{N}$ and that the arc $a \in A$ appears in this solution, i.e., $x_a = 1$. Let $z^\star$ be the relaxation's optimal objective value. Then we store the triple $(z^\star, j, a)$ in a set O and define the *average objective value* $\mathrm{AO}(a)$ of the arc $a \in A$ as:

$$\mathrm{AO}(a) := \left(\sum_{(z,j,a') \in O \,:\, a' = a} z \right) / \left| \{(z,j,a') \in O \mid a' = a\} \right|.$$

At this point we considered the following four criteria for choosing a branching candidate $a \in \bigcup_{i=1}^{m} I_i$: $\mathrm{PL}(a)$, $\mathrm{LB}(a)$, $\mathrm{SB}(a)$, and $\mathrm{AO}(a)$. Each of these criteria can be minimized as well as maximized. Also any lexicographic order (e.g., first select all arcs $a \in A$ minimizing $\mathrm{PL}(a)$, of these maximize $\mathrm{LB}(a)$, etc.) can be chosen. This gives rise to $2^4 \cdot 4! = 384$ possibilities which we implemented all in order to prove the optimality of an already optimal solution for the instances: `br17` (ATSP), `gr17` (TSP), and `eil22` (CVRP). Most of the 384 rules are obviously not competitive. But twelve rules are not evidently dominated, see Table 3. We declare the rule ($\max \mathrm{LB}$, $\max \mathrm{AO}$, $\min \mathrm{PL}$, $\max \mathrm{SB}$) as (our) clear winner by considering that computing $\mathrm{LB}(a)$ is much faster ($O(|V|)$) than computing $\mathrm{SB}(a)$ ($O(|V|^2)$ with warm start and $O(|V|^3)$ without).

3.2 Assignment Reduction

The assignment relaxation RCAP′ is derived by deleting the RESOURCE CONSTRAINT from model ($\mathrm{RCAP}_{\mathrm{CIP}}$). It is a valid relaxation which we use for bounding within our branch and bound algorithm. The assignment problems are solved with an $O(|V|^3)$ implementation of the Hungarian method described in the paper [11] that celebrates its 60th birthday this year. The Hungarian algorithm produces optimal dual variables π_u and π_v for each arc $a = (u, v) \in A$. Let z_{LB} be the optimal objective value of RCAP′ and z_{UB} an already known upper bound for the RCAP. Then an arc $a \in A$ can be detached if $z_{\mathrm{LB}} + c_a - \pi_u - \pi_v \geq z_{\mathrm{UB}}$, a rule which is known under the name *reduced cost presolving* [1].

Let $M = \{a \in A \mid x_a = 1\}$ be the solution of some assignment relaxation. It is easy to see that arcs can be detached by imposing $x_a = 0$ for an arc $a \in M$ and $x_a = 1$ for an arc $a \in A \setminus M$ if the corresponding sub-problems turn out to be infeasible or dominated by the best known upper bound. However, solving all these sub-problems can be computationally expensive. This computational burden can be mitigated by performing a local optimization before solving the sub-problems. Namely, if we try to detach $a = (u, v) \in A$ from the current sub-problem, we can locally optimize in $O(|V|)$ over all 2-OPT moves defined by $\delta^+(v) \setminus \{a\}$. If the best objective value during this local optimization is below the best known upper bound we do not have to solve the assignment problem that forces $x_a = 0$ (this is can be done similarly for $a \in A \setminus M$).

3.3 Shortest-Path Reduction

In this section we aim at developing a pruning rule that eliminates an arc $a \in A$ if it can be proven that a feasible path $P \subseteq A$ with $a \in P$ does not exist in the current sub-problem. To this end, we transform the directed graph $D = (V, A)$ into another directed graph D_{SP}. We introduce the node set $V_{\mathrm{SP}} := V \cup \{s, t\}$ of D_{SP}, i.e., we extend D by a source s and a target t. For $a = (u, v) \in A$ we apply the following transformation:

$$A_{\mathrm{SP}}(a) := \begin{cases} \{(u,t),(s,v)\}, \text{ if } a \text{ is a replenishment arc} & \left(c^{\mathrm{SP}}_{(u,t)} := r^1_a,\ c^{\mathrm{SP}}_{(s,v)} := r^2_a\right), \\ \{(u,v)\}, \text{ otherwise} & \left(c^{\mathrm{SP}}_{(u,v)} := r^1_a + r^2_a\right). \end{cases}$$

The transformed graph is $D_{\text{SP}} := (V_{\text{SP}}, A_{\text{SP}}) := \big(V \cup \{s,t\}, \bigcup_{a\in A} A_{\text{SP}}(a)\big)$ with well defined objective coefficients c_a^{SP} for all $a \in A_{\text{SP}}$. Every feasible path must be elementary in a solution to the RCAP and every elementary resource path of $\mathbb{P}(A)$ corresponds to an elementary s-t-path P in D_{SP} by construction. Our elimination criterion for an arc $a \in A$ is as follows. If we can prove that a *shortest* elementary s-t-path P in D_{SP} *such that* $a \in P$ has cost $c(P) > B$ we are allowed to detach a. This elimination criterion is NP-hard to compute, as stated in Lemma 5:

▶ **Lemma 5** (Elementary s-v-t-paths in directed graphs are NP-hard to compute)**.** *Given a directed graph $G = (V, A)$ and three different nodes $s, v, t \in V$, it is NP-complete to decide if G contains an elementary path that starts at s, traverses v, and ends at t.*

Proof. Given a directed graph $D = (V, A)$ with four different nodes $v_1, u_1, v_2, u_2 \in V$ the disjoint path problem (DPP) is to find a v_1-u_1-path and a v_2-u_2-path in D such that the two paths are vertex-disjoint. The DPP is NP-hard, see [8] (the DPP for *undirected graphs* is polynomial, see [18]). An instance of the DPP can be instantiated as an elementary s-v-t-path problem by setting $s = v_1$, $t = u_2$ and by introducing arcs (u_1, v) and (v, v_2). ◀

Fortunately, we can relax the criterion by computing non-elementary paths in D_{SP} and also obtain a valid elimination rule. It can be checked by first computing the shortest-paths from s to all nodes of V, followed by computing the shortest-paths from V to t, and finished by iterating over all arcs of A and to evaluate the elimination criterion. This procedure has complexity $O(|V|^2)$.

3.4 Bin-Packing Reduction

Let J be a set of items with associated weights $w_j \in \mathbb{Q}_+$ for $j \in J$ and a bin capacity $B \in \mathbb{Q}_+$. The standard bin-packing problem is to find a block partition $S_1, \ldots, S_k$ of J with $\sum_{j\in S_k} w_j \le B$ for all blocks $S_1, \ldots, S_k$ such that k is minimal. In a solution of the RCAP the nodes are also assigned to capacitated bins, namely, to resource paths. This gives motivation to derive a bin-packing relaxation of the RCAP that can be used for pruning in the branch and bound tree. To this purpose, we interpret the nodes of our graph as items and the feasible paths as bins. The pruning rule contributes if it can be proven that more bins are needed than available. A valid lower bound on the minimal resource consumption that the node (or item) $u \in V$ will contribute to a feasible path can be computed by solving the following assignment problem:

$$\begin{aligned} w_u := \min\ & \textstyle\sum_{a\in\delta^+(u)} r_a x_a \\ \text{s.t.}\quad & \textstyle\sum_{a\in\delta^+(v)} x_a = 1 \text{ and } \sum_{a\in\delta^-(v)} x_a = 1 \ \ \forall\, v \in V, \quad x_a \ge 0 \ \ \forall\, a \in A. \end{aligned} \tag{RCAP$_{\text{ITEMS}}$}$$

These quantities are used as node weights. Moreover an obviously valid upper bound for the maximal number of feasible paths (or bins) can be computed by solving the following model (RCAP$_{\text{BINS}}$):

$$\begin{aligned} z_{\text{UB}} := \max\ & \textstyle\sum_{a\in\tilde{A}} x_a \\ \text{s.t.}\quad & \textstyle\sum_{a\in\delta^+(v)} x_a = 1 \text{ and } \sum_{a\in\delta^-(v)} x_a = 1 \ \ \forall\, v \in V, \quad x_a \ge 0 \ \ \forall\, a \in A. \end{aligned} \tag{RCAP$_{\text{BINS}}$}$$

It maximizes the number of replenishment arcs $\tilde{A} \subseteq A$ which is equivalent to maximizing the number of resource paths. The following lemma summarizes the bin-packing pruning rule.

▶ **Lemma 6.** *Let I be an instance of the RCAP. Let z_{LB} be any valid lower bound for the optimal solution of the bin-packing problem with item set V, weights w_u derived from*

model ($\text{RCAP}_{\text{ITEMS}}$) *for all* $u \in V$ *and a bin capacity of* B*. Further let* z_{UB} *be the optimal objective value of model* ($\text{RCAP}_{\text{BINS}}$)*. If* $z_{LB} > z_{UB}$ *it is proven that* I *is infeasible.*

Proof. Let $z_{LB} > z_{UB}$ and let I be a feasible instance. There must be a cycle partition $C_1, \ldots, C_k$ containing feasible paths. The value z_{UB} is associated with an optimal solution of ($\text{RCAP}_{\text{BINS}}$), therefore $z_{UB} \geq |\mathbb{P}(\bigcup_{i=1}^{k} C_k)|$. Each path in $\mathbb{P}(\bigcup_{i=1}^{k} C_k)$ provides a feasible assignment of items to bins, i.e., an assignment of nodes to feasible paths, because the weight of each item $v \in V(P)$ is underestimated in a worst case by the optimal objective value w_v of model ($\text{RCAP}_{\text{ITEMS}}$), thus $z_{LB} \leq |\mathbb{P}(\bigcup_{i=1}^{k} C_k)|$. The contradiction is given by $z_{LB} \leq |\mathbb{P}(\bigcup_{i=1}^{k} C_k)|$ and $z_{UB} \geq |\mathbb{P}(\bigcup_{i=1}^{k} C_k)|$. ◀

Since the bin-packing problem is NP-hard, we replace z_{LB} by the lower bounds $L2$ and $L3$ from [12]. These bounds can be computed in $O(|V|)$ for $L2$ and in $O(|V|^3)$ for $L3$ and have a worst case quality of $\frac{2}{3} z_{\text{BP}}$ and $\frac{3}{4} z_{\text{BP}}$ where z_{BP} denotes the optimal objective value of the bin-packing problem.

3.5 Symmetry Reduction

In this section we collect some algorithmic insights found by solving symmetric TSP and CVRP instances with our algorithm. This type of problems can be characterized as having the property that *each resource path is a cycle*, and that the cost function is symmetric. Therefore, every cycle can be reversed, such that the cost and the resource consumption of the tour and the reversed tour are equal. This can be problematic in a branch and bound algorithm that has to search through many essentially identical alternatives.

The capacitated vehicle routing problem (CVRP) [5] is to find a minimal set of cycles, called *tours*, in a complete undirected graph $G = (V \cup \{d\}, E)$ with node demands $r_v \in \mathbb{Q}_+$ for all $v \in V$ such that each node of V is covered exactly once by a cycle, every cycle covers the depot node d exactly once, $\sum_{v \in V \cap C} r_v \leq B$ holds for every cycle C of the solution, and the solution minimizes some linear objective function $c : E \mapsto \mathbb{Q}$. We assume that the minimal number of tours t is known (as most of the articles of the CVRP literature do). An instance of the CVRP can be modeled as a RCAP by introducing t copies of d, using the resource function values of the outgoing arcs of a node to model the demands, and declaring the incoming arcs of d as replenishment arcs. For $t = 1$, TSP instances can be modeled directly as RCAPs. Our first observation is:

▶ **Lemma 7.** *Consider a RCAP instance over the directed graph* $D = (V, A)$ *such that each resource path is a cycle and let* $f : V \mapsto \{1, \ldots, |V|\}$ *be some numbering of the nodes. We only have to consider arcs* $a = (u, v) \in A$ *with* $f(u) < f(v)$ *as branching candidates.*

Proof. Consider the set of cycles $C_1, \ldots, C_k$ of an infeasible solution of the current node relaxation. Then, an infeasible path $P \in \mathbb{P}(\bigcup_{i=1}^{n} C_i)$ exists. Since P is a cycle there is at least one arc $a = (u, v) \in P$ with $f(u) < f(v)$ which can be used as a branching candidate. ◀

Note that, although this attractive rule was originally developed to break symmetries, it can also be used in an ATSP context. However, we could not find an effective way to utilize it in our implementation. The concrete reason is unclear to us. We can only speculate that merely using arcs $a = (u, v)$ with $f(u) < f(v)$ as branching candidates destroys the performance of our branch and bound algorithm because in approximately half of the cases the one arc that increases the lower bound at most is not chosen. Nevertheless, we were able to verify by Lemma 7 that our implementation does not suffer from symmetric cost matrices.

Another symmetry issue refers to the depot copies in the CVRP case. We assume that D does not contain loops and arcs connecting depot nodes. Then, each cycle partition of D is symmetric to $t!$ cycle partitions that arise from interchanging the depot nodes. This problem can be easily resolved by excluding all arcs incident to a depot node as branching candidates. If all other arcs, i.e., all arcs that are not incident to the depot, are fixed to one or zero we always obtain single-customer tours for which each $x_a = 0$ leads to an infeasible RCAP instance.

4 Computational Results

All our computations were performed on computers with an Intel(R) Xeon(R) CPU X5672 with 3.20 GHz, 12 MB cache, and 128 GB of RAM by using a single thread under the operating system Ubuntu 14.04. All implementations are written in the `C++` programming language and compiled by the compiler `g++` 4.8.4 released by the Free Software Foundation.

4.1 RCAP instances from the railway application

The interpretation of the RCAP in rolling stock rotation optimization is to cover a given set of timetabled passenger trips by a set of cycles, called rolling stock rotations. The resource constraint models a limit on the driven distance between two consecutive maintenance services. The main objective is to minimize the number of vehicles and the total distance of deadhead trips (needed to overcome different arrival and departure locations between two trips). We tested our regional and global search algorithms for 15 RCAP instances that are specializations of the rolling stock rotation problem (RSRP) [17]. The RS is called once in the root node of our branch and bound tree. For this application, it is advantageous to use the RS of our previous paper [16]. It is also better to turn off the bin-packing reduction in the railway application. By using the RS strictly as presented in this paper we get similar results for these 15 instances w.r.t. solution quality and computation time for less constrained instances (e.g., all with 8000 km and 6000 km and all with 97 nodes). But the computation times for the large and hard constrained instances (e.g., `RCAP_02`, see below) increase w.r.t. to our previous regional search algorithm [16].

Table 1 reports our results for the 15 instances that arise from RSRPs that are associated with three timetables (indicated by the number of nodes in column three) for different upper bounds of a dedicated maintenance constraint denoted in column two. The *root gap* in column four is defined as $\frac{(c^\star - \tilde{c})}{c^\star} * 100$ (all gaps in this paper are computed in this way), i.e., the worst case optimality gap in percent, where $c^\star > 0$ is the objective value of the regionally optimal solution and $\tilde{c}$ the value of the lower bound obtained in the root node of the branching tree. Columns five, six, and seven contain the number of branch and bound nodes, the computation time, and the solution status on termination of the branch and bound algorithm.

In the industrial application, the instances associated with a maintenance constraint of 8000 km are the ones of interest that could all be solved to proven optimality fast. Also tighter constrained instances are solved with very high solution quality. The most difficult instances `RCAP_02`, `RCAP_03`, and `RCAP_12` display worst case optimality gaps. Nevertheless, we claim that they are also completely "resolved" from an applied point of view. In fact, the very large lower bound proves practical inefficiency of the solution beyond doubt.

Table 1 Results for RCAP instances from the railway application.

instance	B [km]	$\|V\|$	root gap	nodes	hh:mm:ss	proved
RCAP_01	1000	617	–	1	00:00:00	infeasibility
RCAP_02	2000	617	25.88	15105	15:50:56	9.81 % gap
RCAP_03	4000	617	3.95	51483	15:45:57	0.21 % gap
RCAP_04	6000	617	0.19	143	03:18:07	optimality
RCAP_05	8000	617	0.13	43	00:07:37	optimality
RCAP_06	1000	97	–	1	00:00:00	infeasibility
RCAP_07	2000	97	12.71	41	00:00:10	optimality
RCAP_08	4000	97	0.00	1	00:00:02	optimality
RCAP_09	6000	97	0.00	1	00:00:02	optimality
RCAP_10	8000	97	0.00	1	00:00:02	optimality
RCAP_11	1000	310	–	1	00:00:00	infeasibility
RCAP_12	2000	310	38.16	944551	16:07:46	16.71 % gap
RCAP_13	4000	310	16.70	119159	09:17:54	optimality
RCAP_14	6000	310	7.78	2053	00:08:33	optimality
RCAP_15	8000	310	7.78	87	00:21:40	optimality

Table 2 Summary of regional search for VRP instances.

type	number of instances	arithmethic mean		shifted geometric mean [1] (shift 1)	
ATSP	19	1.99	(1.70)	1.51	(1.21)
CVRP	106	0.89	(5.09)	0.63	(3.81)
ACVRP	8	1.84		1.34	
TSP	65	2.22	(2.60)	1.71	(1.97)
all	198	1.47	(3.91)	1.04	(2.78)

4.2 TSP, ATSP, CVRP, and ACVRP instances from the literature

We also made experiments for a large number of instances taken from the literature [14, 15] for which we use the regional search algorithm and the branch and bound algorithm strictly as presented in this paper. We present results for all ATSP instances from [15] and for the TSP instances with less than 500 nodes. From [14] we consider all CVRP and ACVRP (the ACVRP instances were not considered in [16]) instances from the test sets A, B, E, F, G, M, P, and V except for six instances for which we could not verify the objective values of the solutions provided in the library (otherwise uncomparable results would appear).

Table 2 provides mean values for the column "bk gap" (i.e., the deviation in percent to the best known objective value) of Table 4 in the appendix. The same summary is made in our previous paper [16] and we provide the corresponding values in braces. In comparison to [16], the exact search over the regions increases solution quality. In comparison to other more problem specific heuristics (especially for the symmetric TSP, see [10]) our regional search is almost competitive w.r.t. solution quality. It is definitely competitive in solving asymmetric instances to proven optimality, as reported in the last three columns of Table 4: 18 of 19 ATSP instances from [15] and all ACVRP instances considered in [7] are solved to proven optimality. These results give evidence that our algorithms are powerful tools for a wide variety of resource constrained assignment problems ranging from recent railway applications via VRPs to classical TSPs and ATSPs.

References

1 Tobias Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2009.

2 M. L. Balinski and R. E. Gomory. A primal method for the assignment and transportation problems. *Management Science*, 10(3):578–593, 1964.

3 M. L. Balinski and Andrew Russakoff. On the assignment polytope. *SIAM Review*, 16(4):pp. 516–525, 1974.

4 Timo Berthold. *Heuristic algorithms in global MINLP solvers.* PhD thesis, Technische Universität Berlin, 2014.

5 G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

6 Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1-3):23–47, 2003.

7 Matteo Fischetti, Paolo Toth, and Daniele Vigo. A Branch-and-Bound Algorithm for the Capacitated Vehicle Routing Problem on Directed Graphs. *Operations Research*, 42(5):846–859, 1994.

8 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

9 Chris Groër, Bruce Golden, and Edward Wasil. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2(2):79–101, 2010.

10 Keld Helsgaun. General k-opt submoves for the Lin–Kernighan TSP heuristic. *Mathematical Programming Computation*, 1(2-3):119–163, 2009.

11 H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

12 Silvano Martello and Paolo Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28(1):59–70, 1990.

13 Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved Branch-Cut-and-Price for Capacitated Vehicle Routing. In Jon Lee and Jens Vygen, editors, *Integer Programming and Combinatorial Optimization*, volume 8494 of *Lecture Notes in Computer Science*, page 393–403. Springer International Publishing, 2014.

14 T. Ralphs. Branch cut and price resource web (http://www.branchandcut.org), June 2014.

15 G. Reinelt. TSPLIB - A T.S.P. Library. Technical Report 250, Universität Augsburg, Institut für Mathematik, Augsburg, 1990.

16 Markus Reuther. Local Search for the Resource Constrained Assignment Problem. In *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 42 of *OASIcs*, pages 62–78, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

17 Markus Reuther, Ralf Borndörfer, Thomas Schlechte, and Steffen Weider. Integrated optimization of rolling stock rotations for intercity railways. In *Proceedings of RailCopenhagen*, Copenhagen, Denmark, May 2013.

18 Yossi Shiloach. The two paths problem is polynomial. Technical report, Stanford University, Stanford, CA, USA, 1978.

19 P. Toth and D. Vigo. *Vehicle Routing.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 2014.

20 Marcel Turkensteen, Diptesh Ghosh, Boris Goldengorin, and Gerard Sierksma. Tolerance-based Branch and Bound algorithms for the ATSP. *EJOR*, 189(3):775–788, 2008.

A Appendix: Tables

Table 3 Computational evaluation of branching rules: The notation defines the lexicographic order of the criteria by that arcs are selected as branching candidates. The last column denotes the number of branching nodes needed to proof optimality for an already optimal incumbent solution.

instance	branching rule	nodes
br17	max SB min PL max AO max LB	76425
eil22	max SB min PL max AO max LB	65769
gr17	max SB min PL max AO max LB	765
br17	max SB min PL max LB max AO	78579
eil22	max SB min PL max LB max AO	59833
gr17	max SB min PL max LB max AO	785
br17	max SB max AO min PL max LB	52415
eil22	max SB max AO min PL max LB	61155
gr17	max SB max AO min PL max LB	781
br17	max SB max AO max LB min PL	44581
eil22	max SB max AO max LB min PL	61247
gr17	max SB max AO max LB min PL	781
br17	max SB max LB min PL max AO	35959
eil22	max SB max LB min PL max AO	57941
gr17	max SB max LB min PL max AO	795
br17	max SB max LB max AO min PL	32159
eil22	max SB max LB max AO min PL	57853
gr17	max SB max LB max AO min PL	795
br17	max LB max SB min PL max AO	44233
eil22	max LB max SB min PL max AO	40961
gr17	max LB max SB min PL max AO	485
br17	max LB max SB max AO min PL	30103
eil22	max LB max SB max AO min PL	41193
gr17	max LB max SB max AO min PL	485
br17	max LB min PL max SB max AO	44505
eil22	max LB min PL max SB max AO	41199
gr17	max LB min PL max SB max AO	493
br17	max LB min PL max AO max SB	45355
eil22	max LB min PL max AO max SB	42747
gr17	max LB min PL max AO max SB	535
br17	max LB max AO max SB min PL	26821
eil22	max LB max AO max SB min PL	43383
gr17	max LB max AO max SB min PL	531
br17	max LB max AO min PL max SB	26847
eil22	max LB max AO min PL max SB	43391
gr17	max LB max AO min PL max SB	531

Table 4 Regional and global search for VRP instances. The third column gives the number of nodes of the considered RCAP instance; the fourth column is the deviation in percentage of the initial solution for our regional search (computed with a poor greedy heuristic) w.r.t. the best known objective value [9] (column five). The columns "bk gap" and "lb gap" give the deviation in percent of the regionally optimal objective value (column "reg. sec." denotes its computation seconds) w.r.t. column "best" and w.r.t. the lower bound obtained in the root node of the branching tree, respectively. The last two columns give the number of branching nodes and the computation time if our branch and bound approach was able to solve all remaining sub-problems.
(For `M-n200-k17` (`G-n262-k25`) we computed a solution with objective value 1344 (5856). These values are below the best known values provided in [9] and excluded in Table 2.)

instance	type	$\|V\|$	initial gap	best	lb gap	bk gap	reg. sec.	nodes	dd:hh:mm:ss
A034-02f	ACVRP	35	48.91	1406	14.15	0.00	7.4	18093	00:00:00:18
A036-03f	ACVRP	38	46.43	1644	12.78	4.08	5.4	37037	00:00:00:38
A039-03f	ACVRP	41	55.16	1654	9.92	4.00	10.9	11043	00:00:00:25
A045-03f	ACVRP	47	58.19	1740	6.72	0.11	5.9	2025	00:00:00:10
A048-03f	ACVRP	50	63.05	1891	8.39	2.12	4.4	11865	00:00:00:19
A056-03f	ACVRP	58	65.61	1739	13.64	2.41	15.1	1192799	00:00:30:17
A065-03f	ACVRP	67	69.61	1974	7.45	0.00	32.7	83185	00:00:02:23
A071-03f	ACVRP	73	71.24	2054	10.11	2.00	10.1	121205	00:00:05:27
br17	ATSP	17	76.65	39	100.00	0.00	2.6	152825	00:00:00:23
ft53	ATSP	53	50.52	6905	16.97	3.33	23.1	441917	00:00:15:39
ft70	ATSP	70	31.04	38673	2.09	0.29	17.2	1462829	00:00:58:00
ftv170	ATSP	171	61.45	2755	6.87	2.48	37.2	6683339	01:03:08:19
ftv33	ATSP	34	42.56	1286	13.63	6.27	5.3	157	00:00:00:05
ftv35	ATSP	36	40.44	1473	7.32	1.14	4.0	1353	00:00:00:04
ftv38	ATSP	39	38.90	1530	7.05	1.10	4.5	5407	00:00:00:12
ftv44	ATSP	45	39.77	1613	8.43	2.89	4.7	2323	00:00:00:09
ftv47	ATSP	48	58.59	1776	10.22	3.48	4.6	26341	00:00:01:09
ftv55	ATSP	56	59.54	1608	15.09	4.85	4.5	209665	00:00:08:42
ftv64	ATSP	65	61.55	1839	10.08	3.92	12.3	46923	00:00:03:02
ftv70	ATSP	71	59.84	1950	11.35	2.11	5.9	452675	00:00:16:56
kro124p	ATSP	100	82.71	36230	6.28	0.07	166.6	14253731	01:23:08:01
p43	ATSP	43	8.77	5620	97.37	0.05	6.5		
rbg323	ATSP	323	79.37	1326	0.90	0.90	112.0	739	00:00:08:02
rbg358	ATSP	358	83.58	1163	0.34	0.34	113.2	663	00:00:02:46
rbg403	ATSP	403	69.02	2465	0.88	0.88	94.6	177	00:00:06:39
rbg443	ATSP	443	68.80	2720	0.98	0.98	121.3	43	00:00:06:32
ry48p	ATSP	48	73.42	14422	15.64	2.80	10.2	150917	00:00:05:24
A-n32-k5	CVRP	36	52.94	784	31.63	0.00	22.0		
A-n33-k5	CVRP	37	49.70	661	38.07	2.07	62.1		
A-n33-k6	CVRP	38	42.92	742	36.74	0.13	84.4		
A-n34-k5	CVRP	38	52.73	778	35.99	1.39	67.5		
A-n36-k5	CVRP	40	50.00	799	38.17	0.99	68.3		
A-n37-k5	CVRP	41	56.61	669	26.99	1.33	22.7		
A-n37-k6	CVRP	42	42.03	949	45.31	0.00	321.0		
A-n38-k5	CVRP	42	54.74	730	43.72	0.27	69.3		
A-n39-k5	CVRP	43	59.86	822	37.08	0.72	206.6		
A-n39-k6	CVRP	44	55.06	831	38.42	0.24	112.0		
A-n44-k6	CVRP	49	56.03	937	31.10	0.21	75.2		
A-n45-k6	CVRP	50	55.00	944	37.18	0.00	118.7		
A-n45-k7	CVRP	51	51.89	1146	40.40	0.43	1025.8		
A-n46-k7	CVRP	52	58.68	914	37.31	0.00	187.9		
A-n48-k7	CVRP	54	52.35	1073	39.09	2.45	549.6		
A-n53-k7	CVRP	59	59.26	1010	39.45	1.37	677.7		
A-n54-k7	CVRP	60	54.68	1167	51.99	3.07	1700.3		
A-n55-k9	CVRP	63	54.78	1073	40.04	1.01	491.2		
A-n60-k9	CVRP	68	54.23	1354	54.60	1.17	4232.7		
A-n61-k9	CVRP	69	55.98	1034	41.66	0.29	1080.4		
A-n62-k8	CVRP	69	59.67	1288	48.94	2.13	3293.0		
A-n63-k10	CVRP	72	54.07	1314	49.89	0.38	3884.1		
A-n63-k9	CVRP	71	54.22	1616	48.84	1.10	4342.6		

Continued on next page

Table 4 – continued from previous page

instance	type	$\|V\|$	initial gap	best	lb gap	bk gap	reg. sec.	nodes	dd:hh:mm:ss
A-n64-k9	CVRP	72	57.66	1401	41.51	1.27	3110.8		
A-n65-k9	CVRP	73	59.86	1174	37.05	0.00	1275.8		
A-n69-k9	CVRP	77	62.16	1159	37.10	0.69	1497.4		
A-n80-k10	CVRP	89	58.80	1763	41.97	0.96	6728.1		
att-n48-k4	CVRP	51	63.86	40002	26.12	0.52	83.3		
bayg-n29-k4	CVRP	32	55.70	2050	17.71	0.00	12.2	34154469	00:06:08:31
bays-n29-k5	CVRP	33	46.72	2963	25.89	0.00	29.2		
B-n31-k5	CVRP	35	29.56	672	30.06	0.00	41.0		
B-n34-k5	CVRP	38	44.35	788	32.83	0.13	76.2		
B-n35-k5	CVRP	39	53.30	955	37.28	0.00	130.4		
B-n38-k6	CVRP	43	57.34	805	43.85	0.00	174.0		
B-n39-k5	CVRP	43	63.20	549	52.82	0.00	61.3		
B-n41-k6	CVRP	46	54.90	829	61.88	0.00	176.5		
B-n43-k6	CVRP	48	58.38	742	52.70	0.00	425.0		
B-n44-k7	CVRP	50	50.81	909	61.72	0.00	336.1		
B-n45-k5	CVRP	49	53.06	751	45.94	0.00	184.2		
B-n45-k6	CVRP	50	56.23	678	43.11	0.59	349.7		
B-n50-k7	CVRP	56	67.11	741	34.82	0.00	314.5		
B-n50-k8	CVRP	57	50.13	1312	56.93	1.20	2457.7		
B-n51-k7	CVRP	57	53.78	1032	36.88	0.10	566.7		
B-n52-k7	CVRP	58	66.00	747	61.50	0.13	846.0		
B-n56-k7	CVRP	62	66.41	707	62.94	0.00	673.5		
B-n57-k7	CVRP	63	29.65	1153	66.67	2.45	1912.7		
B-n57-k9	CVRP	65	43.09	1598	34.31	0.68	1695.2		
B-n63-k10	CVRP	72	60.39	1496	58.82	2.67	3284.2		
B-n64-k9	CVRP	72	66.83	861	46.58	0.23	2814.0		
B-n66-k9	CVRP	74	52.97	1316	58.12	0.15	3445.1		
B-n67-k10	CVRP	76	65.36	1032	43.33	0.19	3108.3		
B-n68-k9	CVRP	76	60.09	1272	56.44	0.16	2461.7		
B-n78-k10	CVRP	87	62.37	1221	61.02	0.00	8131.7		
dantzig-n42-k4	CVRP	45	34.67	1142	49.61	1.97	76.3		
E-n101-k14	CVRP	114	64.54	1071	29.07	2.10	8541.6		
E-n101-k8	CVRP	108	65.83	817	20.61	0.97	2077.9		
E-n13-k4	CVRP	16	38.10	247	10.93	0.00	2.7	143	00:00:00:04
E-n22-k4	CVRP	25	38.73	375	30.13	0.00	4.5	74055	00:00:00:45
E-n23-k3	CVRP	25	50.48	569	21.44	0.00	4.9	9321	00:00:00:09
E-n30-k3	CVRP	32	52.28	534	40.97	0.56	70.3		
E-n31-k7	CVRP	37	66.93	379	19.26	0.00	11.3	155737	00:00:02:24
E-n33-k4	CVRP	36	34.61	835	28.50	0.00	119.8		
E-n51-k5	CVRP	55	62.00	521	21.75	3.16	55.0		
E-n76-k10	CVRP	85	63.16	830	29.86	0.48	1899.9		
E-n76-k14	CVRP	89	47.43	1021	35.36	1.35	3552.1		
E-n76-k7	CVRP	82	69.68	682	23.75	2.43	201.5		
E-n76-k8	CVRP	83	61.98	735	26.28	0.94	290.0		
F-n135-k7	CVRP	141	71.80	1162	52.65	0.51	6891.6		
F-n45-k4	CVRP	48	65.61	724	42.99	0.55	32.9		
F-n72-k4	CVRP	75	74.10	237	31.22	0.00	151.2		
fri-n26-k3	CVRP	28	23.56	1353	17.75	0.37	6.1	842175	00:00:06:11
gr-n17-k3	CVRP	19	29.88	2685	28.31	0.00	5.6	13977	00:00:00:09
gr-n21-k3	CVRP	23	36.02	3704	27.54	0.00	6.3	29293	00:00:00:17
gr-n24-k4	CVRP	27	46.04	2053	28.30	0.00	11.7	5919153	00:00:47:50
gr-n48-k3	CVRP	50	66.55	5985	25.71	0.22	28.3		
hk-n48-k4	CVRP	51	56.96	14749	25.74	0.09	361.6		
M-n101-k10	CVRP	110	66.26	820	33.98	0.49	657.2		
M-n121-k7	CVRP	127	67.19	1034	64.71	8.09	37860.9		
M-n151-k12	CVRP	162	67.60	1053	34.00	0.28	12133.0		
M-n200-k17	CVRP	215	66.28	1373	-	-	-		
P-n101-k4	CVRP	104	71.61	681	15.04	2.44	219.1		
P-n16-k8	CVRP	23	1.75	450	14.67	0.00	4.1	3033	00:00:00:07
P-n19-k2	CVRP	20	37.09	212	21.70	0.00	4.2	16959	00:00:00:12
P-n20-k2	CVRP	21	43.31	216	19.46	2.26	3.1	10593	00:00:00:08
P-n21-k2	CVRP	22	42.03	211	18.48	0.00	3.7	4787	00:00:00:05

Continued on next page

Table 4 – continued from previous page

instance	type	$\|V\|$	initial gap	best	lb gap	bk gap	reg. sec.	nodes	dd:hh:mm:ss
P-n22-k2	CVRP	23	45.04	216	17.13	0.00	5.6	6765	00:00:00:07
P-n22-k8	CVRP	29	20.66	603	39.97	0.00	19.6	818203	00:00:09:20
P-n23-k8	CVRP	30	19.73	529	37.62	0.00	26.7	9609861	00:02:09:07
P-n40-k5	CVRP	44	57.08	458	18.12	0.00	12.8		
P-n45-k5	CVRP	49	61.07	510	19.22	0.00	17.1		
P-n50-k10	CVRP	59	41.46	696	28.43	0.57	407.6		
P-n50-k7	CVRP	56	52.08	554	22.10	1.25	69.3		
P-n50-k8	CVRP	57	50.43	631	31.54	5.68	353.7		
P-n51-k10	CVRP	60	44.62	741	31.44	2.11	372.3		
P-n55-k10	CVRP	64	48.74	694	25.71	0.86	388.4		
P-n55-k15	CVRP	69	28.28	989	38.10	4.35	4983.0		
P-n55-k7	CVRP	61	61.18	568	21.38	2.07	146.9		
P-n55-k8	CVRP	62	62.93	588	19.83	1.18	105.0		
P-n60-k10	CVRP	69	52.55	744	29.61	2.11	656.2		
P-n60-k15	CVRP	74	42.59	968	31.49	0.72	1568.4		
P-n65-k10	CVRP	74	57.67	792	26.28	1.37	339.0		
P-n70-k10	CVRP	79	62.34	827	29.73	1.66	704.2		
P-n76-k4	CVRP	79	74.01	593	16.97	1.33	64.0		
P-n76-k5	CVRP	80	68.19	627	20.59	2.18	90.6		
swiss-n42-k5	CVRP	46	46.79	1668	31.85	1.24	30.9		
ulysses-n16-k3	CVRP	19	100.00	7965	18.75	2.60	6.1	5871	00:00:00:07
ulysses-n22-k4	CVRP	25	32.88	9179	34.51	1.21	21.4	60874403	00:07:31:41
a280	TSP	280	8.16	2579	8.94	3.08	1063.7		
att48	TSP	48	78.68	10628	22.02	1.67	9.4	777323	00:00:21:35
bayg29	TSP	29	65.19	1610	10.56	0.00	6.1	2661	00:00:00:09
bays29	TSP	29	64.88	2020	12.93	0.30	4.3	2441	00:00:00:07
berlin52	TSP	52	66.03	7542	21.54	5.88	14.7	22145	00:00:01:50
bier127	TSP	127	69.98	118282	20.36	1.68	940.1		
brazil58	TSP	58	80.35	25395	35.50	1.12	23.2	741555	00:00:41:02
brg180	TSP	180	98.36	1950	100.00	2.99	256.9		
burma14	TSP	14	27.16	3323	17.33	0.00	3.3	191	00:00:00:05
ch130	TSP	130	87.22	6110	29.68	1.93	600.4		
ch150	TSP	150	87.64	6528	16.09	1.45	323.0		
d198	TSP	198	29.86	15780	33.40	0.92	1752.7		
d493	TSP	493	69.17	35002	15.97	2.89	11463.8		
dantzig42	TSP	42	0.00	699	23.89	0.00	8.6	224263	00:00:03:58
eil101	TSP	101	69.50	629	11.75	2.78	133.3		
eil51	TSP	51	67.43	426	13.16	1.62	16.3	765927	00:00:21:30
eil76	TSP	76	72.68	538	13.26	3.58	63.6	1929865	00:02:55:49
fl417	TSP	417	78.61	11861	37.68	0.40	24856.1		
fri26	TSP	26	17.81	937	11.10	0.00	4.6	553	00:00:00:06
gil262	TSP	262	90.96	2378	21.33	2.66	1593.5		
gr120	TSP	120	86.12	6942	18.18	3.14	107.4		
gr137	TSP	137	28.07	69853	19.10	0.96	211.8		
gr17	TSP	17	55.84	2085	20.77	0.00	5.6	843	00:00:00:03
gr202	TSP	202	30.94	40160	16.45	2.92	442.3		
gr21	TSP	21	59.11	2707	10.60	0.00	3.5	43	00:00:00:05
gr229	TSP	229	25.15	134602	19.48	1.54	3895.2		
gr24	TSP	24	62.98	1272	17.30	0.00	5.9	215	00:00:00:06
gr431	TSP	431	26.45	171414	21.27	5.88	31311.0		
gr48	TSP	48	74.56	5046	18.28	0.30	12.5	3900747	00:01:24:20
gr96	TSP	96	31.85	55209	16.99	0.15	290.8		
hk48	TSP	48	76.21	11461	16.13	2.61	9.6	141947	00:00:04:43
kroA100	TSP	100	88.88	21282	19.71	0.00	222.8		
kroA150	TSP	150	90.79	26524	23.00	5.07	822.6		
kroA200	TSP	200	92.15	29368	24.31	3.76	606.8		
kroB100	TSP	100	85.91	22141	25.83	2.20	237.3		
kroB150	TSP	150	90.44	26130	24.08	3.14	565.2		
kroB200	TSP	200	91.01	29437	23.19	3.41	1018.1		
kroC100	TSP	100	88.69	20749	23.10	4.68	82.9		
kroD100	TSP	100	87.55	21294	27.39	6.52	371.0		
kroE100	TSP	100	88.28	22068	25.71	1.74	120.2		

Continued on next page

Table 4 – continued from previous page

instance	type	$\|V\|$	initial gap	best	lb gap	bk gap	reg. sec.	nodes	dd:hh:mm:ss
lin105	TSP	105	60.58	14379	39.56	2.96	181.1		
lin318	TSP	318	64.94	42029	38.36	5.06	3329.4		
pcb442	TSP	442	77.07	50778	11.32	3.84	7646.0		
pr107	TSP	107	29.40	44303	46.48	2.05	1204.9		
pr124	TSP	124	40.34	59030	34.54	0.73	494.6		
pr136	TSP	136	66.28	96772	15.11	3.98	488.2		
pr144	TSP	144	37.41	58537	66.33	1.49	847.3		
pr152	TSP	152	54.23	73682	42.51	1.58	1713.9		
pr226	TSP	226	27.21	80369	39.11	2.01	3059.9		
pr264	TSP	264	36.99	49135	35.98	4.75	8072.2		
pr299	TSP	299	42.29	48191	19.47	2.69	4455.0		
pr439	TSP	439	60.38	107217	31.70	4.75	20186.8		
pr76	TSP	76	28.27	108159	30.33	2.28	147.5		
rat195	TSP	195	42.36	2323	14.17	4.83	551.8		
rat99	TSP	99	42.98	1211	11.46	1.54	67.4		
rd100	TSP	100	84.36	7910	21.89	5.80	229.0		
rd400	TSP	400	92.91	15281	20.93	2.24	3940.3		
si175	TSP	175	18.79	21407	6.00	0.59	382.1		
st70	TSP	70	80.21	675	25.22	2.74	65.5		
swiss42	TSP	42	55.08	1273	22.44	2.15	9.9	19241	00:00:00:42
ts225	TSP	225	54.20	126643	11.63	3.20	1431.0		
tsp225	TSP	225	62.16	3916	12.98	0.31	494.2		
u159	TSP	159	3.00	42080	17.66	0.00	139.4		
ulysses16	TSP	16	29.03	6859	18.38	0.00	3.5	549	00:00:00:05
ulysses22	TSP	22	42.51	7013	24.58	0.00	4.3	10923	00:00:00:11

Approximation Algorithms for Mixed, Windy, and Capacitated Arc Routing Problems

René van Bevern[1], Christian Komusiewicz[2], and Manuel Sorge[2]

1 **Novosibirsk State University, Novosibirsk, Russia,** `rvb@nsu.ru`
2 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
`{christian.komusiewicz,manuel.sorge}@tu-berlin.de`

Abstract

We show that any $\alpha(n)$-approximation algorithm for the n-vertex metric asymmetric Traveling Salesperson problem yields $O(\alpha(C))$-approximation algorithms for various mixed, windy, and capacitated arc routing problems. Herein, C is the number of weakly-connected components in the subgraph induced by the positive-demand arcs, a number that can be expected to be small in applications. In conjunction with known results, we derive constant-factor approximations if $C \in O(\log n)$ and $O\big(\log C / \log\log C\big)$-approximations in general.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.1.6 Optimization, G.2.1 Combinatorics, G.2.2 Graph Theory, I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases vehicle routing, transportation, Rural Postman, Chinese Postman, NP-hard problem, parameterized algorithm, combinatorial optimization

Digital Object Identifier 10.4230/OASIcs.ATMOS.2015.130

1 Introduction

Golden and Wong [16] introduced the Capacitated Arc Routing problem in order to model the search for minimum-cost routes for vehicles of equal capacity that are initially located in a vehicle depot and have to serve all "customer" demands. Applications of Capacitated Arc Routing include snow plowing, waste collection, meter reading, and newspaper delivery [7]. Herein, the customer demands require that roads of a road network are served. The road network is modeled as a graph whose edges represent roads and whose vertices can be thought of as road intersections. The customer demands are modeled as positive integers assigned to edges of this network. Moreover, each edge has a travel cost.

Capacitated Arc Routing Problem (CARP)
Instance: An undirected graph $G = (V, E)$, a *depot* vertex $v_0 \in V$, travel costs $c\colon E \to \mathbb{N} \cup \{0\}$, edge demands $d\colon E \to \mathbb{N} \cup \{0\}$, and a vehicle capacity Q.
Task: Find a set W of closed walks in G, each corresponding to the route of one vehicle and passing through the depot vertex v_0, and a serving function $s\colon W \to 2^E$ such that

- $\sum_{w \in W} c(w)$ is minimized, where $c(w) := \sum_{i=1}^{\ell} c(e_i)$ for a walk $w = (e_1, e_2, \ldots, e_\ell) \in E^\ell$,
- each closed walk $w \in W$ *serves* a subset $s(w)$ of edges of w and $\sum_{e \in s(w)} d(e) \leq Q$,
- each edge e with $d(e) > 0$ is served by exactly one walk in W.

Note that vehicle routes may traverse each vertex or edge of the input graph multiple times. Well-known special cases of CARP are the NP-hard Rural Postman Problem [21], where the vehicle capacity is unbounded and hence, the goal is to find a shortest possible route for one vehicle that visits all positive-demand edges, and the polynomial-time solvable Chinese Postman Problem [9, 10], where additionally all edges have positive demand.

15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'15).
Editors: Giuseppe F. Italiano and Marie Schmidt; pp. 130–143

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Mixed and windy problem variants. CARP is polynomial-time constant-factor approximable [4, 20, 25]. It is natural to study the approximability of generalizations of CARP on directed, mixed, and windy graphs. This is also noted in a recent survey on the computational complexity of arc routing problems by van Bevern, Niedermeier, Sorge, and Weller [5, Challenge 5]. Herein, a mixed graph may contain directed arcs in addition to undirected edges in order to model one-way roads or the requirement of servicing a road in a *specific* direction or in *both* directions. In a windy graph, the cost for traversing an undirected edge $\{u, v\}$ in the direction from u to v may be different from the cost for traversing it in the opposite direction. (This models sloped roads, for example.) In this work, we present approximation algorithms for mixed and windy variants of CARP. To formally state the problem, we need some terminology related to mixed graphs.

▶ **Definition 1** (Walks in mixed and windy graphs)**.** A *mixed graph* is a triple $G = (V, E, A)$, where V is a set of *vertices*, $E \subseteq \{\{u, v\} \mid u, v \in V\}$ is a set of *(undirected) edges*, $A \subseteq V \times V$ is a set of *(directed) arcs* (that might contain loops), and no pair of vertices has an arc *and* and an edge between them. The *head* of an arc $(u, v) \in V \times V$ is v, its *tail* is u.

A *walk in* G is a sequence $w = (a_1, a_2, \ldots, a_\ell)$ such that, for each $a_i = (u, v)$, $1 \leq i \leq \ell$, we have $(u, v) \in A$ or $\{u, v\} \in E$ and such that the tail of a_i is the head of a_{i-1} for $1 < i \leq \ell$. If (u, v) occurs in w, then we say that w *traverses* the arc $(u, v) \in A$ or the edge $\{u, v\} \in E$, respectively. If the tail of a_1 is the head of a_ℓ, then we call w a *closed walk*.

If $c\colon V \times V \to \mathbb{N} \cup \{0, \infty\}$ is the *travel cost* between vertices of G, the *cost of a walk* $w = (a_1, \ldots, a_\ell)$ is $c(w) := \sum_{i=1}^{\ell} c(a_i)$. The *cost of a set* W *of walks* is $c(W) := \sum_{w \in W} c(w)$.

Formally, we present approximation algorithms for the following problem.

Mixed and Windy Capacitated Arc Routing Problem (MWCARP)
Instance: A mixed graph $G = (V, E, A)$, a depot vertex $v_0 \in V$, travel costs $c\colon V \times V \to \mathbb{N} \cup \{0, \infty\}$, demands $d\colon E \cup A \to \mathbb{N} \cup \{0\}$, and a vehicle capacity Q.
Task: Find a minimum-cost set W of closed walks in G, each passing through the depot vertex v_0, and a serving function $s\colon W \to 2^{E \cup A}$ such that

- each $w \in W$ *serves* a subset $s(w)$ of the edges and arcs it traverses and $\sum_{e \in s(w)} d(e) \leq Q$,
- each edge or arc e with $d(e) > 0$ is served by exactly one walk in W.

For brevity, we use the term "arc" to refer to both edges and arcs. Besides studying the approximability of MWCARP, we also consider the following special case:

Mixed and Windy Rural Postman Problem (MWRPP)
Instance: A mixed graph $G = (V, E, A)$ with travel costs $c : V \times V \to \mathbb{N} \cup \{0, \infty\}$ and a set $R \subseteq E \cup A$ of *required arcs*.
Task: Find a minimum-cost closed walk in G traversing all arcs in R.

If, moreover, $E = \emptyset$, then we obtain the Directed Rural Postman Problem (DRPP).

Relation to metric asymmetric TSP. In the development of approximation algorithms for MWCARP, one has to be aware of the fact that, even for DRPP, there cannot be approximation algorithms better than those for the following strongly related variant of TSP:

Metric Asymmetric Traveling Salesperson Problem ($\triangle$-ATSP)
Instance: A set V of vertices and travel costs $c\colon V \times V \to \mathbb{N} \cup \{0\}$ satisfying the triangle inequality $c(u, v) \leq c(u, w) + c(w, v)$ for all $u, v, w \in V$.
Task: Find a minimum-cost cycle that visits every vertex in V exactly once.

Given a $\triangle$-ATSP instance, one obtains an equivalent DRPP instance simply by adding a zero-cost loop to each vertex and by adding these loops to the set R of required arcs. This leads to the following observation.

▶ **Observation 2.** *Any α-approximation for* DRPP *yields an α-approximation for* $\triangle$-ATSP.

The constant-factor approximability of $\triangle$-ATSP is a long-standing open problem, in contrast to the *symmetric* metric TSP, where the cost of an arc does not depend on its direction. Symmetric metric TSP admits the famous 3/2-approximation by Christofides [6] and Serdyukov [23]. For $\triangle$-ATSP, however, the relatively recent $O(\log n/\log\log n)$-approximation by Asadpour, Goemans, Mądry, Gharan, and Saberi [2] is the first asymptotic improvement over the $O(\log n)$-approximation by Frieze, Galbiati, and Maffioli [15] from 1982.

Our contribution. As discussed above, any α-approximation for DRPP yields an α-approximation for $\triangle$-ATSP. Our contribution is the following theorem for the converse direction.

▶ **Theorem 3.** *If n-vertex* $\triangle$-ATSP *is $\alpha(n)$-approximable in $t(n)$ time, then*
(i) *n-vertex* DRPP *is $(\alpha(C)+1)$-approximable in $O(t(C)+n^3\log n)$ time,*
(ii) *n-vertex* MWRPP *is $(\alpha(C)+3)$-approximable in $O(t(C)+n^3\log n)$ time, and*
(iii) *n-vertex* MWCARP *is $O(\alpha(C+1))$-approximable in $O(t(C+1)+n^3\log n)$ time,*
where C is the number of weakly connected components in the subgraph induced by the positive-demand arcs and edges.

The theorem shows that, although MWCARP is generally not easier to approximate than $\triangle$-ATSP, the approximation quality of MWCARP depends mainly on the number C of weakly connected components in the subgraph induced by positive-demand arcs. There are applications where C is small, which is also exploited in exact exponential-time algorithms for DRPP [12, 17, 24]. For example, the company Berliner Stadtreinigungsbetriebe provided us with instances arising in snow plowing in Berlin, in which the required arcs induce a subgraph with only three or four weakly connected components.

A consequence of Theoem 3 is the following corollary, which follows from the exact $O(2^n n^2)$ time algorithm for n-vertex $\triangle$-ATSP by Bell [3], Held, and Karp [19]:

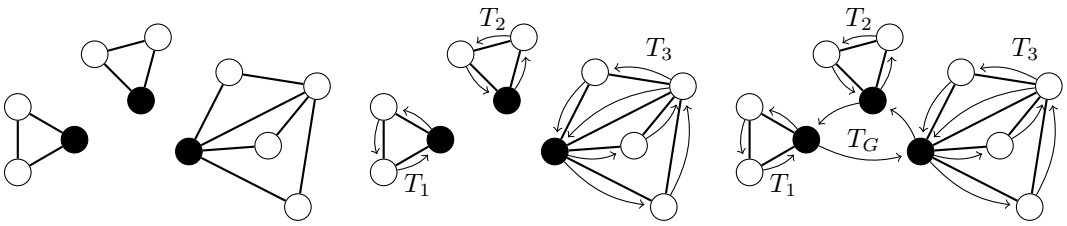
▶ **Corollary 4.** MWCARP *is constant-factor approximable in $O(2^C C^2+n^3\log n)$ time and thus in polynomial time for $C\in O(\log n)$.*

For perspective on Corollary 4, recall that finding a polynomial-time constant-factor approximation for MWCARP in general would, via Observation 2, answer a question open since 1982 [15]. Computing *optimal* solutions of MWCARP is NP-hard even if $C=1$ [5].

2 Preliminaries

Although DRPP, MWRPP, and MWCARP are problems on mixed graphs as defined in Definition 1, in some of our proofs we use more general mixed *multigraphs* $G=(V,E,A)$ with a set $V=:V(G)$ of *vertices*, a multiset $E=:E(G)$ over $\{\{u,v\}\mid u,v\in V\}$ of *(undirected) edges*, a multiset $A=:A(G)$ over $V\times V$ of *(directed) arcs* that may contain self-loops, and *travel costs* $c\colon V\times V\to\mathbb{N}\cup\{0,\infty\}$. If $E=\emptyset$, then G is a *directed multigraph*.

From Definition 1, recall the definition of walks in mixed graphs. An *Euler tour for G* is a closed walk that traverses each arc and each edge of G as often as it is present in G. A graph is *Eulerian* if it allows for an Euler tour. Let $w=(a_1,a_2,\ldots,a_\ell)$ be a walk. The *starting point of w* is the tail of a_1, the *end point of w* is the head of a_ℓ. A *segment* of w



(a) Input: Only required arcs R shown, vertices in V_R are black.

(b) Compute Euler tours T_i for each conn. component of $G[R]$.

(c) Add closed walk T_G to get a feasible solution T.

Figure 1 Steps of Algorithm 1 executed to construct a feasible solution for DRPP when all connected components of $G[R]$ are Eulerian.

is a consecutive subsequence of w. Two segments $w_1 = (a_i, \ldots, a_j)$ and $w_2 = (a_{i'}, \ldots, a_{j'})$ of a walk w are *non-overlapping* if $j < i'$ or $j' < i$. Note that two segments of w might be non-overlapping yet share arcs if w contains an arc several times. The *distance* $\mathrm{dist}_G(u, v)$ from vertex u to vertex v of G is the minimum cost of a walk from u to v in G.

The *underlying undirected (multi)graph* of G is obtained by replacing all directed arcs by undirected edges. Two vertices u, v of G are *(weakly) connected* if there is a walk starting in u and ending in v in the underlying undirected graph of G. A *(weakly) connected component* of G is a maximal subgraph of G in which all vertices are mutually (weakly) connected.

For a multiset $R \subseteq V \times V$ of arcs, $G[R]$ is the directed multigraph consisting of the arcs in R and their incident vertices of G. We say that $G[R]$ is the graph *induced by the arcs in* R. For a walk $w = (a_1, \ldots, a_\ell)$ in G, $G[w]$ is the directed multigraph consisting of the arcs $a_1, \ldots, a_\ell$ and their incident vertices, where $G[w]$ contains each arc with the multiplicity it occurs in w. Note that $G[R]$ and $G[w]$ might contain arcs with a higher multiplicity than G and, therefore, are not necessarily sub(multi)graphs of G. Finally, the cost of a multiset R is $c(R) := \sum_{a \in R} \nu(a)c(a)$, where $\nu(a)$ is the multiplicity of a in R.

3 Rural Postman

In this section, we present our approximation algorithms for DRPP and MWRPP, thus proving Theorem 3(i) and (ii). We first present, in Section 3.1, an algorithm for the special case of DRPP where the required arcs induce a subgraph with Eulerian connected components. Sections 3.2 and 3.3 subsequently generalize this algorithm to DRPP and MWRPP by adding to the set of required arcs an arc set of low weight so that the required arcs induce a graph with Eulerian connected components.

3.1 Special Case: Required arcs induce Eulerian components

To turn $\alpha(n)$-approximations for n-vertex $\triangle$-ATSP into $(\alpha(C) + 1)$-approximations for this special case of DRPP, we use Algorithm 1. Figure 1 illustrates its two main steps.

In fact, to solve this special case of DRPP, we will not exploit that Algorithm 1 and the following lemma allow R to be a *multi*set and that they allow V_R, the set of vertices incident with arcs of R, to contain more than one vertex of each connected component of $G[R]$. This will become relevant in Section 3.2, when we plug in Algorithm 1 to solve DRPP in general.

▶ **Lemma 5.** *Let G be a directed graph with travel costs c and R be a multiset of arcs of G such that $G[R]$ consists of C Eulerian connected components, let $V_R \subseteq V(G[R])$ be a vertex*

Algorithm 1: Algorithm for the proof of Lemma 5.

Input: A directed graph G with travel costs c, a multiset R of arcs of G such that $G[R]$ consists of C Eulerian connected components, and a set $V_R \subseteq V(G[R])$ containing at least one vertex of each connected component of $G[R]$.

Output: A closed walk traversing all arcs in R.

1 **for** $i = 1, \ldots, C$ **do**
2 $\quad v_i \leftarrow$ any vertex of V_R in component i of $G[R]$;
3 $\quad T_i \leftarrow$ Euler tour of connected component i of $G[R]$ starting and ending in v_i;
4 $(V_R, c') \leftarrow \triangle$-ATSP instance on the vertices V_R, where $c'(v_i, v_j) := \mathrm{dist}_G(v_i, v_j)$;
5 $T_{V_R} \leftarrow \alpha(|V_R|)$-approximate $\triangle$-ATSP solution for (V_R, c');
6 $T_G \leftarrow$ closed walk for G obtained by replacing each arc (v_i, v_j) on T_{V_R} by a shortest path from v_i to v_j in G;
7 $T \leftarrow$ closed walk obtained by following T_G and taking a detour T_i whenever reaching a vertex v_i;
8 **return** T;

set containing at least one vertex of each connected component of $G[R]$, and let $\tilde{T}$ be any closed walk containing the vertices V_R.

If n-vertex $\triangle$-ATSP is $\alpha(n)$-approximable in $t(n)$ time, then Algorithm 1 applied to (G, c, R) and V_R returns a closed walk of cost at most $c(R) + \alpha(|V_R|) \cdot c(\tilde{T})$ in $O(t(n) + n^3)$ time that traverses all arcs of R.

Proof. We first show that the closed walk T returned by Algorithm 1 visits all arcs in R. Since the $\triangle$-ATSP solution T_{V_R} constructed in line 5 visits all vertices V_R, in particular $v_1, \ldots, v_C$, so does the closed walk T_G constructed in line 6. Thus, for each vertex v_i, $1 \leq i \leq C$, T takes Euler tour T_i through the connected component i of $G[R]$ and, thus, visits all arcs in R.

We analyze the cost $c(T)$. The closed walk T is composed of the Euler tours T_i computed in line 3 and the closed walk T_G computed in line 6. Hence, $c(T) = c(T_G) + \sum_{i=1}^{C} c(T_i)$. Since each T_i is an Euler tour for some connected component i of $G[R]$, each T_i visits each arc of component i as often as it is contained in R. Consequently, $\sum_{i=1}^{C} c(T_i) = c(R)$.

It remains to analyze $c(T_G)$. Observe first that the distances in TSP instance (V_R, c') correspond to shortest paths in G and thus fulfill the triangle inequality. We have $c(T_G) = c'(T_{V_R})$ by construction of the $\triangle$-ATSP instance (V_R, c') in line 4 and by construction of T_G from T_{V_R} in line 6. Let $\tilde{T}$ be any closed walk containing V_R and let $T^*_{V_R}$ be an optimal solution for the $\triangle$-ATSP instance (V_R, c'). If we consider the closed walk $\tilde{T}_{V_R}$ that visits the vertices V_R of the $\triangle$-ATSP instance (V_R, c') in the same order as $\tilde{T}$, we get $c'(T^*_{V_R}) \leq c'(\tilde{T}_{V_R}) \leq c(\tilde{T})$. Since the closed walk T_{V_R} computed in line 5 is an $\alpha(|V_R|)$-approximate solution to the $\triangle$-ATSP instance (V_R, c'), it finally follows that $c(T_G) = c'(T_{V_R}) \leq \alpha(|V_R|) \cdot c'(T^*_{V_R}) \leq \alpha(|V_R|) \cdot c(\tilde{T})$.

Regarding the running time, observe that the instance (V_R, c') in line 4 can be constructed in $O(n^3)$ time using the Floyd-Warshall all-pair shortest path algorithm [11], which dominates all other steps of the algorithm except for, possibly, line 5. ◀

Lemma 5 proves Theorem 3(i) for DRPP instances $I = (G, c, R)$ when $G[R]$ consists of Eulerian connected components: pick V_R to contain exactly one vertex of each of the C connected components of $G[R]$. Since an optimal solution T^* for I visits the vertices V_R and satisfies $c(R) \leq c(T^*)$, Algorithm 1 yields a solution of cost at most $c(T^*) + \alpha(C) \cdot c(T^*)$.

Algorithm 2: Algorithm for the proof of Lemma 8.

Input: A DRPP instance $I = (G, c, R)$ such that $G[R]$ has C connected components and a set V_R of vertices, one of each connected component of $G[R]$.
Output: A feasible solution for I.

```
f ← minimum-cost flow for the UMCF instance (G, balance_{G[R]}, c);
foreach a ∈ A(G) do add arc a with multiplicity f(a) to (initially empty) multiset R*;
T ← closed walk computed by Algorithm 1 applied to (G, c, R ⊎ R*) and V_R;
return T;
```

3.2 Directed Rural Postman

In the previous section, we proved Theorem 3(i) for the special case of DRPP when $G[R]$ consists of Eulerian connected components. We will now reduce DRPP to this special case in order to prove Theorem 3(i) for the general DRPP. To this end, observe that a feasible solution T for a DRPP instance (G, c, R) enters each vertex v of G as often as it leaves. Thus, if we consider the multigraph $G[T]$ on the vertex set $V(G)$ that contains each arc of G with same multiplicity as T, then $G[T]$ is a supermultigraph of $G[R]$ in which every vertex is *balanced* [8, 24]:

▶ **Definition 6** (Balance)**.** By $\text{balance}_G(v) := \text{indeg}_G(v) - \text{outdeg}_G(v)$, we denote the *balance* of a vertex v of a graph G. We call a vertex v *balanced* if $\text{balance}_G(v) = 0$.

Since $G[T]$ is a supergraph of $G[R]$ in which all vertices are balanced and since a directed connected multigraph is Eulerian if and only if all its vertices are balanced, we immediately obtain the below observation. Herein and in the following, for two (multi-)sets X and Y, $X \uplus Y$ is the multiset obtained by adding the multiplicities of each element in X and Y.

▶ **Observation 7.** *Let T be a feasible solution for a* DRPP *instance (G, c, R) such that $G[R]$ has C connected components and let R^* be a minimum-cost multiset of arcs of G such that every vertex in $G[R \uplus R^*]$ is balanced. Then, $c(R \uplus R^*) \leq c(T)$ and $G[R \uplus R^*]$ consists of at most C Eulerian connected components.*

Algorithm 2 computes an $(\alpha(C) + 1)$-approximation for a DRPP instance (G, c, R) by first computing a minimum-cost arc multiset R^* such that $G[R \uplus R^*]$ contains only balanced vertices and then applying Algorithm 1 to $(G, c, R \uplus R^*)$. To find R^*, we use a folklore reduction [8, 10, 13] to the Uncapacitated Minimum-Cost Flow problem:

Uncapacitated Minimum-Cost Flow (UMCF)
Instance: A directed graph $G = (V, A)$ with *supply* $s\colon V \to \mathbb{Z}$ and *costs* $c\colon A \to \mathbb{N} \cup \{0\}$.
Task: Find a *flow* $f\colon A \to \mathbb{N} \cup \{0\}$ minimizing $\sum_{a \in A} c(a)f(a)$ such that, for each $v \in V$,

$$\sum_{(v,w) \in A} f(v, w) - \sum_{(w,v) \in A} f(w, v) = s(v). \tag{FC}$$

Equation (FC) is known as the *flow conservation* constraint: for every vertex v with $s(v) = 0$, there are as many units of flow entering the node as leaving it. Nodes v with $s(v) > 0$ "produce" $s(v)$ units of flow, whereas nodes v with $s(v) < 0$ "consume" $s(v)$ units of flow. UMCF is solvable in $O(n^3 \log n)$ time [1, Theorem 10.34].

▶ **Lemma 8.** *Let $I := (G, c, R)$ be a* DRPP *instance such that $G[R]$ has C connected components and let V_R be a vertex set containing exactly one vertex of each connected component of $G[R]$. Moreover, consider two closed walks in G:*

- *let $\tilde{T}$ be any closed walk containing the vertices V_R and*
- *let $\hat{T}$ be any feasible solution for I.*

If n-vertex $\triangle$-ATSP is $\alpha(n)$-approximable in $t(n)$ time, then Algorithm 2 applied to I and V_R returns a feasible solution of cost at most $c(\hat{T}) + \alpha(C) \cdot c(\tilde{T})$ in $O(t(n) + n^3 \log n)$ time.

Proof. Observe that Algorithm 2 in line 2 indeed computes a minimum-cost arc set R^* such that all vertices in $G[R \uplus R^*]$ are balanced (we provide details in Appendix A).

We use the optimality of R^* to give an upper bound on the cost of the closed walk T computed in line 3. Since V_R contains exactly one vertex of each connected component of $G[R]$, it contains at least one vertex of each connected component of $G[R \uplus R^*]$. Therefore, Algorithm 1 is applicable to $(G, c, R \uplus R^*)$ and, by Lemma 5, yields a closed walk in G traversing all arcs in $R \uplus R^*$ and having cost at most $c(R \uplus R^*) + \alpha(|V_R|) \cdot c(\tilde{T})$. This is a feasible solution for (G, c, R) and, since by Observation 7, we have $c(R \uplus R^*) \leq c(\hat{T})$, it follows that this feasible solution has cost at most $c(\hat{T}) + \alpha(C) \cdot c(\tilde{T})$.

Finally, the running time of Algorithm 2 follows from the fact that the minimum-cost flow in line 1 is computable in $O(n^3 \log n)$ time [1, Theorem 10.34] and that Algorithm 1 runs in $O(n^3 + t(C))$ time (Lemma 5). ◀

Proof of Theorem 3(i). Let (G, c, R) be an instance of DRPP and let V_R be a set of vertices containing exactly one vertex of each connected component of $G[R]$. An optimal solution T^* for I contains all arcs in R and all vertices in V_R and hence, by Lemma 8, Algorithm 2 computes a feasible solution T with $c(T) \leq c(T^*) + \alpha(C) \cdot c(T^*)$ for I. ◀

3.3 Mixed and Windy Rural Postman

In the previous section, we presented Algorithm 2 for DRPP in order to prove Theorem 3(i). We now show how to apply Algorithm 2 to MWRPP in order to prove Theorem 3(ii).

To this end, we replace each undirected edge $\{u, v\}$ in an MWRPP instance by two directed arcs (u, v) and (v, u), where we force the undirected *required* edges of the MWRPP instance to be traversed in the cheaper direction:

▶ **Lemma 9.** *Let $I := (G, c, R)$ be an* MWRPP *instance and let $I' := (G', c, R')$ be the* DRPP *instance obtained from I as follows:*

- *G' is obtained by replacing each edge $\{u, v\}$ of G by two arcs (u, v) and (v, u),*
- *R' is obtained from R by replacing each edge $\{u, v\} \in R$ by an arc (u, v) if $c(u, v) \leq c(v, u)$ and by (v, u) otherwise.*

Then, each feasible solution for I' is a feasible solution of the same cost for I and, for each feasible solution T for I, there is a feasible solution T' for I' with $c(T') < 3c(T)$.

We prove Lemma 9 in Appendix B. Using Lemma 9, it is easy to prove Theorem 3(ii).

Proof of Theorem 3(ii). Given an MWRPP instance $I = (G, c, R)$, compute a DRPP instance $I' := (G', c, R')$ as described in Lemma 9. This can be done in linear time.

Let V_R be a set of vertices containing exactly one vertex of each connected component of $G'[R']$ and let T^* be an optimal solution for I. Observe that T^* is not necessarily a feasible solution for I', since it might serve required arcs of I' in the wrong direction. Yet T^* is a closed walk in G' visiting all vertices of V_R. Moreover, by Lemma 9, I' has a feasible solution T' with $c(T') \leq 3c(T^*)$.

Thus, applying Algorithm 2 to I' and V_R yields a feasible solution T of cost at most $c(T') + \alpha(C) \cdot c(T^*) \leq 3c(T^*) + \alpha(C) \cdot c(T^*)$ due to Lemma 8. Finally, T is also a feasible solution for I by Lemma 9. ◀

Algorithm 3: Algorithm for the proof of Proposition 12.

Input: An MWCARP instance $I = (G, v_0, c, d, Q)$ such that $(v_0, v_0) \in R_d$ and such that $G[R_d]$ has C connected components.
Output: A feasible solution for I.

```
/* Compute a base tour containing all demand arcs and the depot      */
```
$I' \leftarrow$ MWRPP instance $I' := (G, c, R_d)$;
$T \leftarrow \beta(C)$-approximate MWRPP tour for I' starting and ending in v_0;
```
/* Split the base tour into one tour for each vehicle                */
```
$(W, s) \leftarrow$ a feasible splitting of T;
foreach $w \in W$ **do**
    close w by adding shortest paths from v_0 to s and from t to v_0 in G, where s, t are the start and endpoints of w, respectively;
return (W, s);

4 Capacitated Arc Routing

Our approximation algorithm for MWCARP uses the fact that joining all vehicle tours of a solution gives an MWRPP tour traversing all positive-demand arcs and the depot. Thus, in order to approximate MWCARP, the idea is to first compute an approximate MWRPP tour and then split it into subtours, each of which can be served by a vehicle of capacity Q. Then we close each subtour by shortest paths via the depot. This algorithm is inspired by the CARP algorithms of Jansen [20] and Wøhlk [25] and the algorithm of Frederickson, Hecht, and Kim [14] for (undirected) k-person minimax routing problems. Our analysis, however, is necessarily different, since we cannot use arcs and edges in backwards direction.

▶ **Definition 10** (Demand arc). For a demand function $d\colon E(G) \cup A(G) \to \mathbb{N} \cup \{0\}$ we define $R_d := \{a \in E(G) \cup A(G) \mid d(a) > 0\}$ to be the set of *demand arcs*.

We will construct an MWCARP solution from a *feasible splitting* of an MWRPP tour T.

▶ **Definition 11** (Feasible splitting). For an MWCARP instance $I = (G, v_0, c, d, Q)$, let T be a closed walk containing all arcs in R_d and $W = (w_1, \dots, w_\ell)$ be a tuple of segments of T. In the following, we refer by W to both the tuple and the set of walks it contains.

Furthermore, consider a serving function $s\colon W \to 2^{R_d}$ that assigns to each walk the set of arcs in R_d it serves. We call (W, s) a *feasible splitting of* T if the following conditions hold:

1. the walks in W are mutually non-overlapping segments of T,
2. when concatenating the walks in W in order, one obtains a subsequence of T,
3. each $w_i \in W$ begins and ends with an arc in $s(w_i)$,
4. $\{s(w_i) \mid w_i \in W\}$ is a partition of R_d, and
5. for each $w_i \in W$, we have $\sum_{e \in s(w_i)} d(e) \leq Q$ and, if $i < \ell$, then $\sum_{e \in s(w_i)} d(e) + d(a) > Q$, where a is the first arc served by w_{i+1}.

A feasible splitting of a given closed walk T as above can be computed in linear time using a greedy strategy (we refer to Appendix C for details).

The algorithm. Algorithm 3 constructs an MWCARP solution from an approximate MWRPP solution T containing all demand arcs and the depot v_0. In order to ensure that T contains v_0, Algorithm 3 assumes that the input graph has a demand loop (v_0, v_0): if this loop is not present, one can add it with zero cost. Note that, while this does not change

the cost of an optimal solution, it might increase the number of connected components in the subgraph induced by demand arcs by one. To compute an MWCARP solution from T, Algorithm 3 first computes a feasible splitting (W, s) of T. To each walk $w_i \in W$, it then adds a shortest path from the end of w_i to the start of w_i via the depot. It is not hard to check that Algorithm 3 indeed outputs a feasible solution by using the properties of feasible splittings and the fact that T contains all demand arcs. The remainder of this section is devoted to the analysis of the solution, thus proving the following proposition and, consequently, Theorem 3(iii).

▶ **Proposition 12.** *Let $I = (G, v_0, c, d, Q)$ be an* MWCARP *instance and let I' be the instance obtained from I by adding a zero-cost demand arc (v_0, v_0) if it is not present.*

If MWRPP *is $\beta(C)$-approximable in $t(n)$ time, then Algorithm 3 applied to I' computes a $(8\beta(C+1)+3)$-approximation for I in $O(t(n)+n^3)$ time. Herein, C is the number of connected components in $G[R_d]$.*

The following lemma follows from the observation that the concatenation of all vehicle tours in any MWCARP solution yields an MWRPP tour containing all demand arcs and the depot. It is proven in Appendix D.

▶ **Lemma 13.** *Let $I = (G, v_0, c, d, Q)$ be an* MWCARP *instance with $(v_0, v_0) \in R_d$ and an optimal solution (W^*, s^*). The closed walk T and its feasible splitting (W, s) computed in lines 2 and 3 of Algorithm 3 satisfy $c(W) \leq c(T) \leq \beta(C)c(W^*)$, where C is the number of connected components in $G[R_d]$.*

It remains to analyze the length of the shortest paths from v_0 to $w_i \in W$ and from w_i to v_0 added in line 5 of Algorithm 3. We bound their lengths in the lengths of an auxiliary walk $A(w_i)$ from v_0 to w_i and of an auxiliary walk $Z(w_i)$ from w_i to v_0. The auxiliary walks $A(w_i)$ and $Z(w_i)$ consist of arcs of W, whose total cost is bounded by Lemma 13, and of arcs of an optimal solution (W^*, s^*). We show that, in total, the walks $A(w_i)$ and $Z(w_i)$ for *all* $w_i \in W$ use each subwalk of W and W^* at most a constant number of times. For this, we group the walks in W into consecutive pairs, for each of which we will be able to charge the cost of the auxiliary walks to a distinct vehicle tour of the optimal solution.

▶ **Definition 14** (Consecutive pairing)**.** For a feasible splitting (W, s) with $W = (w_1, \ldots, w_\ell)$, we call $W^2 := \{(w_{2i-1}, w_{2i}) \mid i \in \{1, \ldots, \lfloor \ell/2 \rfloor\}\}$ a *consecutive pairing*.

We can now show, by applying Hall's theorem [18], that each pair traverses an arc from a *distinct* tour of an optimal solution (Appendix E).

▶ **Lemma 15.** *Let $I = (G, v_0, c, d, Q)$ be an* MWCARP *instance with an optimal solution (W^*, s^*) and let W^2 be a consecutive pairing of some feasible splitting (W, s). Then, there is an injective map $\phi\colon W^2 \to W^*, (w_i, w_{i+1}) \mapsto w^*$ such that $(s(w_i) \cup s(w_{i+1})) \cap s^*(w^*) \neq \emptyset$.*

In the following, we fix an arbitrary arc in $(s(w_i) \cup s(w_{i+1})) \cap s^*(w^*)$ and call it the *pivot arc* of (w_i, w_{i+1}). Informally, the auxiliary walks for each w_i are constructed as follows. To get from the endpoint of w_i to v_0, walk along the closed walk T until traversing the first pivot arc a. To get from the head of a to v_0, walk along w^*, which is the walk of W^* containing a. To get from v_0 to w_i, take the same approach, that is, walk backwards on T from the start point of w_i until traversing a pivot arc and then follow the tour of W^* containing a. The formal definition of the auxiliary walks $A(w)$ and $Z(w)$ is given below (see also Figure 2).

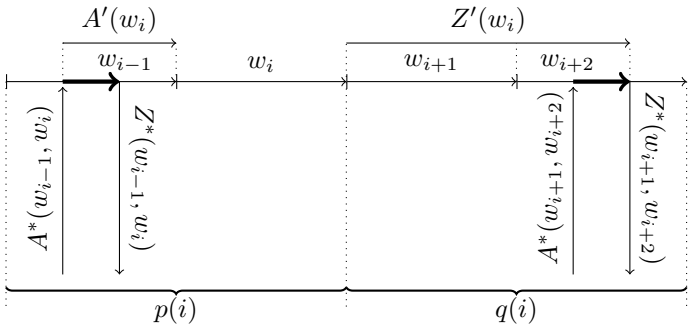


Figure 2 Illustration of Definition 16. Dotted lines are ancillary lines. Thin arrows are walks. The braces along the bottom show a consecutive pairing of walks $w_{i-1}, \dots, w_{i+2}$. Bold arcs are pivot arcs. Here, $p(i)$ is exactly the pair that contains w_i and $q(i)$ is the next pair.

▶ **Definition 16** (Auxiliary walks). Let $I = (G, v_0, c, d, Q)$ be an MWCARP instance, (W^*, s^*) be an optimal solution, and W^2 be a consecutive pairing of some feasible splitting (W, s) of a closed walk T containing all arcs R_d and v_0, where $W = (w_1, \dots, w_\ell)$.

Let $\phi\colon W^2 \to W^*$ be an injective map as in Lemma 15 and for each pair $(w, w') \in W^2$ let
$\boldsymbol{A^*(w_i, w_{i+1})}$ be a subwalk of $\phi(w_i, w_{i+1})$ from v_0 to the tail of the pivot arc of (w_i, w_{i+1}),
$\boldsymbol{Z^*(w_i, w_{i+1})}$ be a subwalk of $\phi(w_i, w_{i+1})$ from the head of the pivot arc of (w_i, w_{i+1}) to v_0.
For each walk $w_i \in W$ with $i \geq 3$ (that is, w_i is not in the first pair of W^2), let
$\boldsymbol{p(i)}$ be the index of the pair whose pivot arc is traversed first when walking T backwards starting from the starting point of w_i,
$\boldsymbol{A'(w_i)}$ be the subwalk of T starting at the end point of $A^*(w_{2p(i)}, w_{2p(i)+1})$ and ending at the start point of w_i, and
$\boldsymbol{A(w_i)}$ be the walk from v_0 to the start point of w_i following first $A^*(w_{2p(i)}, w_{2p(i)+1})$ and then $A'(w_i)$.
For each walk $w_i \in W$ with $i \leq \ell - 3$ (that is, w_i is not in the last pair of W^2, where w_ℓ might not be in any pair if ℓ is odd), let
$\boldsymbol{q(i)}$ be the index of the pair whose pivot arc is traversed first when following T starting from the end point of w_i,
$\boldsymbol{Z'(w_i)}$ be the subwalk of T starting at the end point of w_i and ending at the start point of $Z^*(w_{2q(i)}, w_{2q(i)+1})$, and, finally, let
$\boldsymbol{Z(w_i)}$ be the walk from the end point of w_i to v_0 following $Z'(w_i)$ and $Z^*(w_{2q(i)}, w_{2q(i)+1})$.
We are now ready to prove Proposition 12, which also concludes our proof of Theorem 3.

Proof of Proposition 12. Let $I = (G, v_0, c, d, Q)$ be an MWRPP instance and (W^*, s^*) be an optimal solution. If there is no demand arc (v_0, v_0) in I, then we add it with zero cost in order to make Algorithm 3 applicable. This clearly does not change the cost of an optimal solution but may increase the number of connected components of $G[R_d]$ to $C + 1$.

In lines 2 and 3, Algorithm 3 computes a tour T and its feasible splitting (W, s). Denote $W = (w_1, \dots, w_\ell)$. The solution returned by Algorithm 3 consists, for each $1 \leq i \leq \ell$, of a tour starting in v_0, following a shortest path to the starting point of w_i, then w_i, and a shortest path back to v_0.

For $i \geq 3$, the shortest path from v_0 to the starting point of w_i has length at most $c(A(w_i))$. For $i \leq \ell - 3$, the shortest path from the end point of w_i to v_0 has length at most $c(Z(w_i))$. This amounts to $\sum_{i=3}^{\ell} c(A(w_i)) + \sum_{i=1}^{\ell-3} c(Z(w_i))$. To bound the costs of the shortest paths added for $i \in \{1, 2, \ell-2, \ell-1, \ell\}$, observe the following. For each $i \in \{1, 2\}$, the shortest paths

from v_0 to the start point of w_i and from the end point of $w_{\ell-i}$ to v_0 together have length at most $c(T)$. The shortest path from the end point of w_ℓ to v_0 has length at most $c(T) - c(W)$. Thus, the solution returned by Algorithm 3 has cost at most

$$\begin{aligned}
&\sum_{i=1}^{\ell} c(w_i) + \sum_{i=3}^{\ell} c(A(w_i)) + \sum_{i=1}^{\ell-3} c(Z(w_i)) + 3c(T) - c(W) \\
&= \sum_{i=3}^{\ell} c(A(w_i)) + \sum_{i=1}^{\ell-3} c(Z(w_i)) + 3c(T) \\
&= 3c(T) + \\
&\quad + \sum_{i=3}^{\ell} c(A^*(w_{2p(i)}, w_{2p(i)+1})) + \sum_{i=1}^{\ell-3} c(Z^*(w_{2q(i)}, w_{2q(i)+1})) + \qquad \text{(S1)} \\
&\quad + \sum_{i=3}^{\ell} c(A'(w_i)) + \sum_{i=1}^{\ell-3} c(Z'(w_i)). \qquad \text{(S2)}
\end{aligned}$$

Observe that, for a fixed i, one has $p(i) = p(j)$ only for $j \leq i + 2$ and $q(i) = q(j)$ only for $j \geq i - 2$. Moreover, by Lemma 15 and Definition 16, for $i \neq j$, $A^*(w_i, w_{i+1})$ and $A^*(w_j, w_{j+1})$ are subwalks of distinct walks of W^*. Similarly, $Z^*(w_i, w_{i+1})$ and $Z^*(w_j, w_{j+1})$ are subwalks of distinct walks of W^* if $i \neq j$. Hence, sum (S1) counts every arc of W^* at most three times and is therefore bounded from above by $3c(W^*)$. Moreover, for a walk w_i, let $\mathcal{A}_i$ be the set of walks w_j such that any arc a of w_i is contained in $A'(w_j)$ and let $\mathcal{Z}_i$ be the set of walks such that any arc a of w_i is contained in $Z'(w_j)$. Observe that $A'(w_j)$ and $Z'(w_j)$ cannot completely contain two walks of the same pair of the consecutive pairing W^2 of W since, by Lemma 15, each pair has a pivot arc and $A'(w_j)$ and $Z'(w_j)$ both stop after traversing a pivot arc. Hence, the walks in $\mathcal{A}_i \cup \mathcal{Z}_i$ can be from at most three pairs of W^2: the pair containing w_i and the two neighboring pairs. Finally, observe that w_i itself is not contained in $\mathcal{A}_i \cup \mathcal{Z}_i$. Thus, $\mathcal{A}_i \cup \mathcal{Z}_i$ contains at most five walks (Figure 3 in the appendix shows such a worst-case example). Therefore, sum (S2) counts every arc of W at most five times and is bounded from above by $5c(W)$.

Thus, Algorithm 3 returns a solution of cost $3c(T)+5c(W)+3c(W^*)$ which, by Lemma 13, is at most $8c(T) + 3c(W^*) \leq 8\beta(C+1)c(W^*) + 3c(W^*) \leq (8\beta(C+1)+3)c(W^*)$. ◀

5 Conclusion

With the exception of MWCARP, we expect our algorithms to yield good heuristics. In particular, the $\triangle$-ATSP instances should be sufficiently small to allow for the computation of optimal solutions. For MWCARP, a better approach than the presented one could be to compute an MWRPP tour and then compute an *optimal* splitting of this tour into vehicle tours. Our analysis gives a worst-case bound for this approach. We conclude with an open question: can the $(\alpha(C) + 3)$-approximation for MWRPP in Theorem 3(ii) be improved to an $(\alpha(C) + 3/2)$-approximation using the 3/2-approximation for Mixed Chinese Postman given by Raghavachari and Veerasamy [22]?

Acknowledgments. We thank Sepp Hartung, Iyad Kanj, and André Nichterlein for fruitful discussions. This research was initiated during a research retreat of the algorithms and complexity group of TU Berlin, held in Rothenburg/Oberlausitz, Germany, in March 2015, while René van Bevern was with TU Berlin under support of the DFG, project DAPA (NI 369/12). Manuel Sorge was supported by the DFG, project DAPA (NI 369/12).

References

1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows—Theory, Algorithms and Applications*. Prentice Hall, 1993.

2 Arash Asadpour, Michel X. Goemans, Aleksander Mądry, Shayan Oveis Gharan, and Amin Saberi. An $O(\log n/\log\log n)$-approximation algorithm for the asymmetric traveling salesman problem. In *Proc. SODA'10*, pages 379–389. SIAM, 2010.

3 Richard Bellman. Dynamic programming treatment of the Travelling Salesman Problem. *J. ACM*, 9(1):61–63, 1962.

4 René van Bevern, Sepp Hartung, André Nichterlein, and Manuel Sorge. Constant-factor approximations for capacitated arc routing without triangle inequality. *Oper. Res. Lett.*, 42(4):290–292, 2014.

5 René van Bevern, Rolf Niedermeier, Manuel Sorge, and Mathias Weller. Complexity of arc routing problems. In *Arc Routing: Problems, Methods, and Applications*. SIAM, 2014.

6 Nicos Christofides. Worst case analysis of a new heuristic for the traveling salesman problem. Management Science Research Rept. 388, Carnegie-Mellon University, 1976.

7 Ángel Corberán and Gilbert Laporte, editors. *Arc Routing: Problems, Methods, and Applications*. SIAM, 2014.

8 Frederic Dorn, Hannes Moser, Rolf Niedermeier, and Mathias Weller. Efficient algorithms for Eulerian Extension and Rural Postman. *SIAM J. Discrete Math.*, 27(1):75–94, 2013.

9 Jack Edmonds. The Chinese postman problem. *Oper. Res.*, pages B73–B77, 1975. Suppl. 1.

10 Jack Edmonds and Ellis L. Johnson. Matching, Euler tours and the Chinese postman. *Math. Program.*, 5:88–124, 1973.

11 Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, June 1962.

12 Greg N. Frederickson. *Approximation Algorithms for NP-hard Routing Problems*. PhD thesis, Faculty of the Graduate School of the University of Maryland, 1977.

13 Greg N. Frederickson. Approximation algorithms for some postman problems. *J. ACM*, 26(3):538–554, 1979.

14 Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation algorithms for some routing problems. *SIAM J. Comput.*, 7(2):178–193, 1978.

15 A. M. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12(1):23–39, 1982.

16 Bruce L. Golden and Richard T. Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.

17 Gregory Gutin, Magnus Wahlström, and Anders Yeo. Parameterized Rural Postman and Conjoining Bipartite Matching problems. Available on arXiv:1308.2599v4, 2014.

18 P. Hall. On representatives of subsets. *J. London Math. Soc.*, 10:26–30, 1935.

19 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *J. SIAM*, 10(1):196–210, 1962.

20 Klaus Jansen. An approximation algorithm for the general routing problem. *Inform. Process. Lett.*, 41(6):333–339, 1992.

21 J. K. Lenstra and A. H. G. Rinnooy Kan. On general routing problems. *Networks*, 6(3):273–280, 1976.

22 Balaji Raghavachari and Jeyakesavan Veerasamy. A 3/2-approximation algorithm for the Mixed Postman Problem. *SIAM J. Discrete Math.*, 12(4):425–433, 1999.

23 A. I. Serdyukov. O nekotorykh ekstremal'nykh obkhodakh v grafakh. *Upravlyayemyye sistemy*, 17:76–79, 1978. [On some extremal by-passes in graphs. *zbMATH 0475.90080*].

24 Manuel Sorge, René van Bevern, Rolf Niedermeier, and Mathias Weller. A new view on Rural Postman based on Eulerian Extension and Matching. *J. Discrete Alg.*, 16:12–33, 2012.

25 Sanne Wøhlk. An approximation algorithm for the Capacitated Arc Routing Problem. *The Open Operational Research Journal*, 2:8–12, 2008.

A Omitted details in the proof of Lemma 8

Proof. To complete the proof of Theorem 8, we prove that Algorithm 2 in line 2 indeed computes a minimum-cost arc set R^* such that all vertices in $G[R \uplus R^*]$ are balanced. This follows from the one-to-one correspondence between arc multisets R' such that $G[R \uplus R']$ has only balanced vertices and flows f for the UMCF instance $I' := (G, \text{balance}_{G[R]}, c)$:

1. For each vertex v with $\text{balance}_{G[R]}(v) = 0$, R' has to contain as many incident in-arcs as out-arcs so that $\text{balance}_{G[R \uplus R']}(v) = 0$. Likewise, by (FC), in any feasible flow for I', as many units of flow enter v as leave v.
2. Each vertex v with $\text{balance}_{G[R]}(v) > 0$ has $\text{balance}_{G[R]}(v)$ more incident in-arcs than out-arcs in $G[R]$ and, thus, in order for $\text{balance}_{G[R \uplus R']}(v) = 0$ to hold, R' has to contain $\text{balance}_{G[R]}(v)$ more out-arcs than in-arcs incident to v. Likewise, by (FC), in any feasible flow for I', there are $\text{balance}_{G[R]}(v)$ more units of flow leaving v than entering v.
3. For each vertex v with $\text{balance}_{G[R]}(v) < 0$, analogous arguments apply.

Thus, from a multiset R' of arcs such that $G[R \uplus R']$ is balanced, we get a feasible flow f for I' by setting $f(v, w)$ to the multiplicity of the arc (v, w) in R'. From a feasible flow f for I', we get a multiset R' of arcs such that $G[R \uplus R']$ is balanced by adding to R' each arc (v, w) with multiplicity $f(v, w)$. We conclude that the arc multiset R^* computed in line 2 is such that $G[R \uplus R^*]$ is balanced. Moreover, it is a minimum-cost such set, since a set of lower cost would yield a flow cheaper than the optimum flow f computed in line 1. ◀

B Proof of Lemma 9

Proof. It is obvious that each feasible solution T' for I' is a feasible solution for I, since each required edge of I is served by T' in at least one direction. Moreover, the cost functions in I and I' are the same.

Now, for the opposite direction, let T be a feasible solution for I. We obtain a feasible solution T' of I' as follows:

1. For each arc (u, v) or non-required edge $\{u, v\}$ traversed by T in direction from u to v, T' traverses arc (u, v).
2. For each required edge $\{u, v\}$ traversed by T from u to v such that $c(u, v) \leq c(v, u)$, T' traverses (u, v).
3. For each required edge $\{u, v\}$ traversed by T from u to v such that $c(u, v) > c(v, u)$, T' traverses $(u, v), (v, u)$, and again (u, v).

The closed walk T' is indeed a feasible solution to I': in (2), note that $(u, v) \in R'$ and that it is served by T'. In (3), in contrast, $(v, u) \in R'$, which is also served by T'.

To compute the cost of T', observe that only (3) increases the cost of T' compared to T: instead of $c(u, v)$, which is paid by T for traversing $\{u, v\}$ in the direction from u to v, the closed walk T' pays $c(u, v) + c(v, u) + c(u, v) < 3c(u, v)$ since $c(v, u) < c(u, v)$. ◀

C Obtaining feasible splittings

Given an MWCARP instance $I = (G, v_0, c, d, Q)$, a feasible splitting (W, s) of a closed walk T that traverses all arcs in R_d can be computed in linear time as follows. We assume that each arc has demand at most Q since otherwise I has no feasible solution. Now, traverse T, successively defining subwalks $w \in W$ and the corresponding sets $s(w)$ one at a time. The traversal starts with the first arc $a \in R_d$ of T and by creating a subwalk w consisting only of a and $s(w) = \{a\}$. On discovery of a still unserved arc $a \in R_d \setminus (\bigcup_{w' \in W} s(w'))$ do the

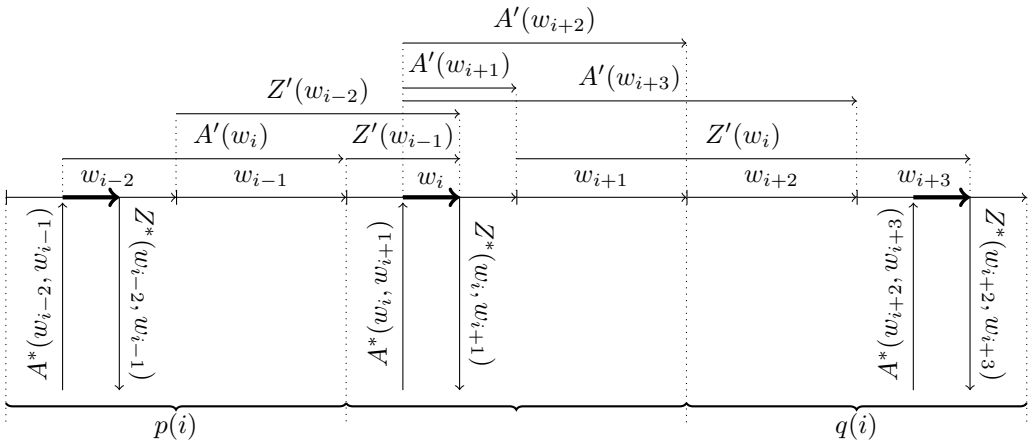


Figure 3 Illustration of the situation in which a maximum number of five different walks in W traverse the same pivot arc (the bold arc of w_i) in their respective auxiliary walks.

following. If $\sum_{e \in s(w)} d(e) + d(a) \leq Q$, then add a to $s(w)$ and append to w the subwalk of T that was traversed since discovery of the previous unserved arc in R_d. Otherwise, mark w and $s(w)$ as finished, start a new tour $w \in W$ with a as the first arc, set $s(w) = \{a\}$, and continue the traversal of T. If no arc a is found, then stop. It is not hard to verify that indeed, (W, s) is a feasible splitting.

D Proof of Lemma 13

Proof. Consider an optimal solution (W^*, s^*) to I. The closed walks in W^* visit all arcs in R_d. Concatenating them to a closed walk T^* gives a feasible solution for the MWRPP instance $I' = (G, c, R_d)$ in line 1 of Algorithm 3. Moreover, $c(T^*) = c(W^*)$. Thus, we have $c(T) \leq \beta(C)c'(T^*)$ in line 2. Moreover, by Condition 1 of Definition 11, one has $c(W) \leq c(T)$. This finally implies $c(W) \leq c(T) \leq \beta(C)c(T^*) = \beta(C)c(W^*)$ in line 3. ◀

E Proof of Lemma 15

Proof. Define an undirected bipartite graph B with the partite sets W^2 and W^*. A pair $(w, w') \in W^2$ and a closed walk $w^* \in W^*$ are adjacent in B if $(s(w) \cup s(w')) \cap s^*(w^*) \neq \emptyset$. We prove that B allows for a matching that matches each vertex of W^2 to some vertex in W^*. To this end, by Hall's theorem [18], it suffices to prove that, for all subsets $S \subseteq W^2$, it holds that $|N_B(S)| \geq |S|$, where $N_B(S) := \bigcup_{v \in S} N_B(v)$ and $N_B(v)$ is the set of neighbors of a vertex v in B. Observe that, by Condition 5 of Definition 11 of feasible splittings, for each pair $(w, w') \in W^2$ we have $d(s(w) \cup s(w')) \geq Q$. Since the pairs serve pairwise disjoint sets of demand arcs (Condition 4 of feasible splittings), the pairs in S serve a total demand of at least $Q \cdot |S|$ in the closed walks $N_B(S) \subseteq W^*$. Since each closed walk in $N_B(S)$ serves demand at most Q, the set $N_B(S)$ is at least as large as S, as required. ◀