Tropical Neighbourhood Search: A New Heuristic for Periodic Timetabling

Enrico Bortoletto
Zuse Institute Berlin, Germany

Niels Lindner
Zuse Institute Berlin, Germany

Berenike Masing
Zuse Institute Berlin, Germany

Abstract

Periodic timetabling is a central aspect of both the long-term organization and the day-to-day operations of a public transportation system. The Periodic Event Scheduling Problem (PESP), the combinatorial optimization problem that forms the mathematical basis of periodic timetabling, is an extremely hard problem, for which optimal solutions are hardly ever found in practice. The most prominent solving strategies today are based on mixed-integer programming, and there is a concurrent PESP solver employing a wide range of heuristics [3]. We present tropical neighborhood search (`tns`), a novel PESP heuristic. The method is based on the relations between periodic timetabling and tropical geometry [4]. We implement `tns` into the concurrent solver, and test it on instances of the benchmarking library `PESPlib`. The inclusion of `tns` turns out to be quite beneficial to the solver: `tns` is able to escape local optima for the modulo network simplex algorithm, and the overall share of improvement coming from `tns` is substantial compared to the other methods available in the solver. Finally, we provide better primal bounds for five `PESPlib` instances.

2012 ACM Subject Classification Applied computing → Transportation; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Network flows; Mathematics of computing → Solvers; Mathematics of computing → Integer programming; Computing methodologies → Concurrent algorithms

Keywords and phrases Periodic Timetabling, Tropical Geometry, Neighborhood Search, Mixed-Integer Programming

Digital Object Identifier 10.4230/OASIcs.ATMOS.2022.3

Funding *Enrico Bortoletto*: Funded within the Research Campus MODAL, funded by the German Federal Ministry of Education and Research (BMBF) (fund number 05M20ZBM).
Berenike Masing: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID: 390685689).

1 Introduction

Rhythm is to music what the timetable is for a public transit system. Periodicity of transportation networks is a common characteristic, quite useful in practice, and so periodic timetables are of particular importance. Setting up departure and arrival times in a feasible way is quite complicated, and the standard framework to model and optimize such timetables is that of the *Periodic Event Scheduling Problem* (PESP), first devised by Serafini and Ukovich [29]. Other than public transportation, PESP is also useful in automated production systems [11], and more generally in any case where periodicity constraints are in effect. Deciding whether a PESP instance is feasible is known to be NP-hard for any fixed period time $T \geq 3$ [23, 26], or when the underlying graph is series-parallel [20].

22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).
Editors: Mattia D'Emidio and Niels Lindner; Article No. 3; pp. 3:1–3:19
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Other than a basic MIP formulation, in practice there have been many attempts to tackle the problem by a plethora of techniques [6, 7, 8, 9, 18, 19, 21, 22, 25, 27]. However, the most successful method in practice remains the concurrent solver of Borndörfer, Lindner, and Roth [3], which, in parallel, implements MIP-based branch-and-cut [15], the modulo network simplex algorithm (`mns`, [8, 25]) as a local improvement heuristic, and a maximum-cut based heuristic [18], together with other features.

In this paper we introduce a novel heuristic, called *Tropical Neighbourhood Search* (`tns`). The `tns` algorithm is based on the link between the space of feasible periodic timetables and tropical geometry established in [4]. We will recall useful theoretical results in Section 2, which provide geometrical insight to the algorithm then described in Section 3. Finally, we will describe our implementation of `tns` in Section 4 and evaluate it in Section 5 on a subset of the instances of the benchmarking library `PESPlib` [5]. We conclude the paper with an outlook in Section 6.

2 Tropical Decomposition of Periodic Timetable Space

The goal of periodic timetabling in public transport is to assign timestamps to departure and arrival events of, e.g., trains at stations, so that the time between such events is within some given bounds. By the periodic nature, it suffices to consider timestamps modulo a period time T. The standard mathematical model for periodic timetabling is the *Periodic Event Scheduling Problem (PESP)* [29]. A PESP instance is comprised of a tuple (G, T, ℓ, u, w), whose elements are:

- An *event-activity network* G, a directed graph whose vertices $V(G)$ represent *events* in the network, and whose arcs $A(G)$ represent *activities* between events. In the context of periodic timetabling, these events describe the points in time of departures or arrivals, while the activities model the time durations of driving between stations, dwelling at a station, transferring between lines, turning at terminal stations, or fixing headways [16]. We will assume that G is simple and weakly connected, which is no restriction [15].
- A *period time* $T \in \mathbb{N}$, indicating after what time an event should occur again.
- Vectors $\ell, u \in \mathbb{R}^{A(G)}$ of *lower* and *upper bounds* on the activities, such that $0 \le \ell_a < T$ and $0 \le u_a - \ell_a < T$, indicating minimum and maximum durations of $a \in A(G)$.
- A vector $w \in \mathbb{R}^{A(G)}$ of *weights*, often modelling an ascribed importance to a given activity, for example represented by the number of passengers partaking in said activity.

The variables to determine are the *periodic timetable*, a vector $\pi \in \mathbb{R}^{V(G)}$, and the *periodic tension*, a vector $x \in \mathbb{R}^{A(G)}$. A pair (π, x) of timetable and tension is said to be *feasible* if

$$\forall (i,j) \in A(G): \qquad \pi_j - \pi_i \equiv x_{ij} \bmod T \qquad \text{and} \qquad \ell_{ij} \le x_{ij} \le u_{ij}, \tag{1}$$

where the first constraint models the periodicity property, while the second ensures that the tension is within the given bounds. Note that due to $0 \le u_a - \ell_a \le T$ for all $a \in A(G)$, for any given π there is at most one x such that (π, x) is feasible. In this case, we will hence speak of *the* tension associated to a timetable.

Given an appropriate tuple (G, T, ℓ, u, w), PESP consists in finding a feasible pair (π, x) such that the weighted tension $w^\top x$ is minimized. If ℓ and u are integral, which is true for most practical purposes, by a result of [26] the feasibility of the instance implies the existence of an integral optimal solution.

PESP can be formulated as a mixed-integer program by employing some auxiliary integer variables $p \in \mathbb{Z}^{A(G)}$ to model the modulo constraints by $\pi_j - \pi_i + Tp_{ij} = x_{ij}$ for all $(i,j) \in A(G)$. Then, using the incidence matrix $B \in \{-1, 0, 1\}^{V(G) \times A(G)}$ of G, the problem is as follows:

$$\begin{aligned} \text{Minimise} \quad & w^\top x \\ \text{subject to} \quad & -B^\top \pi + Tp = x \\ & \ell \leq x \leq u, \\ & p \in \mathbb{Z}^{A(G)}, \pi \in \mathbb{R}^{V(G)}, x \in \mathbb{R}^{A(G)} \end{aligned} \tag{2}$$

where each p_{ij} is called the *(periodic) offset* of the arc (i,j). If (π, x) is a feasible timetable-tension-pair, it is straightforward to compute the unique corresponding vector of offsets.

Each of the three variables in the problem are of interest in themselves. The space of periodic tensions x has been analysed in-depth, in particular the convex hull X of all feasible tensions, see [2, 19, 23, 26]. Also the space of periodic offsets p has received attention, or better that of periodic cycle offsets z, which are analogous to periodic offsets and arise in an alternative MIP formulation of PESP, namely the cycle formulation

$$\begin{aligned} \text{Minimise} \quad & w^\top x \\ \text{subject to} \quad & \Gamma x = Tz, \\ & \ell \leq x \leq u, \\ & z \in \mathbb{Z}^{\mathcal{B}}, x \in \mathbb{R}^{A(G)} \end{aligned} \tag{3}$$

where $\mathcal{B}$ is some integral cycle basis with cycle matrix $\Gamma \in \{-1,0,1\}^{\mathcal{B} \times A(G)}$, and z is an integer vector of so-called *periodic cycle offsets*, see, e.g., [15] for further details. In [4] the polytope of feasible fractional cycle offset variables is recognised to be a zonotope, and several properties of PESP are derived via tilings of said zonotope.

What is instead of main interest in this paper is the space of periodic timetables.

▶ **Definition 1.** *For an instance (G, T, ℓ, u, w), the set Π of feasible periodic timetables can be written as*

$$\Pi := \left\{ \pi \in \mathbb{R}^{V(G)} \,\middle|\, \exists p \in \mathbb{Z}^{A(G)}, \forall (i,j) \in A(G) \colon \ell_{ij} \leq \pi_j - \pi_i + Tp_{ij} \leq u_{ij} \right\}. \tag{4}$$

In particular, by defining for each $p \in \mathbb{Z}^{A(G)}$ the polyhedron

$$R(p) := \left\{ \pi \in \mathbb{R}^{V(G)} \,\middle|\, \forall (i,j) \in A(G) \colon \ell_{ij} - Tp_{ij} \leq \pi_j - \pi_i \leq u_{ij} - Tp_{ij} \right\}, \tag{5}$$

the feasible timetable space can be expressed as the union

$$\Pi = \bigcup_{p \in \mathbb{Z}^{A(G)}} R(p). \tag{6}$$

As introduced in [4], each $R(p)$ is a *weighted digraph polyhedron* [14]. Namely, for any fixed $p \in \mathbb{Z}^{A(G)}$ it can be described as

$$R(p) = \left\{ \pi \in \mathbb{R}^{V(\overline{G})} \,\middle|\, \forall (i,j) \in A(\overline{G}) \colon \pi_j - \pi_i \leq \kappa(p)_{ij} \right\}, \tag{7}$$

for the weighted digraph $(\overline{G}, \kappa(p))$, with the following:

- vertices $V(\overline{G}) := V(G)$,
- arcs $A(\overline{G}) := A(G) \cup A(G^\top)$, where $A(G^\top) = \{(j,i) | (i,j) \in A(G)\}$,
- weights $\kappa(p)_{ij} := u_{ij} - Tp_{ij}$ for all $(i,j) \in A(G)$, and $\kappa(p)_{ij} := Tp_{ji} - \ell_{ji}$ for all $(i,j) \in A(G^\top)$.

By construction, every $(\overline{G}, \kappa(p))$ is strongly connected, therefore the lineality space of its weighted digraph polyhedron is solely $\mathbb{R}\mathbf{1}$ [14].

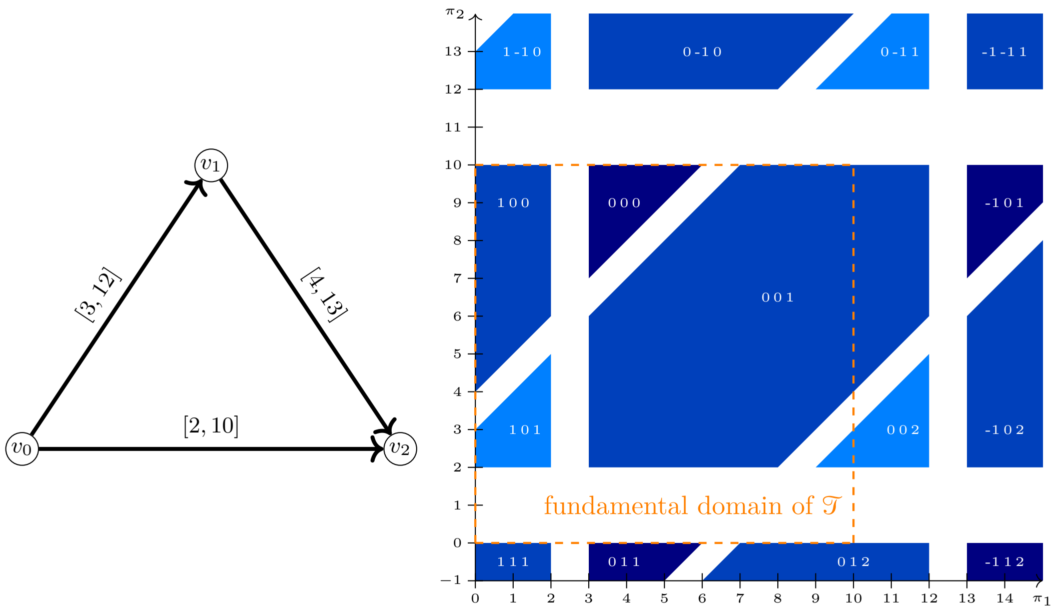


Figure 1 A PESP instance and its polytropal decomposition $\Pi/\mathbb{R}\mathbf{1}$ ($T = 10$, w arbitrary).

Let $\mathbb{T} := \mathbb{R} \cup \{\infty\}$ be the *tropical semiring*, with the tropical sum $a \oplus b := \min\{a, b\}$ and the tropical product $a \odot b := a + b$, see, e.g., [12] for more background. A set $S \subset \mathbb{T}^n$ is *tropically convex* if $(a \odot x) \oplus (b \odot y) \in S$ for every $x, y \in S$, and any $a, b \in \mathbb{T}$. It was shown in [14] that weighted digraph polyhedra arise as the *tropical convex hull* of finitely many points with coordinates in $\mathbb{T}$. Moreover, when the underlying digraph is strongly connected, no ∞-coordinates appear. In this case, which is the one interesting to us, all weighted digraph polyhedra can be seen as *polytropes* [13] by quotienting out the trivial lineality space, i.e., $\mathbb{R}\mathbf{1}$. We refer to [4] to see how general properties of polytropes translate to the context of periodic timetabling, e.g., the relation between polytrope vertices, vertices of the tension polytope X, and spanning tree structures. See also [24].

It is clear that $R(p) \cap R(q) = \varnothing$ holds for any two distinct offset vectors p and q, since $u - \ell < T$ by hypothesis. If $\Pi \neq \varnothing$, then the set $\Pi/\mathbb{R}\mathbf{1}$ is therefore a disjoint union of infinitely many polytropes, as we have visualized for a small exemplary instance in Figure 1. However numerous, these polytropes adhere to a certain structure, which we summarise in the following proposition.

▶ **Proposition 2** ([4], §3.3). *Consider the PESP instance (G, T, ℓ, u, w) with timetable space Π, denoting as B the incidence matrix of G, and as $\mathcal{B}$ an integral cycle basis of G, with cycle matrix Γ. Then:*

1. *For any feasible timetable $\pi \in \Pi$ all of its translations by integer multiples of T are feasible: If $\pi \in R(p)/\mathbb{R}\mathbf{1}$, then $\pi + Tq \in R(p + B^\top q)/\mathbb{R}\mathbf{1}$ for all $q \in \mathbb{Z}^{V(G)}$.*
2. *Two feasible timetables $\pi, \pi' \in \Pi$ have the same associated periodic tension if and only if there exists $q \in \mathbb{Z}^{V(G)}$ such that $\pi' = \pi + Tq$.*
3. *Two feasible timetables $\pi, \pi' \in \Pi$ have the same associated periodic tension if and only if $\Gamma p = \Gamma p'$ for the associated offsets $p, p' \in \mathbb{Z}^{A(G)}$.*

The unbounded set $\Pi/\mathbb{R}\mathbf{1}$ then turns out to be simpler than expected, since its ambient space can be restricted by another quotient, based on the equivalence relation

$$p \cong p' \iff \Gamma p = \Gamma p' \tag{8}$$

implied in the above proposition. In view of the cycle formulation (3) of PESP, we consider p and p' equivalent whenever they correspond to the same cycle offset. With $n := |V(G)|$, we define the *torus of feasible periodic timetables* $\mathcal{T} := (\mathbb{R}^n / (T\mathbb{Z})^n) / \mathbb{R}\mathbf{1}$. This is an $(n-1)$-dimensional torus of side length T, whose representative can be any full-dimensional hypercube of side length T in $\mathbb{R}^n/\mathbb{R}\mathbf{1}$, called *fundamental domain*. It now makes sense to also have a shorthand notion to refer to the quotient of our weighted digraph polyhedra in $\mathcal{T}$. We choose $(R(p)/(T\mathbb{Z})^n)/\mathbb{R}\mathbf{1} =: \mathbf{R}(p) \subseteq \mathcal{T}$.

To conclude this recapitulatory section, it is now possible to describe how the polytropes position themselves inside some fundamental domain, along the lines of [4]. Given a PESP instance (G, T, ℓ, u, w), we define (in breach of our hypothesis of $u - \ell < T$) its *limit instance*, where all upper bounds u_a are substituted with $\ell_a + T$, and denote it by $(G, T, \ell, w)_\infty$. For a polytrope $\mathbf{R}(p)$ of the base instance we denote as $\mathbf{R}'(p)$ the polytrope in the limit instance that contains it. We now say that two non-empty polytropes $\mathbf{R}(p)$ and $\mathbf{R}(q)$ are *neighbours* when $\mathbf{R}'(p)$ and $\mathbf{R}'(q)$ intersect in a common facet.

▶ **Proposition 3** ([4], §3.7). *Let $p \in \mathbb{Z}^{A(G)}$ be an offset vector with $\mathbf{R}(p) \neq \emptyset$, k an integer, and $e_{ij} \in \mathbb{Z}^{A(G)}$ the canonical basis vector of the arc $(i,j) \in A(G)$. Then:*
1. *If $|k| > 2$, then $\mathbf{R}(p + ke_{ij})$ is empty.*
2. *If $|k| > 1$, then $\mathbf{R}(p)$ and $\mathbf{R}(p + ke_{ij})$ are not neighbours.*
3. *If $|k| = 1$ and $\mathbf{R}(p + ke_{ij}) \neq \varnothing$, then $\mathbf{R}(p)$ and $\mathbf{R}(p + ke_{ij})$ are neighbours, and one of the two inequalities defined by the arc (i,j) is facet-defining for $\mathbf{R}(p)$: For $k = 1$ this is the lower bound inequality $\pi_j - \pi_i \geq \ell_{ij} - Tp_{ij}$, for $k = -1$ this is the upper bound inequality $\pi_j - \pi_i \leq u_{ij} - Tp_{ij}$.*
4. *Two non-empty polytropes $\mathbf{R}(p)$ and $\mathbf{R}(q)$ are neighbours whenever there exist representatives of the equivalence classes of p and q whose difference is, up to sign, a canonical basis vector. In other words, whenever there exists an arc $(i,j) \in A(G)$ such that $[p]_\cong - [q]_\cong = [\pm e_{ij}]_\cong$.*

This allows the construction of the *neighbourhood graph* of an instance, whose nodes are the equivalence classes of offsets, and two classes are adjacent if their respective polytropes are neighbours in $\mathcal{T}$. For the limit instance of the instance of Figure 1, the polytropal decomposition is depicted in Figure 2 next to the neighbourhood graph derived from it: Each dark blue triangle corresponds to the equivalence class $[0,0,0]_\cong$ and shares a facet only with the hexagon, corresponding to the class $[0,0,1]_\cong$. This, in turn, has both $[0,0,0]_\cong$ and $[0,0,2]_\cong$ as neighbours.

3 Tropical Neighbourhood Search

We can now outline the core steps undertaken by the promised heuristic, which we call *Tropical Neighbourhood Search* (`tns`). It is a local improvement heuristic that operates within the framework of the concurrent solver [3]. The solver keeps a *pool* of the feasible solutions it finds, ordered by objective value, and various local improvement heuristics use the pooled solutions as starting points. In particular, `tns`, once given a starting solution $(\pi^\star, x^\star, p^\star)$, identifies the polytrope $\mathbf{R}(p^\star)$ and proceeds to explore some neighbouring polytropes, i.e., it determines the optimal weighted tension over each $\mathbf{R}(p)$ for (a subset of) neighbours p of $p^\star$. If an improving solution is found, it is added to the pool. Doing so, it in fact operates on the neighbourhood graph of the given instance.

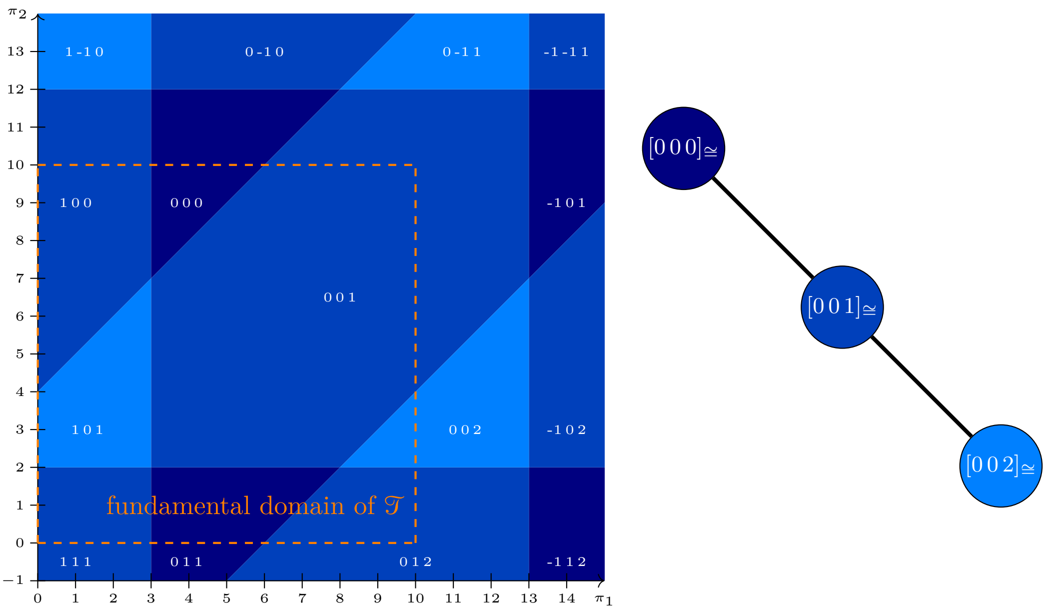


Figure 2 Limit instance and neighbourhood graph for the same instance as in Figure 1.

Algorithm 1 Tropical Neighbourhood Search `tns`.

Require: PESP instance (G, T, ℓ, u, w)

$(\pi^\star, x^\star, p^\star) \leftarrow$ pick a starting solution from the pool
$x_{\texttt{tns}} \leftarrow x^\star$
for arc (i, j) and direction $k \in \{-1, +1\}$ in *exploreList* **do**
  fix offset $p \leftarrow p^\star + k e_{ij}$
  solve $\mathrm{PESP}|_p$
  if $\mathrm{PESP}|_p$ feasible **then**
    $(\pi_{\mathrm{opt}}, x_{\mathrm{opt}}, p_{\mathrm{opt}}) \leftarrow$ optimal solution of $\mathrm{PESP}|_p$
    $improv \leftarrow (w^\top x_{\texttt{tns}} - w^\top x_{\mathrm{opt}})/(w^\top x_{\texttt{tns}})$
    if $improv > 0$ **then**
      Add $(\pi_{\mathrm{opt}}, x_{\mathrm{opt}}, p_{\mathrm{opt}})$ to the pool
      if $improv > qualityFactor$ **then**
        break
      $x_{\texttt{tns}} \leftarrow x_{\mathrm{opt}}$

Formally, our heuristic can be described by Algorithm 1, where

- $\mathrm{PESP}|_p$ is simply PESP restricted to the specific offset vector p, i.e., we solve (2) with all integer variables fixed to p. This is a linear program, which is dual to an uncapacitated minimum cost network flow problem [25].
- *exploreList* is a list of arc-direction tuples, indicating which neighbours to explore. It may contain only a subset of all possibilities.
- The solution picking method could vary in principle, although in our implementation it always selects the solution in the pool with smallest weighted tension.
- *qualityFactor* $\in [0, 1]$ is a factor utilized as a preemptive exit condition, which triggers when the percentage improvement of a newly found solution exceeds this factor.

Note that Algorithm 1 is a description of `tns` with the incidence matrix formulation of PESP (2). One can equivalently work with the cycle formulation (3) instead, which changes the LP subproblem of PESP to $\mathrm{PESP}\big|_z$ for a vector z of periodic cycle offsets. This poses no issue, and we refer to the next section for more details.

It is known that `tns` can be used to escape local minima reached via the modulo network simplex, as there even exist such instances where the neighbourhood graph has none [4].

4 Implementation details

As anticipated, there are various elements in Algorithm 1 that may alter the overall behaviour and performance of the algorithm depending on how they are adjusted. Therefore, before moving forward with the computational experiments, we will now detail the characteristics of such elements, what settings and strategies we decided to employ, the motivations that moved us, and other minor implementation details.

4.1 Preparing the *exploreList*

A deciding factor for both the speed and the behaviour of our `tns` heuristic is determining the search space. Clearly, choosing which or how many neighbouring polytropes to explore is a factor that deserves consideration, but even the order of exploration may affect the overall performance, since it has potentially positive interplay with concurrency.

As we know from Proposition 3, only arcs (i, j) whose inequalities are facet-defining for $\mathbf{R}(p^\star)$ can yield feasible neighbours, and only in the appropriate direction. Unfortunately, knowing a priori which inequalities are facet-defining is not trivial, and we therefore detail two strategies: one precise but slow, one fast but imprecise. When setting up *exploreList* in our tests, we decided to either scan all possible neighbours, i.e., all pairs $(a, +1)$ and $(a, -1)$ for all arcs $a \in A(G)$, or to restrict ourselves to a subset of the facet-defining inequalities, namely those that are tight in the starting solution $(\pi^\star, x^\star, p^\star)$, i.e., pairs $(a, +1)$ if $x^\star_a = \ell_a$ and pairs $(a, -1)$ if $x^\star_a = u_a$. This second way only the faces on a particular side of the polytrope are considered. We mark the first *exploreList* strategy by `all`, and the second one by `side`. The `side` strategy is quick to set up, but has the defect of not considering all facet-defining inequalities. Note that when a simplex-based LP solver is invoked on $\mathrm{PESP}\big|_p$ or $\mathrm{PESP}\big|_z$, then $(\pi^\star, x^\star, p^\star)$ will be a vertex of $\mathbf{R}(p^\star)$.

Given the equivalence relation $\cong$ (8), we know that polytropes are uniquely determined by their cycle offset $z = \Gamma p$. Given then neighbouring periodic offsets $p + e_{i_1 j_1}$ and $p + e_{i_2 j_2}$, for arcs (i_1, j_1) and (i_2, j_2) in $A(G)$, it may happen that $\Gamma(p + e_{i_1 j_1}) = \Gamma(p + e_{i_2 j_2})$ and the two explorations end up being identical. Therefore, another way of avoiding irrelevant explorations is to fix any cycle matrix Γ of the instance graph and pre-process all arcs, storing a unique representative for all arcs whose columns in Γ are identical.

4.2 Sorting the *exploreList*

Another choice to be made while preparing *exploreList* is the order in which to consider the arc-direction pairs. This can be influential because if a good solution is put into the pool earlier, then it is earlier available to other methods in the concurrent solver. In particular, in combination with the quality factor, this can lead to `tns`-loops that are shorter but still improvement-dense. We decided to use four different strategies to sort the arcs:

- `s1` descending weight w_a, to prioritize exploration of heavy and hence influential arcs;
- `s2` descending span $u_a - \ell_a$, to prioritize exploration of neighbours that are close-by, and therefore more likely feasible, since two neighbouring polytropes $\mathbf{R}(p)$ and $\mathbf{R}(p \pm e_{ij})$ have distance at most $T - (u_{ij} - \ell_{ij})$;

s3 descending weighted span $w_a(u_a - \ell_a)$, to combine the two sorting strategies above;

s4 descending average improvement, so as to prioritize exploration via arcs that on average have given good improvements in previous iterations. While the previous three strategies are pre-processed at the beginning, this is a dynamic sorting strategy, which keeps track of the average (positive and negative) improvements given by each arc throughout the various iterations. The rationale behind this is to prioritize all those arcs which provided net improvements but not the best improvement overall. Initially all averages are set to 0, and no changes is made in case of infeasibility. This is similar to pseudocost branching in mixed-integer programming [1].

4.3 The *qualityFactor*

The quality factor can be interpreted as a percentage, based on which the `tns`-loop is terminated early in case of a percentage-improvement that exceeds the given bound. As limit cases this means that any positive improvement whatsoever is enough to conclude the search when the factor is set to 0%, and that no quality-based exit can happen when the factor is set to 100% or more. In our tests, we perform our tests using two quality factors:

- `q0.001`, meaning 0.1% quality factor.
- `q1`, meaning 100% quality factor: every arc-direction tuple of *exploreList* is considered.

4.4 Subproblem Formulation: Arc Offsets vs. Cycle Offsets

As mentioned, Algorithm 1 is `tns` with respect to the incidence formulation of PESP (2), but one can equivalently perform `tns` using the cycle formulation (3) instead. The algorithm then reads the same as Algorithm 1, except that the cycle offsets are computed and used instead, with line 4 changing to $z \leftarrow z^\star + k\Gamma e_{ij}$, and line 5 now solving $\text{PESP}\big|_z$. Notice that Γe_{ij} is indeed the (i, j)-th column of the cycle matrix. In this, the choice of which cycle basis to use can be quite influential on the solving speed of the linear programs $\text{PESP}\big|_z$. Preliminary tests showed that for each instance there can be impressive differences, up to a factor 14, between the average solving times of different problem formulations and different cycle bases.

In order to choose which formulation to use for each instance, we compared the average `for`-loop iteration time of each of them and then simply picked the fastest one. The formulations tested were the incidence matrix formulation, and four variants of the cycle matrix formulation. One used a minimum width cycle basis [17], whereas three used different fundamental cycle bases: from a minimum span, minimum weight, and a minimum weighted span spanning tree, respectively. Since the average iteration time appeared very consistent throughout the tests and short even in the worst cases, tests of less than a minute per formulation are more than enough to process hundreds of linear programs and thereby compute an applicable average iteration time. In particular, the cycle formulation performed well overall, with the fundamental cycle bases of a minimum weighted span spanning tree being the fastest in all but two instances, where the fundamental cycle bases of a minimum span spanning tree were best instead.

Regardless of the specific cycle bases used in our tests, these evaluations were fast to obtain, and it can be suggested that a similar pre-evaluation strategy could be systematically used in the future.

We use Gurobi 9.5 [10] to solve each iteration's linear program.

4.5 Hashing Visited Polytropes

Throughout repeated use of `tns`, in particular in the exploration of different neighbourhoods, it is possible to explore the same polytrope multiple times, since the neighbourhoods of any two polytropes may have non-trivial intersection. A way to prevent this from happening is then to progressively keep a record of every processed offset vector and skip it whenever it is encountered again. In our preliminary performance evaluations this tracking method seemed to have little effect, positive or negative. We therefore decided to maintain it, hoping for a stronger impact in longer tests.

5 Computational Experiments and Results

We conducted several tests on eight `PESPlib` instances [5] of varying size, namely `R1L1`, `R2L2`, `R3L3`, `R4L4`, `R1L1v`, `R4L4v`, `BL1`, `BL3`. The last two are bus timetabling instances, whereas the rest are based on railway networks. For each we used both `warm` starts, employing initial solutions close to the `PESPlib` current best primal bounds (cf. Table 1), and `cold` starts without any initial solution.

Table 1 Initial solution values for the warm starts.

R1L1	31 099 786	R2L2	43 404 232	R3L3	44 837 461	R4L4	38 836 756
R1L1v	43 258 386	R4L4v	64 408 523	BL1	8 457 513	BL3	8 502 382

Table 2 Parameter combinations for `tns`.

Instances	*exploreList* arcs	*exploreList* sort	*qualityFactor*	Initial Solution
R1L1, R2L2, R3L3, R4L4, R1L1v, R4L4v, BL1 , BL3	all side	s1 s2 s3 s4	q0.001 q1	cold warm

Overall the various `tns` parameters are summarised in Table 2. Each combination was tested within the concurrent solver [3]. Going forward we will refer to `mns` tests when the modulo network simplex method works alone, `tns+mns` tests when the two heuristics run concurrently, and `complete` tests when `tns` and all the methods implemented in the concurrent solver are used together.

5.1 Impact of Parameter Choices for `tns`

In a first step, we want to evaluate how much the choice of arc-direction tuples and their sorting influence our results. We run `tns+mns` and compare it to `mns` alone, for all parameter combinations, see Table 2. For a meaningful comparison of the test runs, we disabled multi-node cuts within the `mns` implementation, because of their randomizing character. To complete the analysis we also run `complete` tests. The computation time per configuration is one hour wall time each, performed on an Intel i7-9700K CPU with 64 GB RAM.

5.1.1 `mns+tns` vs. `mns`

We can make the following observations: In combination with `mns`, our new heuristic was able to beat `mns` alone for all instances, as becomes evident from Table 6. Highlighted entries correspond to the best objective in comparison to the other parameter choices per instance.

The last column, corresponding to the objective value obtained by `mns` alone is never in the first place, while any other column is the winner at least once.

We point out that for one instance, namely the `warm`-started `R4L4`, `mns` was not able to find any improvement, while all four sortings of `all` arcs with low *qualityFactor* found the same improvement. This supports our claim that `tns` can be used to escape local minima.

To assess each heuristic's performance, we rank them by their objective value after 6 minutes (i.e., 10% of the total running time) and after 1 hour (100%), such that the best objective is ranked in first place, and assign the same placement number for equal objectives. When comparing the average ranking values, it is hard to discern a clear ranking in between the `tns` parameter choices: `all` with pseudocost-like improvement (`s4`) and *qualityFactor* `q1` seems best on average, but is a clear winner only for the `cold` `R3L3` instance.

The two exemplary plots for the two instances `R1L1v` and `BL3`, see Figure 3, show the development of the objective for `mns` vs. `tns`+`mns`. It is evident that each of the parameter choices is reasonable, and depending on the instance may perform well or not. For example, `side-s1-q1` is the best for `BL3-cold`, yet one of the worst heuristics in the `R1L1v-cold` run.

Another property which can be seen in the figures is that in the beginning, the `side` instances tend to perform better. After a while however, the `all`-runs become competitive.

5.1.2 `complete` Runs

This phenomenon is even stronger when evaluating the parameter choices in the `complete` runs: When comparing the average ranking of the methods after 6 minutes with the final state after an hour, as indicated in Table 7, one can observe that – with the exception of `all-s3-q0.001` – the `all`-heuristics rank better, while `side` methods worsen.

This behaviour can be also observed when looking at the graphs for the `complete` case in Figure 4. What catches the eye in these figures is that the `all` runs seem to have the same shape in objective development as the `side` runs, but lag behind. After a while however, the dark (`all`) strands catch up to the light (`side`) strands. A similar pattern can be observed in most of the instances, particularly for the larger ones. An explanation for this could be that in the beginning, improvements are easily found, and `side` will quickly update the pool and restart with a better solution, while `all` will continue to iterate through all options, even though better solutions have already been obtained (possibly) by other concurrent methods. In contrast, in the later stages, when improving solutions are hard to find, it pays off to search through all of the neighbouring polytropes. In contrast to `mns`+`tns`, in `complete` a low quality factor produces better results on average. This can be explained in a similar way as above: A low quality factor disrupts unnecessary explorations when larger improvements are found in the beginning. With time, the improvements in objective become smaller, such that *qualityFactor* has less of an impact, so that most of the arc-direction pairs in *exploreList* are explored anyway.

We conclude that all sorting factors are relevant, as each one performs well for some instances. Which one is the best choice is hard to predict in advance, and overall – particularly in the interplay with other concurrent heuristics – their influence is not large. Both `side` and `all` are valid choices for *exploreList*: The former is better suited for earlier stages of solving, while the latter performs well once improvements become hard to find.

5.2 Contribution of `tns` in Comparison to Other Methods

Aside from the behaviour as discussed in the previous section, we want to analyze the quality of the contributions of `tns` in the scope of the concurrent solver. To this end, we compare the improvement of the objective value obtained by the different algorithms in the `complete` runs.

Table 3 `tns`' contribution to the improvement gained in the `complete` runs in %.

	warm			cold		
	avg.	min	max	avg.	min	max
`BL1`	7.43	0.0	17.97	3.68	0.02	8.52
`BL3`	4.68	0.0	14.59	2.92	0.0	6.04
`R1L1`	7.66	6.38	10.04	2.95	0.04	5.15
`R1L1v`	0.81	0.0	5.02	2.96	0.03	5.92
`R2L2`	6.89	0.0	31.64	2.76	0.05	5.46
`R3L3`	15.05	6.98	21.71	2.95	1.05	6.21
`R4L4`	9.52	1.16	12.45	1.89	0.89	3.23
`R4L4v`	0.56	0.0	1.5	1.48	0.0	3.76

What should be noted first, is that the total improvements of `cold` starts are significantly larger than those of `warm` starts, and this also holds for `tns` in the `complete` runs. In relation to the total improvements however, the contribution of `tns` is larger for `warm` starts. We see evidence of that in Table 3: It shows the average, minimum and maximum improvement found by any of the *exploreList*'s choices in relation to the total improvement in the `complete` run in percent. With the exception of `R4L4v` and `R1L1v`, the average (and in most cases also the maximal) values of the `warm` started instances is larger than of the `cold` started ones. Table 4 displays the same, except that the first 6 minutes are excluded from the improvements. One can observe that compared to Table 3 the contribution of `tns` increases for the `cold` instances. This observation suggests that `tns` is particularly well suited for the later stages of solving a PESP instance, namely when improvements increment more slowly. At the beginning, when still far from a (local) minimum, `tns` is dominated by other algorithms in the solver.

Table 4 `tns`' contribution to the improvement gained in the `complete` runs in % after 6 minutes.

	warm			cold		
	avg.	min.	max.	avg.	min.	max.
`BL1`	4.97	0.06	11.87	4.73	0.0	13.78
`BL3`	6.74	0.0	26.61	5.32	0.01	26.48
`R1L1`	0.0	0.0	0.0	8.28	0.0	41.78
`R1L1v`	9.46	0.0	72.64	7.41	1.15	28.49
`R2L2`	2.76	0.0	11.18	2.22	0.0	10.61
`R3L3`	1.79	0.47	4.02	7.15	0.0	37.09
`R4L4`	2.84	0.0	15.62	1.37	0.0	3.45
`R4L4v`	0.42	0.0	1.38	2.59	0.0	14.44

Very noticeable in Table 3 and Table 4 is the wide range of `tns`' contribution for the different choices of *exploreList*. In almost all instances there is at least one choice which provides close to zero improvement, while the maximum value goes up to double-digit percentages. This property can also be observed in Figure 5. Here, we have chosen the exemplary instance `R3L3` and displayed the fractional contributions of all used heuristics in the `complete` solver. The `warm` started instance (left) has significantly more contribution through `tns` (green parts) in comparison to the `cold` started one. The top plots display the contributions for the whole time frame, while the lower plots show them for the last 54 minutes. When comparing the upper to the lower plots, it becomes evident, that some of *exploreList*'s choices gain in importance, while others seem to perform particularly badly.

E.g., for the `cold` run, each of the sortings with low *qualityFactor* seem to contribute similarly in the beginning, but after the first 6 minutes have passed, `s4` clearly contributes the most to the concurrent solution, yet the largest total improvement is found by `s1`, with only little direct `tns` contribution. Which one of the sortings provide the best solution is not clear however, our experiments did not show any clear indication. We therefore conclude that it may be worth it to try different sorting techniques in `tns` if no good improvements are found.

Based on Figure 5, we observe that the runs with high *qualityFactor* result in less improvement than with low *qualityFactor*, and the `tns` contribution is also often higher for low quality factors. While not the case for each instance and sorting, this seems to be a general tendency. When comparing `tns`' influence over time, this hierarchy is less prominent.

This supports again our interpretation of the previous section: Low quality factors are advantageous in the beginning. At a later stage, when the objective improvements become smaller, the *qualityFactor* exit condition is rarely triggered, regardless of low or large choice.

5.3 New `PESPlib` Incumbents

Based on the observation that `tns` contributes significantly to finding better solutions for PESP instances within the framework of the concurrent solver, we were able to compute new best primal solutions for 5 out of the 8 considered `PESPlib` instances. For some of these instances, we could find such a solution already within one hour in our `complete` experiments (see, e.g., `BL3` in Table 7). We then let the solver run for another 8 hours to further improve the timetables. We summarize the objective values of these new incumbents in Table 5.

Table 5 New incumbents for 5 `PESPlib` instances found with the help of `tns`. The old values are as of July 7, 2022. The last column shows the (wall) time of discovery.

Instance	New Value	Old Value	Time (s)
`BL3`	6 675 098	6 999 313	25 732
`R1L1v`	42 591 141	42 667 746	9 110
`R3L3`	40 483 617	40 849 585	3 547
`R4L4`	36 703 391	36 728 402	11 122
`R4L4v`	61 968 380	64 327 217	3 625

6 Outlook

The `tns` algorithm turns out to be a valuable supplement to the already enormous zoo of periodic timetabling heuristics, being capable to provide timely and practical schedule improvements. For future research, it seems reasonable to embed `tns` in a metaheuristic such as tabu search or simulated annealing in order to overcome local optima. Another branch of research would be to employ automated algorithm configuration techniques [28] to find out which parameters work best for a given instance.

References

1 M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, December 1971. doi:10.1007/BF01584074.

2 R. Borndörfer, H. Hoppmann, M. Karbstein, and N. Lindner. Separation of cycle inequalities in periodic timetabling. *Discrete Optimization*, 35:100552, February 2020. doi:10.1016/j.disopt.2019.100552.

3 R. Borndörfer, N. Lindner, and S. Roth. A concurrent approach to the periodic event scheduling problem. *Journal of Rail Transport Planning & Management*, 15:100175, September 2020. `doi:10.1016/j.jrtpm.2019.100175`.

4 E. Bortoletto, N. Lindner, and B. Masing. The tropical and zonotopal geometry of periodic timetables, 2022. `doi:10.48550/ARXIV.2204.13501`.

5 M. Goerigk. PESPlib - A benchmark library for periodic event scheduling, 2012. URL: `http://num.math.uni-goettingen.de/%7Em.goerigk/pesplib/`.

6 M. Goerigk and C. Liebchen. An Improved Algorithm for the Periodic Timetabling Problem. In G. D'Angelo and T. Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 12:1–12:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ATMOS.2017.12`.

7 M. Goerigk, A. Schöbel, and F. Spühler. A Phase I Simplex Method for Finding Feasible Periodic Timetables. In M. Müller-Hannemann and F. Perea, editors, *21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2021)*, volume 96 of *Open Access Series in Informatics (OASIcs)*, pages 6:1–6:13, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ATMOS.2021.6`.

8 M. Goerigk and A. Schöbel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers and Operations Research*, 40(5):1363–1370, May 2013. `doi:10.1016/j.cor.2012.08.018`.

9 P. Großmann, S. Hölldobler, N. Manthey, K. Nachtigall, J. Opitz, and P. Steinke. Solving Periodic Event Scheduling Problems with SAT. In J. He, D. Wei, A. Moonis, and W. Xindong, editors, *Advanced Research in Applied Artificial Intelligence*, Lecture Notes in Computer Science, pages 166–175, Berlin, Heidelberg, 2012. Springer. `doi:10.1007/978-3-642-31087-4_18`.

10 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL: `https://www.gurobi.com`.

11 C. Helmberg, T. Hofmann, and D. Wenzel. Periodic event scheduling for automated production systems. *INFORMS Journal on Computing*, 34(2):1291–1304, 2022. `doi:10.1287/ijoc.2021.1101`.

12 M. Joswig. *Essentials of tropical combinatorics*, volume 219 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2021.

13 M. Joswig and K. Kulas. Tropical and ordinary convexity combined. *Advances in Geometry*, 10(2):333–352, 2010. `doi:doi:10.1515/advgeom.2010.012`.

14 M. Joswig and G. Loho. Weighted digraphs and tropical cones. *Linear Algebra and its Applications*, 501:304–343, 2016. `doi:10.1016/j.laa.2016.02.027`.

15 C. Liebchen. *Periodic timetable optimization in public transport.* PhD thesis, Technische Universität Berlin, 2006.

16 C. Liebchen and R. H. Möhring. The Modeling Power of the Periodic Event Scheduling Problem: Railway Timetables — and Beyond. In F. Geraets, L. Kroon, A. Schoebel, D. Wagner, and C. D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, Lecture Notes in Computer Science, pages 3–40, Berlin, Heidelberg, 2007. Springer. `doi:10.1007/978-3-540-74247-0_1`.

17 C. Liebchen and L. Peeters. Integral cycle bases for cyclic timetabling. *Discrete Optimization*, 6:98–109, February 2009. `doi:10.1016/j.disopt.2008.09.003`.

18 N. Lindner and C. Liebchen. New Perspectives on PESP: T-Partitions and Separators. In V. Cacchiani and A. Marchetti-Spaccamela, editors, *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*, volume 75 of *OpenAccess Series in Informatics (OASIcs)*, pages 2:1–2:18, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 2190-6807. `doi:10.4230/OASIcs.ATMOS.2019.2`.

19 N. Lindner and C. Liebchen. Determining All Integer Vertices of the PESP Polytope by Flipping Arcs. In D. Huisman and C. D. Zaroliagis, editors, *20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2020)*, volume 85 of *OpenAccess Series in Informatics (OASIcs)*, pages 5:1–5:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2020.5.

20 N. Lindner and J. Reisch. An analysis of the parameterized complexity of periodic timetabling. *Journal of Scheduling*, February 2022. doi:10.1007/s10951-021-00719-1.

21 N. Lindner and R. van Lieshout. Benders decomposition for the periodic event scheduling problem. Technical Report 21-29, ZIB, Takustr. 7, 14195 Berlin, 2021.

22 G. P. Matos, L. M. Albino, R. L. Saldanha, and E. M. Morgado. Solving periodic timetabling problems with SAT and machine learning. *Public Transport*, August 2020. doi:10.1007/s12469-020-00244-y.

23 K. Nachtigall. Cutting Planes for a Polyhedron Associated with a Periodic Network. *undefined*, 1996.

24 K. Nachtigall. Periodic network optimization and fixed interval timetables. Technical report, Deutsches Zentrum für Luft- und Raumfahrt e.V., 1999. LIDO-Berichts. URL: https://elib.dlr.de/3657/.

25 K. Nachtigall and J. Opitz. Solving Periodic Timetable Optimisation Problems by Modulo Simplex Calculations. In M. Fischetti and P. Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*, volume 9 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 2190-6807. doi:10.4230/OASIcs.ATMOS.2008.1588.

26 M. A. Odijk. Construction of periodic timetables, part 1: A cutting plane algorithm. Technical Report 94-61, TU Delft, 1994.

27 J. Pätzold and A. Schöbel. A Matching Approach for Periodic Timetabling. In M. Goerigk and R. Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 2190-6807. doi:10.4230/OASIcs.ATMOS.2016.1.

28 E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, and K. Tierney. A survey of methods for automated algorithm configuration, 2022. doi:10.48550/ARXIV.2202.01651.

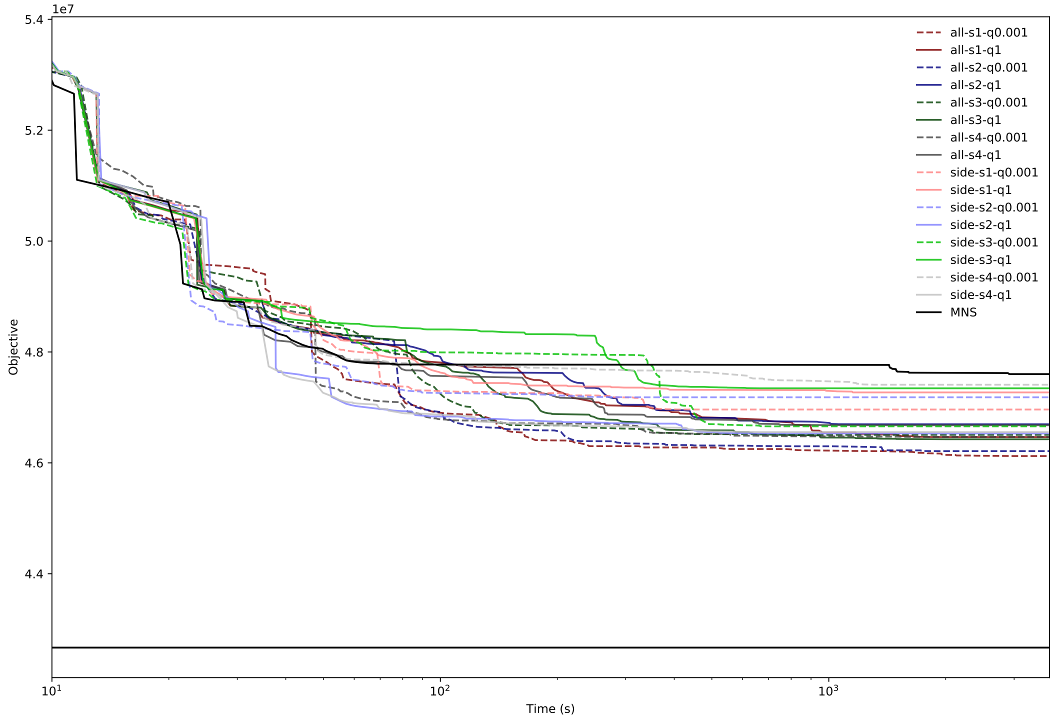
29 P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989. doi:10.1137/0402049.

A Appendix

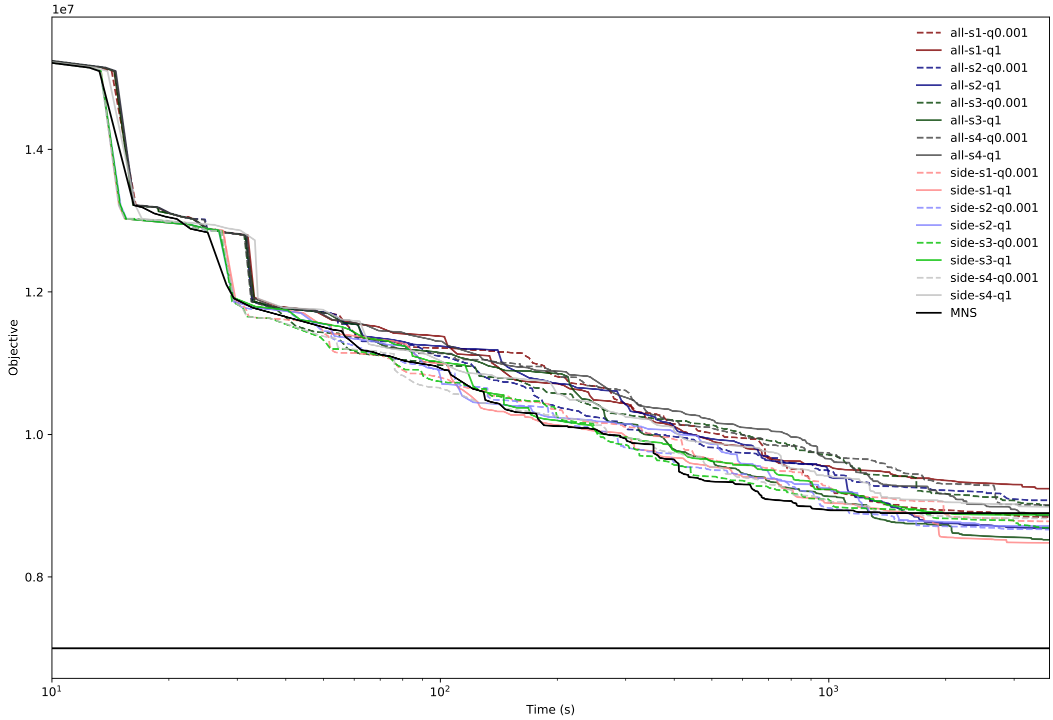
Table 6 Objective values of `tns` and `mns` in parallel after 1h wall time, in comparison to `mns` alone.

	all								
	s1-q0.001	s1-q1	s2-q0.001	s2-q1	s3-q0.001	s3-q1	s4-q0.001	s4-q1	
BL1-warm	8258561	8235826	8186321	8300527	8182495	8238180	8208581	8213667	
BL1-cold	8477888	8403606	8549631	8857331	8275775	8520448	8817740	8346471	
BL3-warm	8359354	8348109	8098761	8312052	8062572	8348109	8327236	8379410	
BL3-cold	8842185	9239538	9075754	8684087	8999692	8522210	9012239	8861540	
R1L1-warm	30678496	30610793	30600296	30588100	30678496	30578866	30600296	30583524	
R1L1-cold	35788003	35103477	35300490	35646174	35588509	35374751	35589367	35382965	
R1L1v-warm	42943355	42943355	42943355	42943355	42943355	42943355	42943355	42943355	
R1L1v-cold	46122798	46465421	46212022	46696787	46504815	46425825	46467141	46678146	
R2L2-warm	43398483	43382565	43382736	43382736	43398483	43382565	43398483	43382736	
R2L2-cold	45545963	46106414	44143731	45270818	45016237	45032259	44405920	44233438	
R3L3-warm	44546204	44544442	44591244	44539593	44544442	44544948	44548577	44593350	
R3L3-cold	44296073	43407015	42430742	42488680	43840438	42806733	42610123	42176416	
R4L4-warm	37387179	37460489	37405396	37492682	37312244	37367970	37366297	37363497	
R4L4-cold	42975571	41336513	42929915	42261165	42627952	41546954	42003636	42335060	
R4L4v-warm	64403669	64408523	64403669	64406909	64403669	64408523	64403669	64408523	
R4L4v-cold	65376186	66594246	66351066	66694524	65949084	66391164	66296248	65823105	
6 min. ranking	9.88	10.62	7.25	8.38	8.62	9.12	8.38	6.69	
final ranking	9.0	8.06	6.75	9.0	7.06	6.5	7.62	6.19	

	side								
	s1-q0.001	s1-q1	s2-q0.001	s2-q1	s3-q0.001	s3-q1	s4-q0.001	s4-q1	MNS
BL1-warm	8156407	8217310	8273813	8197344	8204102	8214464	8145927	8177211	8362237
BL1-cold	8790388	8464797	8523165	8509967	8681972	8375229	8688826	8676485	8973473
BL3-warm	8376740	8263350	8408354	8065263	8408354	7946851	8193045	8029258	8359914
BL3-cold	8780061	8479738	8664483	8715578	8692029	8873040	8821544	8985128	8895307
R1L1-warm	30674972	30658531	30691866	30688021	30756833	30688021	30756833	30681160	30684785
R1L1-cold	36423144	35686177	36133582	35353728	36044751	36078420	36541854	35633823	36390414
R1L1v-warm	42943355	42943355	42943355	42943355	42943355	42943355	42943355	42943355	42946450
R1L1v-cold	46961984	47270557	47182681	46530813	46658767	47345018	47409082	46555105	47600910
R2L2-warm	43398483	43364985	43398483	43386980	43386980	43398483	43398483	43364985	43385954
R2L2-cold	44667116	44593903	44502873	44640728	44496851	44428158	44403440	44522515	44504010
R3L3-warm	44795451	44795451	44795451	44795451	44795451	44795451	44795451	44795451	44810246
R3L3-cold	42187095	43170308	44316260	43665920	43376728	43507592	43430305	43574669	45577898
R4L4-warm	37415040	37399063	37356432	37386139	37314559	37288889	37363831	37311361	37444171
R4L4-cold	42846346	42044976	41788252	41528536	41952148	41772120	42575956	42763956	42622532
R4L4v-warm	64408523	64408523	64408523	64408523	64408523	64408523	64408523	64408523	64408523
R4L4v-cold	66228809	65578506	65739359	65741111	66134278	66032110	65175140	65877024	66270894
6 min. ranking	8.12	6.19	7.75	6.88	8.62	6.56	7.56	7.88	6.81
final ranking	9.94	7.06	9.5	7.19	8.69	7.81	8.69	7.5	13.44



(a) `R1L1v-cold`.



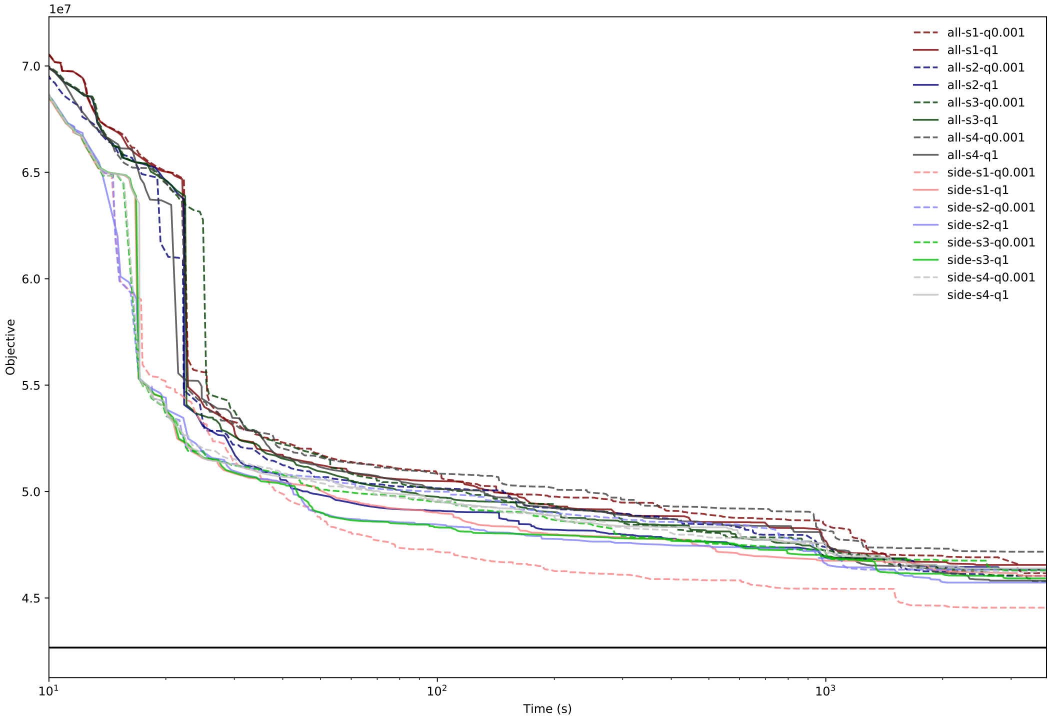
(b) `BL3-cold`.

Figure 3 Examples of the objective progression in comparison to different parameter choices for parallel `mns`+`tns` in comparison to `mns`.

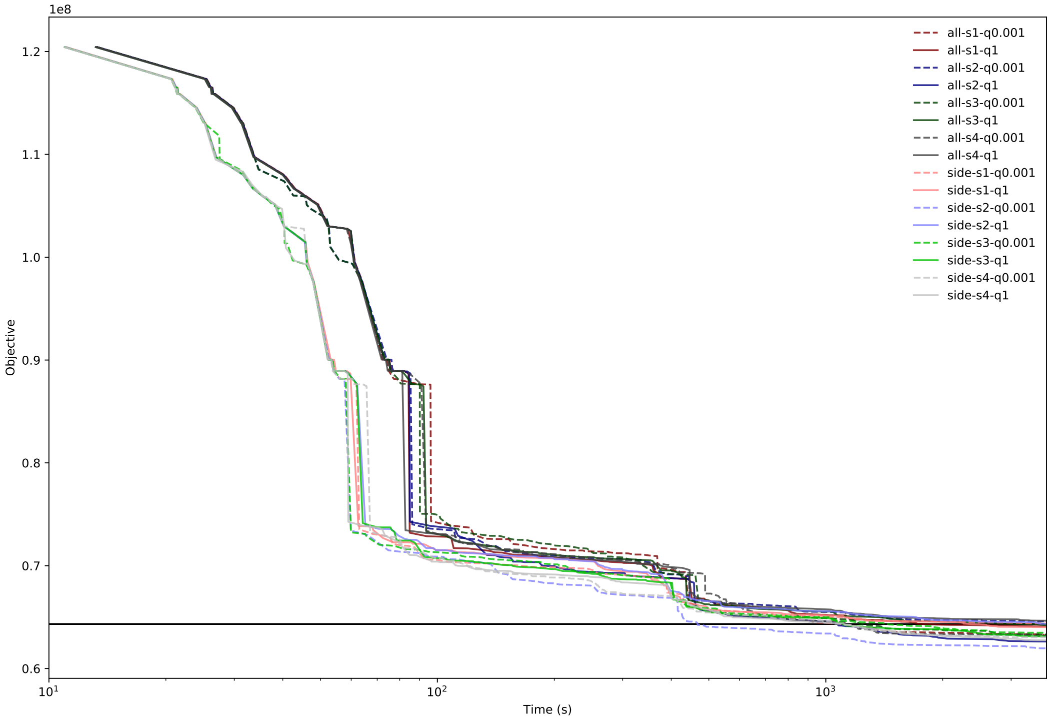
Table 7 Objective values after 1h runtime with the `complete` strategy.

	all							
	s1-q0.001	s1-q1	s2-q0.001	s2-q1	s3-q0.001	s3-q1	s4-q0.001	s4-q1
BL1-warm	6792526	6935103	6547850	6697639	6572855	6740896	6562891	7025590
BL1-cold	7621147	7457877	6465738	6758000	7144032	6699148	7147572	6911380
BL3-warm	7334701	7140000	7259769	7023881	6974763	7206833	7553069	7164378
BL3-cold	7406444	7727433	7629390	7753854	7768767	7233719	7462949	7716933
R1L1-warm	30426994	30423140	30423140	30426994	30426994	30426994	30431036	30426994
R1L1-cold	34020450	33522247	31692344	34177686	31856836	33795188	34125594	33794486
R1L1v-warm	42943355	42943355	42943355	42814750	42801531	42943355	42943355	42943355
R1L1v-cold	46169674	46551486	46041342	46326249	45768586	46341535	47172536	45813504
R2L2-warm	43207702	43319135	43206244	43275789	43263142	43329691	43047738	43329691
R2L2-cold	42361958	42416233	41640213	43131819	42337570	42512473	42586419	41960342
R3L3-warm	44408363	44397465	44379012	44399516	44378435	44371830	44397486	44411156
R3L3-cold	41222660	42739161	41228048	42440292	42384919	40483617	42758160	44132022
R4L4-warm	36909735	36916544	36901735	36935621	36928689	36960219	36997153	36990506
R4L4-cold	41823843	41243736	43339597	42061213	42174340	40282996	43336050	41504147
R4L4v-warm	64330043	64328991	64340252	64285960	64340252	64339747	64339747	64340252
R4L4v-cold	63222568	64090696	64580054	62632707	63302814	64225355	63173171	64649625
6 min. ranking	12.5	8.62	9.5	9.88	7.56	10.19	10.38	10.31
final ranking	8.69	9.19	6.75	9.19	7.94	8.06	11.0	10.5

	side							
	s1-q0.001	s1-q1	s2-q0.001	s2-q1	s3-q0.001	s3-q1	s4-q0.001	s4-q1
BL1-warm	6527346	6593280	6429697	6504754	6483936	6508182	6570960	6533503
BL1-cold	6542043	7101531	6650416	7195907	6455312	6791979	6808078	6649762
BL3-warm	6871983	7308628	7341305	7030706	7362216	7040927	6909267	6903738
BL3-cold	7163591	7504180	7349736	7614397	7514692	7232455	7212076	7247561
R1L1-warm	30426994	30426994	30426994	30423140	30425260	30426994	30426994	30425260
R1L1-cold	32816267	33578895	33551641	33642348	33857174	33321098	33785910	33708393
R1L1v-warm	42886458	42943355	42943355	42943355	42943355	42943355	42943355	42943355
R1L1v-cold	44544618	46040263	46330633	45723589	46295094	45923497	46228814	46390710
R2L2-warm	43203025	43318930	43191843	43004597	43318930	43222313	43100634	43047991
R2L2-cold	41095503	42522032	41856192	42195345	42390560	42168432	41914851	42347940
R3L3-warm	44430322	44382720	44433727	44414666	44321035	44404928	44385201	44400190
R3L3-cold	41391010	41773723	41985509	41473233	40993187	42284859	40959638	44300876
R4L4-warm	36974381	36923867	36953228	36915142	36935274	37064318	36913014	36814620
R4L4-cold	42207683	41930820	41605374	42155011	41124227	41549938	41784069	40541428
R4L4v-warm	64340252	64340252	64250155	64339747	64339747	64339747	64339747	64339747
R4L4v-cold	64389979	64026343	61968380	64348631	63482236	63127690	63036902	62757263
6 min. ranking	4.19	6.31	5.69	5.31	4.75	6.06	5.31	6.25
final ranking	6.5	9.0	6.69	7.06	7.0	6.81	5.56	6.06

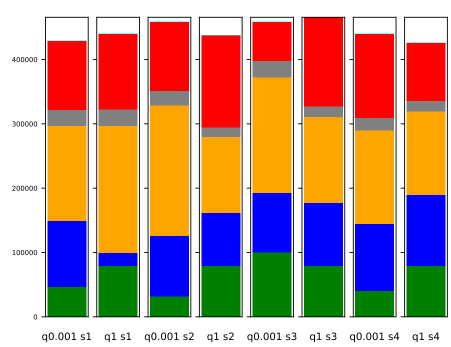


(a) `R1L1v-cold`.

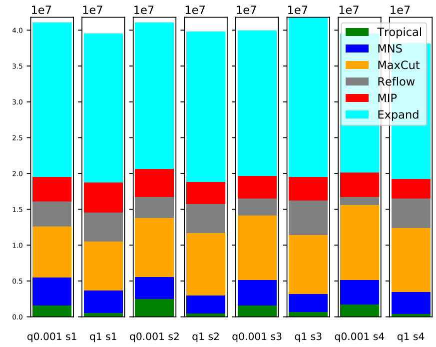


(b) `R4L4v-cold`.

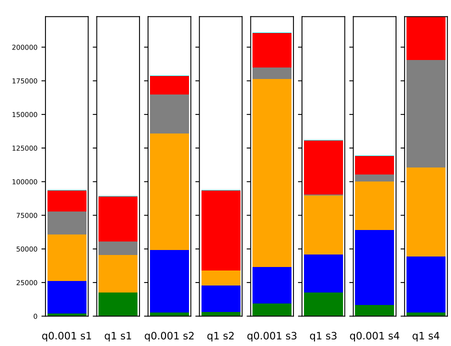
Figure 4 Examples objective progression in comparison to different heuristics in the complete concurrent solver.



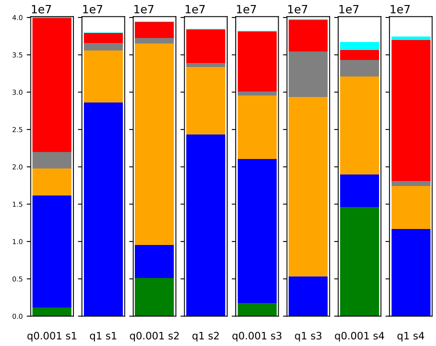
(a) `warm`.



(b) `cold`.



(c) `warm` after 6 minutes.



(d) `cold` after 6 minutes.

Figure 5 Contribution of the individual methods in the concurrent solver and `tns` (green) for the instance `R3L3`.