



A Phase I Simplex Method for Finding Feasible Periodic Timetables

Marc Goerigk¹  

Network and Data Science Management, Universität Siegen, Germany

Anita Schöbel  

Department of Mathematics, TU Kaiserslautern, Germany

Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM, Kaiserslautern, Germany

Felix Spühler  

Business Information Systems, TU Braunschweig, Germany

Abstract

The periodic event scheduling problem (PESP) with various applications in timetabling or traffic light scheduling is known to be challenging to solve. In general, it is already NP-hard to find a feasible solution. However, depending on the structure of the underlying network and the values of lower and upper bounds on activities, this might also be an easy task.

In this paper we make use of this property and suggest phase I approaches (similar to the well-known phase I of the simplex algorithm) to find a feasible solution to PESP. Given an instance of PESP, we define an auxiliary instance for which a feasible solution can easily be constructed, and whose solution determines a feasible solution of the original instance or proves that the original instance is not feasible. We investigate different possibilities on how such an auxiliary instance can be defined theoretically and experimentally. Furthermore, in our experiments we compare different solution approaches for PESP and their behavior in the phase I approach. The results show that this approach can be especially helpful if the instance admits a feasible solution, while it is generally outperformed by classic mixed-integer programming formulations when the instance is infeasible.

2012 ACM Subject Classification Applied computing → Operations research; Theory of computation → Network optimization

Keywords and phrases train timetable optimization, periodic event scheduling problem, modulo simplex

Digital Object Identifier 10.4230/OASICS.ATMOS.2021.6

1 Introduction

Railways play an important role in public transportation planning and form an essential component for green logistics and traveling in the future. Timetabling is a key element for planning public passenger transportation. In particular, periodic timetables, e.g., hourly repeated, are of interest from the passengers' perspective because they are easy to remember.

Periodic timetables have been extensively studied in the literature since their introduction by Serafini and Ukovich in 1989 as the periodic event scheduling problem (PESP) [20]. Works on PESP are not only of theoretical interest, as they are already used to optimize timetables in practice. In 2008, Liebchen successfully implemented an optimized timetable for the Berlin Underground [13]. While providing shorter passenger waiting times, it was also possible to reduce the number of trains. In 2006, Kroon et al. optimized the Dutch Railway System [10]. Their timetable was adapted to current and future needs, improving the service significantly and at the same resulting in approximately 40 million Euro additional annual profit.

¹ Corresponding author



In real-world applications, safety conditions must be considered which further complicate the problem. For example, trains have to maintain a safety distance when sharing the same rail. This is usually modeled by enforcing a time gap between the departures of two consecutive trains. The added complexity of safety constraints makes it more difficult to find feasible timetables for big instances.

Determining optimal timetables is a complex task. Even finding a feasible timetable is known to be NP-hard [20]. Recently, also the parameterized problem complexity has been studied [14]. The authors show that deciding the feasibility of PESP is W[1]-hard when parameterized by the vertex color number. However, it is easy to see that finding an optimal solution is possible in polynomial time on trees.

Solving large-scale problems to optimality remains out of reach for current families of algorithms, such as the modulo simplex method [7, 15], a matching-based heuristic [17], or methods based on SAT solving [8]. Recent papers [1, 2, 4, 9] provide further progress towards this long-term goal.

In this paper, we provide a new approach for finding feasible timetables. This approach is inspired by the phase I of the classic simplex method for linear programming. A timetabling instance is extended by adding virtual edges to the underlying network, which makes it simple to find a feasible solution in the thus extended network. By minimizing its objective function, a feasible solution to the original problem instance can be found, or a certificate of infeasibility is given.

The remainder of this paper is structured as follows. In Section 2, we briefly recall the formal problem definition and basic properties. We then introduce the phase I approach for finding feasible timetables in Section 3. Using instances from the *LinTim*-library, we evaluate advantages and disadvantages of this method in Section 4, before concluding the paper in Section 5.

2 The Periodic Event Scheduling Problem

In this section, the main definitions of the periodic event scheduling problem (PESP) and two mixed-integer formulations are briefly revisited. For more details, we refer to [11, 12, 16].

An event-activity network (EAN) $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ is a directed graph where nodes represent events (such as the departure or arrival of trains of a directed line) and arcs represent activities (such as drive, wait, transfer and security headway activities). Without loss of generality, we assume the event-activity network \mathcal{N} to be connected. If not, all considerations could be applied to each connected component separately. We also assume that $|\mathcal{E}| \leq |\mathcal{A}|$, i.e., there are at least as many activities as events. This assumption holds for all connected event-activity networks unless the event-activity network is a tree. In this case, all considerations of the PESP are trivial.

For each activity $a \in \mathcal{A}$ the minimal and maximal allowed duration are denoted by L_a and U_a with $0 \leq L_a \leq U_a$ and $L_a, U_a \in \mathbb{N}$. Together, $\Delta_a = [L_a, U_a]$ is called the time span of a . A periodic timetable $\pi \in \mathbb{Z}^{|\mathcal{E}|}$, where $\pi_i \in [0, T - 1]$ for all $i \in \mathcal{E}$ with a time period T , is called feasible if for all $a = (i, j) \in \mathcal{A}$ there exist so-called modulo parameters $z \in \mathbb{Z}$ such that $\pi_j - \pi_i + zT \in \Delta_a$. The PESP in the context of timetabling then consists of finding a feasible periodic timetable that minimizes the weighted travel time $\sum_{a=(i,j) \in \mathcal{A}} w_a(\pi_j - \pi_i + z_a T)$ for given (passenger) weights w_a for each $a \in \mathcal{A}$. An instance of the problem is thus defined by the tuple $I = ((\mathcal{E}, \mathcal{A}), w, L, U)$. A well-known mixed-integer programming (MIP) formulation for PESP is the following.

$$\text{(NodeIP)} \quad \min \quad \sum_{a=(i,j) \in \mathcal{A}} w_a(\pi_j - \pi_i + z_a T) \quad (1)$$

$$\text{s.t.} \quad L_a \leq \pi_j - \pi_i + z_a T \leq U_a \quad \forall a = (i, j) \in \mathcal{A} \quad (2)$$

$$\pi_i \in \mathbb{Z} \quad \forall i \in \mathcal{E} \quad (3)$$

$$z_a \in \mathbb{Z} \quad \forall a \in \mathcal{A} \quad (4)$$

We briefly revisit another model that depends on cycles. Any cycle $C \subseteq \mathcal{A}$ is described by the incidence vector $\Gamma(C) \subseteq \{-1, 0, 1\}^{|\mathcal{A}|}$ with components $\Gamma(C)_a = 1$, if $a \in C$ in forward direction, $\Gamma(C)_a = -1$, if $a \in C$ in backward direction, and $\Gamma(C)_a = 0$ else. Using these incidence vectors as row vectors, we describe a set of cycles \mathcal{C} by a matrix $\Gamma \in \{-1, 0, 1\}^{|\mathcal{C}| \times |\mathcal{A}|}$, that is, it contains $\Gamma(C_a)^t$ in row a for each cycle $C_a \in \mathcal{C}$. Given a spanning tree \mathcal{T} , let $\Gamma \in \{-1, 0, 1\}^{|\mathcal{A}| - |\mathcal{E}| + 1, |\mathcal{A}|}$ be the matrix corresponding to its fundamental cycles. Then, Γ is called cycle matrix of \mathcal{N} with respect to \mathcal{T} . A vector $\xi \in \mathbb{Z}^{|\mathcal{A}|}$ is a periodic tension w.r.t. π if there exists $z \in \mathbb{Z}^{|\mathcal{A}|}$ such that $\pi_j - \pi_i + z_a T = \xi_a$ for all $a = (i, j) \in \mathcal{A}$. Note that $\xi \in \mathbb{Z}^{|\mathcal{A}|}$ is a tension in \mathcal{N} if and only if $\Gamma \xi = 0$. This results in the following cycle-based mixed-integer program for the PESP:

$$\text{(CBIP)} \quad \min \quad w^t \xi \quad (5)$$

$$\text{s.t.} \quad \Gamma \xi = T \tilde{z} \quad (6)$$

$$L \leq \xi \leq U \quad (7)$$

$$\xi_a \in \mathbb{Z} \quad \forall a \in \mathcal{A} \quad (8)$$

$$\tilde{z}_a \in \mathbb{Z} \quad \forall a \in \mathcal{A} \setminus \mathcal{T} \quad (9)$$

3 A Phase I Approach to PESP

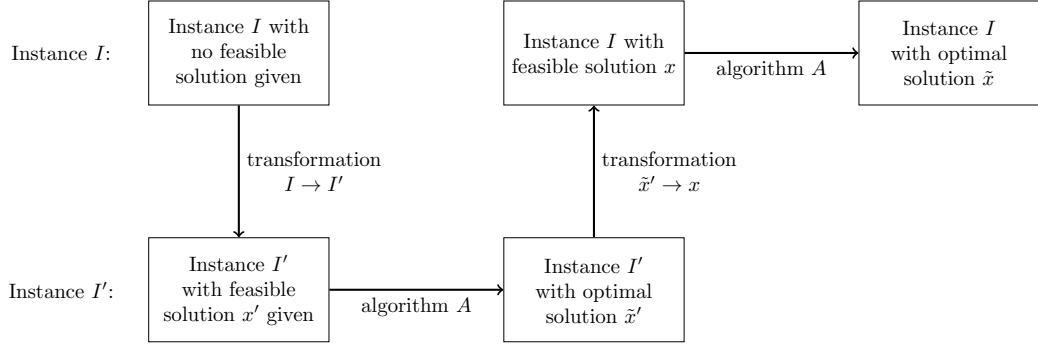
The general idea of phase I approaches can best be recalled looking at the classical simplex algorithm in which a linear program $\min\{c^T x : Ax = b, x \geq 0\}$ with costs $c \in \mathbb{R}^n$, a right-hand side $b \in \mathbb{R}^m$ (w.l.o.g. $b \geq 0$) and a matrix $A \in \mathbb{R}^{m,n}$ is given and we look for an optimal solution $x \in \mathbb{R}^n$. The simplex algorithm needs a feasible solution to get started. If this is not available, the well-known phase I of the simplex algorithm starts: by adding additional columns to the coefficient matrix A , it is extended to $(A|I)$, where I denotes the identity matrix. For the new linear program, the columns of I are chosen as basis, such that $x := (\underbrace{0, \dots, 0}_{\in \mathbb{R}^n}, \underbrace{b^t}_{\in \mathbb{R}^m})^t$ is a feasible starting solution. The auxiliary problem asks to

minimize the unit costs of the new variables as auxiliary objective function and is solved by the simplex algorithm (which is now possible since a starting solution is known). If and only if the auxiliary problem has objective value of zero, the original instance is feasible. In this case, a feasible solution to the original instance can be constructed by pivoting the auxiliary variables out of the basis. The generalized scheme of this process is depicted in Figure 1.

For the PESP we now proceed analogously: Given an instance of PESP $I = ((\mathcal{E}, \mathcal{A}), w, L, U)$ we construct an auxiliary instance by extending the given instance to

$$I^{\text{ext}} = ((\mathcal{E}^{\text{ext}}, \mathcal{A}^{\text{ext}}), w^{\text{ext}}, L^{\text{ext}}, U^{\text{ext}}).$$

We show that for I^{ext} a feasible solution can be easily constructed and that the original instance I has a feasible solution if and only if the optimal objective value of I^{ext} is zero.



■ **Figure 1** Schematic process of the phase I Approach with the original instance I , the extended instance I' and an algorithm A .

Recall that cycles make the PESP a hard problem. We hence want to make sure that each cycle contains a *flexible* activity, i.e., an activity with bounds $\Delta_a = [0, T - 1]$. Such an activity can collect all slack needed to ensure that the timetable satisfies the constraint of the respective cycle in constraint (6).

To this end, let the original instance $I = ((\mathcal{E}, \mathcal{A}), w, L, U)$ be given. We fix a set $A \subseteq \mathcal{A}$ (we will discuss later how this set can be chosen), and define the extended instance as follows. Each activity $a = (i, j) \in A$ of the original instance is replaced by two activities, namely by $a^{\text{old}} = (i, i_a)$ and $a^{\text{virt}} = (i_a, j)$ which are linked by one new event i_a , see Figure 2. The first activity carries the old lower and upper bounds, i.e., L_a and U_a from the original instance are now the bounds of a^{old} . The second activity is a flexible activity which receives $\Delta_{a^{\text{virt}}} = [0, T - 1]$ as lower and upper bounds.



■ **Figure 2** Extending an activity $a = (i, j) \in A$ with span $\Delta_a = [L_a, U_a]$ to two activities $a^{\text{old}} = (i^{\text{org}}, i_a^{\text{virt}}) \in \mathcal{A}^{\text{old}}$ with span $\Delta_a^{\text{old}} = [L_a^{\text{old}}, U_a^{\text{old}}] := [L_a, U_a]$ and $a^{\text{virt}} = (i_a^{\text{virt}}, j^{\text{org}})$ with span $\Delta_a^{\text{virt}} := [0, T - 1]$.

Formally, we define $\mathcal{E}^{\text{ext}} = \mathcal{E} \cup \{i_a : a \in A\}$ and $\mathcal{A}^{\text{ext}} = (\mathcal{A} \setminus A) \cup \mathcal{A}^{\text{old}} \cup \mathcal{A}^{\text{virt}}$, where $\mathcal{A}^{\text{old}} = \{(i, i_a) : a = (i, j) \in A\}$ and $\mathcal{A}^{\text{virt}} = \{(i_a, j) : a = (i, j) \in A\}$. As parameters we set:

$$L_a^{\text{ext}} := \begin{cases} L_a & \text{if } a \in \mathcal{A} \setminus A \\ L_{a'} & \text{if } a = (i, i_{a'}) \in \mathcal{A}^{\text{old}}, a' \in A \\ 0 & \text{if } a \in \mathcal{A}^{\text{virt}} \end{cases} \quad (10)$$

$$U_a^{\text{ext}} := \begin{cases} U_a & \text{if } a \in \mathcal{A} \setminus A \\ U_{a'} & \text{if } a = (i, i_{a'}) \in \mathcal{A}^{\text{old}}, a' \in A \\ T - 1 & \text{if } a \in \mathcal{A}^{\text{virt}} \end{cases} \quad (11)$$

$$w_a^{\text{ext}} := \begin{cases} 0 & \text{if } a \in \mathcal{A} \setminus A \\ 0 & \text{if } a \in \mathcal{A}^{\text{old}} \\ 1 & \text{if } a \in \mathcal{A}^{\text{virt}} \end{cases} \quad (12)$$

Given an original instance I and a set A we denote the new instance as $I^{\text{ext}}(A)$ where we leave out A if the context is clear. Clearly, this is again a PESP instance which can hence be formulated by the integer programs provided in Section 2. Note that the objective function

of I^{ext} only includes the virtual activities $\mathcal{A}^{\text{virt}}$. This is in line with the phase I approach for the classic simplex algorithm, in which also only the newly added columns are part of the auxiliary objective function.

Before we discuss how to choose $A \subseteq \mathcal{A}$ and how to find feasible solutions for instances $I^{\text{ext}}(A)$, we provide a basic theorem which states the relation between feasibility of the original instance I and the optimal objective of the extended instance $I^{\text{ext}}(A)$. Note that this statement is independent of the choice of A .

► **Theorem 1.** *Let I be an instance of PESP and $A \subseteq \mathcal{A}$ be a set of activities. Then, I is feasible if and only if the objective value $v(I^{\text{ext}}(A))$ of the extended instance is zero.*

Proof. First, let π be a feasible solution for I . Set

$$\pi_i^{\text{ext}} := \begin{cases} \pi_i & \text{if } i \in \mathcal{E} \\ \pi_{j'} & \text{if } i = i_{a'} \text{ with } a' = (i', j') \in A \end{cases},$$

i.e., on the original events $i \in \mathcal{E}$ we leave the timetable as it is while a virtual event $i_{a'}$ on the edge $a' = (i', j')$ obtains the timetable of the end-node j' of a' . To see that π^{ext} is feasible for I^{ext} we have to look at the three types of activities:

- For $a = (i, j) \in \mathcal{A} \setminus A$, both i and j are in \mathcal{E} and feasibility of π^{ext} for I^{ext} follows from feasibility of π for I .
- For $a = (i, i_{a'}) \in \mathcal{A}^{\text{old}}$ with $a' = (i, j) \in A$, $\pi_{i_{a'}}^{\text{ext}} - \pi_i^{\text{ext}} + z_a T \in \Delta_a$ holds since due to $\pi_{i_{a'}}^{\text{ext}} = \pi_j$ this is the constraint for the original activity $a' \in A$ which is satisfied, because the timetable π is feasible for I .
- Finally, for $a \in \mathcal{A}^{\text{virt}}$ feasibility is always satisfied since a is a flexible activity with $\Delta_a = [0, T - 1]$.

Note that the objective value of this solution is zero.

For the reverse direction, we start with a feasible timetable π for I^{ext} with objective

$$v(I^{\text{ext}}) = \sum_{a^{\text{virt}} = (i_a, j) \in \mathcal{A}^{\text{virt}}} (\pi_j^{\text{ext}} - \pi_{i_a}^{\text{ext}} + z_{a^{\text{virt}}} T) = 0.$$

Due to the constraints we know that $\pi_j^{\text{ext}} - \pi_{i_a}^{\text{ext}} + z_{a^{\text{virt}}} T \geq L_{a^{\text{virt}}} = 0$ for all $a^{\text{virt}} = (i_a, j) \in \mathcal{A}^{\text{virt}}$, hence we conclude that

$$\pi_j^{\text{ext}} - \pi_{i_a}^{\text{ext}} + z_{a^{\text{virt}}} T = 0 \tag{13}$$

for some $z_{a^{\text{virt}}} \in \mathbb{Z}$.

Given the timetable π^{ext} for I^{ext} we define the timetable π for I by projection, i.e., we just leave the values π_i , $i \in \mathcal{E}$ as they have been in I^{ext} . We now show that this timetable is feasible for the original instance I , i.e., that there exist modulo parameters z_a such that

$$\pi_j - \pi_a + z_a T \in \Delta_a$$

for all $a = (i, j) \in \mathcal{A}$. To this end, we consider the activities in $\mathcal{A} \setminus A$ and in A separately.

- For $a \in \mathcal{A} \setminus A$ feasibility of π for I follows from feasibility of π^{ext} for I^{ext} .
- Now let $a' = (i, j) \in A$. We know that for $a^{\text{old}} = (i, i_{a'}) \in \mathcal{A}^{\text{old}}$ we have

$$L_a \leq \pi_{i_{a'}}^{\text{ext}} - \pi_i^{\text{ext}} + z_{a^{\text{old}}} T \leq U_a \tag{14}$$

where we used that $L_a = L_{a^{\text{old}}}$ and $U_a = U_{a^{\text{old}}}$ according to (10) and (11). Adding (13) and (14) we receive

$$L_a \leq \pi_j - \pi_i + (z_{a^{\text{old}}} + z_{a^{\text{virt}}}) T \leq U_a.$$

Hence, $z_a := z_{a^{\text{old}}} + z_{a^{\text{virt}}} \in \mathbb{Z}$ is the required modulo parameter for $a' \in A$ and the claim is shown. ◀

We can directly conclude that looking at a lower bound while solving I^{ext} may suffice to decide non-feasibility of I .

► **Corollary 2.** *Let γ be a lower bound on the objective value of I^{ext} , i.e., $\gamma \leq v(I^{\text{ext}})$. If $\gamma > 0$, I is not feasible.*

We now analyze possibilities how the set $A \subseteq \mathcal{A}$ can be chosen. Keep in mind that we want to find a feasible solution of I^{ext} efficiently. We use a result for a special case, namely, if the event-activity network \mathcal{N} is a tree, the solution of PESP is easy: In the cycle-based formulation, the cycle matrix vanishes and $\xi = L$ is an optimal solution. Based on this result, the general idea to construct an instance with easy-to-find feasible solution is to add the flexible activities to the original activities $A \in \mathcal{A}$ in such a way that \mathcal{A}^{ext} without the flexible activities is a forest for which a timetable can be found easily for each of its connected components. The following result is easy to verify and therefore given without proof.

► **Lemma 3.** *Let $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ be an event-activity network with a spanning tree \mathcal{T} . Let $A_1 = \mathcal{A} \setminus \mathcal{T}$, $A_2 \subseteq \mathcal{T}$ and $A = A_1 \cup A_2$ the set for which virtual activities should be added. Finally, let $\mathcal{N}^{\text{ext}} = (\mathcal{E}^{\text{ext}}, \mathcal{A}^{\text{ext}})$ the extended event-activity network where $\mathcal{A}_1^{\text{ext}} \subseteq \mathcal{A}^{\text{ext}}$ corresponds to the activities in A_1 and $\mathcal{A}_2^{\text{ext}} \subseteq \mathcal{A}^{\text{ext}}$ corresponds to the activities in A_2 . Then, $(\mathcal{A} \setminus A) \cup \mathcal{A}_1^{\text{old}} \cup \mathcal{A}_2^{\text{old}} \cup \mathcal{A}_2^{\text{virt}}$ defines a spanning tree of the extended event-activity network.*

This means that we can solve PESP on the tree and obtain a feasible solution since all activities which are not in the tree are flexible activities. Specific choices for set A are as follows.

- *full*: Add a virtual activity for each activity in the original-event-activity network.
- *cycle_base*: Add a virtual activity for each fundamental circuit for a given spanning tree.
- *minimal*: Add a virtual activity for each fundamental circuit for a given spanning tree if this circuit does not already contain a flexible activity.

We also mention that the complexity of PESP does not increase when we turn from I to I^{ext} , since the number of cycles in a cycle basis stays the same.

► **Lemma 4.** *Let the original event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ with a spanning tree \mathcal{T} , the set $A \subseteq \mathcal{A}$ for which virtual activities should be added, and the corresponding extended event-activity network $\mathcal{N}^{\text{ext}} = (\mathcal{E}^{\text{ext}}, \mathcal{A}^{\text{ext}})$ with a spanning tree \mathcal{T}^{ext} be given. Then, the number of fundamental circuits is the same.*

Proof. The number of fundamental circuits of the original event-activity network is $|\mathcal{A}| - |\mathcal{T}|$ and for the extended network $|\mathcal{A}^{\text{ext}}| - |\mathcal{T}^{\text{ext}}|$. Then, by definition of the sizes of the extended sets of events and activities, it follows that

$$\begin{aligned} |\mathcal{A}^{\text{ext}}| - |\mathcal{T}^{\text{ext}}| &= |\mathcal{A}^{\text{ext}}| - (|\mathcal{E}^{\text{ext}}| - 1) = (|\mathcal{A}| + |A|) - ((|\mathcal{E}| + |A|) - 1) \\ &= |\mathcal{A}| - |\mathcal{E}| + 1 = |\mathcal{A}| - (|\mathcal{E}| - 1) = |\mathcal{A}| - |\mathcal{T}|. \end{aligned} \quad \blacktriangleleft$$

4 Experiments

4.1 Computational setup

We use the scientific software toolbox *LinTim*² [19, 5]. All algorithms except for the phase I approach are already implemented in *LinTim*. The code of the phase I approach is written in Python 3. If required, the mathematical optimization solver *Gurobi*³ in Version 9 is used and the Python package *networkx*⁴ for calculating spanning trees.

² See <https://www.lintim.net/>

³ See <https://www.gurobi.com/>

⁴ See <https://networkx.github.io/>

All experiments were conducted on a server with 12 Intel(R) Xeon(R) CPU X5675 processors (each at 3.07GHz) and 128 GB RAM. All algorithms have a time limit of 30 minutes and are limited to using one kernel.

The instances that are used for the computational experiments are also part of the *LinTim* toolbox, namely the data sets *toy*, *grid* (bus), *lowersaxony* (rail), *athens* (metro), *bahn-01*, *bahn-02*, *bahn-03*, and *bahn-04* (German high-speed network). On the basis of a given public transportation network with its stops, edges, and line concept, the event-activity network is created. Since the standard event-activity networks are all feasible, headway constraints are added to the event-activity network to complicate the problem and also potentially create infeasible instances. For each event-activity network there are ten different versions with headway times from 1 to 10 minutes.

Table 1 shows the number of events, the number of activities, especially the number of headway activities, and the number of fundamental circuits of the event-activity network for each data set.

■ **Table 1** Size of the event-activity networks of the used *LinTim* data sets. The number of activities includes the number of headway activities.

data set	events	activities	headway activities	fundamental circuits
<i>toy</i>	156	304	116	149
<i>grid</i>	448	901	264	454
<i>lowersaxony</i>	536	1077	388	542
<i>athens</i>	1388	3892	1576	2505
<i>bahn-01</i>	5036	16543	6766	11508
<i>bahn-02</i>	5468	19726	7774	14259
<i>bahn-03</i>	3592	10041	2734	6450
<i>bahn-04</i>	5356	19136	6192	13781

We test nine variations of the described phase I approach, where we use the three different extension methods *minimal*, *cycle_base*, and *full* from Section 3 and the following three algorithms to solve the extended PESP instance I^{ext} :

- *NodeIP*: Solving the node-based MIP (1)-(4)
- *CBIP*: Solving the cycle-based MIP (5)-(9)
- *MNS*: Using the modulo network simplex [15, 7]

NodeIP and *CBIP* are chosen because they are in principle able to solve the PESP optimally, and because they also update the lower bound of the objective function. We use a stopping criterion when reaching a lower bound of the objective value greater than 0. *MNS* is chosen as a heuristic approach that performs well for the PESP. For a detailed description of these methods please refer to [19] and [7]. In the following, the phase I and its combinations are denoted as phase I (<extending method>, <algorithm>). All methods are provided with the same starting solution.

To benchmark the phase I approach, we choose three other algorithms that are already implemented in *LinTim*, i.e., they do not use an extended network. The node-based integer formulation and the cycle-based formulation are chosen because they are straightforward approaches and may also be good for showing infeasibility. We slightly adapt them to make them comparable to the phase I by stopping them when they find the first feasible solution. In the following these variants are denoted as NodeIP-Feas and CBIP-Feas. The third algorithm is the Constraint Propagation, denoted as ConProp, because this algorithm is often used to find feasible solutions, see [6] and [19].

■ **Table 2** Size of the extended event-activity networks for the different extending methods.

data set		original	minimal	cycle_base	full
<i>toy</i>	<i>events</i>	156	296	305	460
	<i>activities</i>	304	434	453	608
<i>grid</i>	<i>events</i>	448	741	902	1349
	<i>activities</i>	901	1033	1355	1802
<i>lowersaxony</i>	<i>events</i>	536	1023	1078	1613
	<i>activities</i>	1077	1509	1619	2154
<i>athens</i>	<i>events</i>	1388	3320	3893	5280
	<i>activities</i>	3892	5244	6397	7784
<i>bahn-01</i>	<i>events</i>	5036	12405	16544	21579
	<i>activities</i>	16543	19773	28051	33086
<i>bahn-02</i>	<i>events</i>	5468	13751	19727	25194
	<i>activities</i>	19726	22031	33985	39452
<i>bahn-03</i>	<i>events</i>	3592	6211	10042	13633
	<i>activities</i>	10041	8827	16491	20082
<i>bahn-04</i>	<i>events</i>	5356	11555	19137	24492
	<i>activities</i>	19136	17751	32917	38272

In the following, we briefly explain how the run times are determined. For both solvers of the MIP formulations that use *Gurobi*, only the real optimization time is taken as computation time, i.e., without the time for reading the input data or calculating the spanning tree in case of the cycle-based MIP. For the other two algorithms the complete run time is considered, e.g., with reading input, because they could not be integrated in the existing algorithms. However, read-in routines take only a few seconds for the largest instances. Regarding the run times of the phase I, only the run time of the algorithms are taken into account, i.e., without the time that is needed to build the extended event-activity network.

4.2 Results

In the following, we analyze and compare all instances of all data sets together. We have 80 different instances (10 instances for each of the 8 data sets) in total and 120 experiments for each data set (employing each of the 12 algorithms for each of the 10 instances). For all but one instance, namely *bahn-04*, *headway=3*, we could decide whether they are feasible or not by at least one method. Due to the heuristic nature of MNS, it may happen that it stops without reaching an objective value of zero on the extended instance in phase I and before reaching the time limit. In this case, the run is counted as reaching the time limit.

Table 2 shows the number of events and activities for each way of extending the event-activity network. For the full method it is clear that the number of activities is doubled and hence also the number of events is more than doubled. We observe a similar behavior for the cycle_base method, although not as prominent as in the full case. The minimal method results in very similar numbers of events and activities as the cycle_base method for the smaller instances. Both methods noticeably differ only for the data sets *bahn-03* and *bahn-04* since many full span activities are removed before adding the virtual activities.

Due to the time limit and the heuristic nature of the modulo simplex, not all problems were solved correctly. If MNS did not reach the time limit, but found a solution with objective value greater than zero for a feasible instance, it is counted as time limit. On the other hand, if it reached an objective value greater than zero for an infeasible instance and stopped before

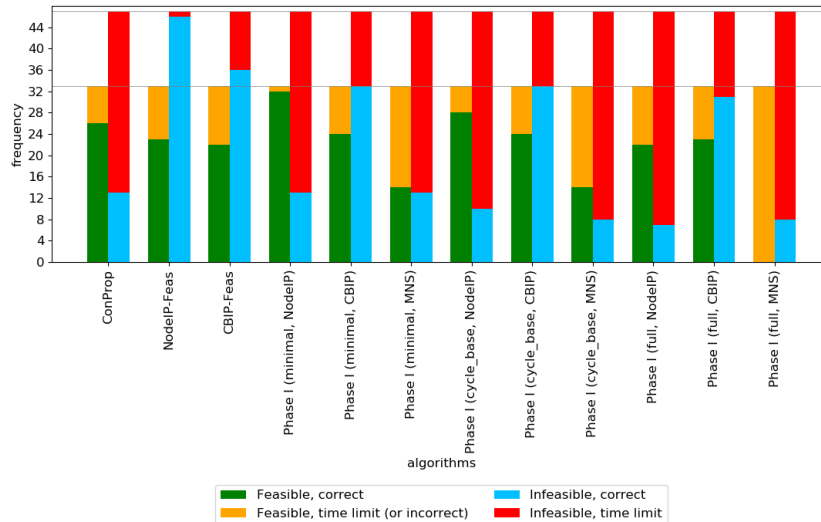


Figure 3 Correctness of all algorithms over all data sets split into feasible and infeasible instances.

the time limit, it is counted as a correct identification (however, the method is not able to prove this correctness). Figure 3 shows the correctness of the algorithms split in feasible and infeasible instances. We note that NodeIP-Feas is best in proving infeasibility. Also CBIP-Feas and phase I with CBIP perform well in proving infeasibility. ConProp and other phase I algorithms perform poorly in comparison. On the other hand, phase I with minimal or cycle_base and NodeIP finds a feasible solution more often than all other algorithms. ConProp and phase I with minimal or cycle_base and CBIP belong to the algorithms that find a feasible solution for most of the instances. Phase I with the full method and MNS does not find a correct feasible solution at all.

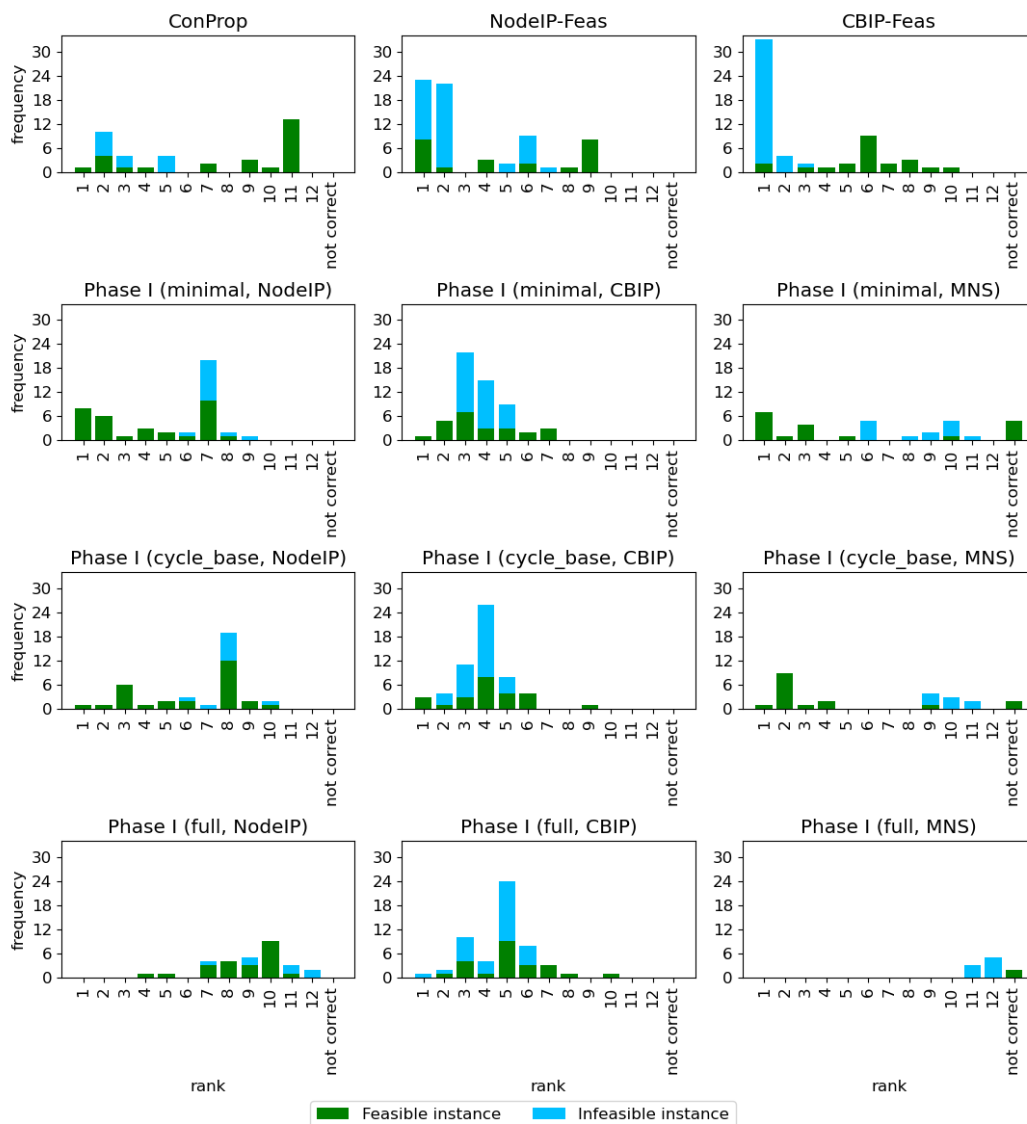
In Figure 4, the run times of the algorithms for each instance are ranked, i.e., how often an algorithm was the fastest, second fastest, . . . and how often the algorithm was not correct. For clarity reasons, there is no bar for the time limit. For the feasible instances, NodeIP-Feas and phase I with minimal and cycle_base as extending methods and NodeIP and CBIP as algorithms are the fastest algorithms over all. For the infeasible instances, CBIP-Feas is the fastest algorithm for most instances. Also NodeIP-Feas belongs to the group of the fastest algorithm, followed by phase I with CBIP.

Figure 5 shows a performance profile [3] over all 80 data sets. It shows the ratio of how many instances were solved within the time factor τ of the fastest algorithms for each instance. This means that at $\tau = 1$ the distribution of the fastest algorithms is shown, while at $\tau \approx 10^6$ the percent of solved instances is shown. We see that NodeIP-Feas and CBIP-Feas perform best with regards to the number of correctly solved instances. They are followed by phase I with the minimal and cycle_base method which has similar values. As already observed in previous figures, phase I with MNS and the full method performs worst.

4.3 Discussion

We first discuss the behavior of the algorithms on the different data sets. On the smallest data set *toy*, all algorithms consistently solve the instances correctly. Only phase I with MNS fails to determine the optimal solution within the time limit for some instances. The CBIP-Feas algorithm outperforms all other algorithms for all instances of *toy* with respect to the run time.

6:10 A Phase I Simplex Method for Finding Feasible Periodic Timetables

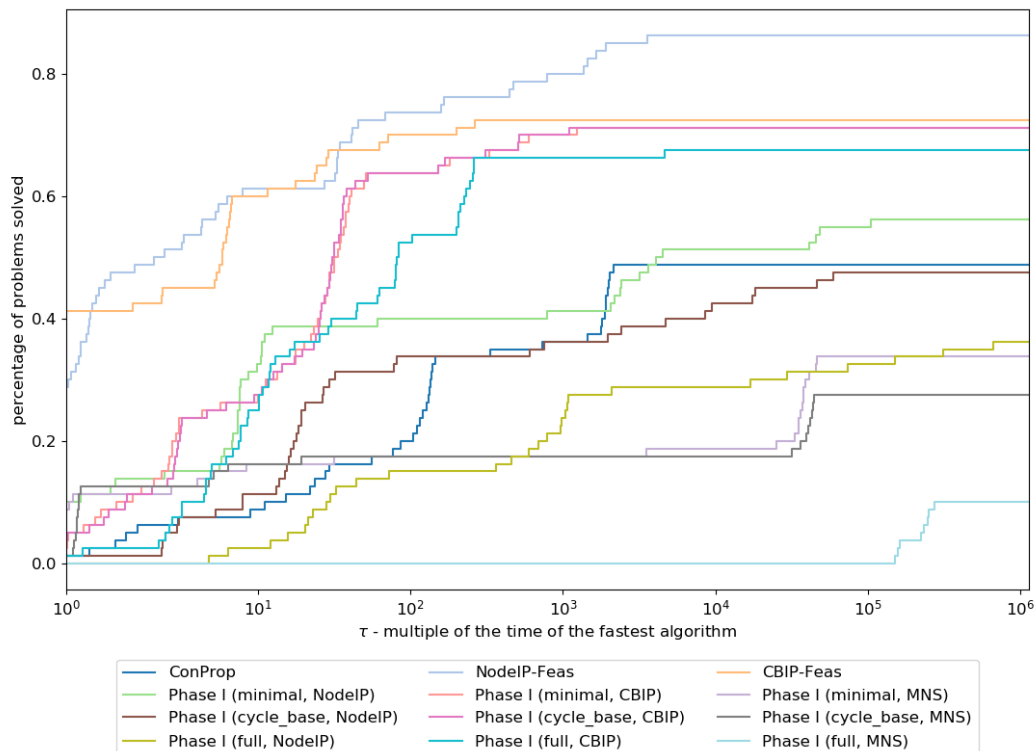


■ **Figure 4** Ranking of run times of all algorithms over all data sets split into feasible and infeasible instances.

For the middle-sized data sets *grid* and *lowersaxony*, classic MIP solvers NodeIP-Feas and CBIP-Feas excel. They solve all instances correctly and within the time limit. The remaining algorithms are not able to solve all instances within the time limit.

The *athens* data set is slightly larger than *grid* and *lowersaxony*, however, the main difference is that the underlying public transportation network has only a few cycles – a characteristic that benefits MNS. All but one algorithm are always correct.

Finally, due to the size of the *bahn* data sets, the time limit is often exceeded. On feasible instances, phase I with NodeIP outperforms all other algorithms. On infeasible instances, NodeIP-Feas is best, followed by CBIP-Feas and phase I with CBIP. NodeIP-Feas is correct for all but one instances.



■ **Figure 5** Performance profile of the run time of all algorithms over all data sets.

We now consider the different variants of the phase I approach. We observe that the algorithms can solve more instances correctly when we extend the event-activity network using the minimal method compared to the cycle_base method. When employing the CBIP algorithm the difference between the minimal method and cycle_base method are small. The difference is more prominent for the other two algorithms. Using the extending method full leads to fewer correctly solved instances within the time limit. We observe a similar pattern when analyzing the run times. The phase I algorithms solve phase I faster on average when the event-activity network is extended with the minimal method compared to the cycle_base method. We observe that phase I with the MIP solvers NodeIP and CBIP are correct more often and faster than phase I with MNS. The only data set where phase I with MNS is faster is *athens*.

We distinguish between feasible and infeasible instances to compare phase I with NodeIP to phase I with CBIP. While phase I with NodeIP solves more feasible instances correctly, phase I with CBIP solves more infeasible instances correctly. Phase I with NodeIP outperforms all other phase I approaches for the large and feasible instances; solving all but one instance correctly. To do so, it requires less run time than all other phase I algorithms.

In the next step, we compare the phase I to the established algorithms ConProp, NodeIP-Feas and CBIP-Feas. A direct comparison of the methods is difficult due to their heterogeneous performance on the instances. For that reason, we analyze them in-depth. We focus on the extending methods minimal and cycle_base combined with the algorithms NodeIP and CBIP. phase I with MNS is excluded from the analysis as it only performs well on *athens*. Likewise, the extending method full is excluded as it is outperformed by the other extending methods.

Comparing ConProp to the phase I approach, we observe that the phase I approach outperforms ConProp on multiple instances, with regards to number of correctly solved instances and run time. On feasible instances, phase I with NodeIP performs better. On infeasible instances, phase I with CBIP performs better. Comparing phase I to the MIP solvers, we should distinguish between feasible and infeasible instances. On feasible instances of the large data sets, classic MIP solvers fail to determine a feasible solution. In such scenarios, phase I is a better choice. For the middle-sized data sets no exact statement can be made. On infeasible instances and almost all cases, the NodeIP-Feas and CBIP-Feas perform best.

To conclude, on the studied infeasible instances the phase I approach cannot compete with the classic MIP solvers NodeIP-Feas and CBIP-Feas. On feasible instances, the phase I approach outperforms ConProp, in particular on large data sets. On these instances, classic MIP solvers often fail in determining a feasible solution within the time limit. Finally, we emphasize that the phase I approach outperforms all other algorithms on the data set *athens* with its special structure.

5 Conclusion

Finding periodic timetables is a well-known challenge when designing public transport systems. While finding a timetable with minimum travel time is notoriously difficult, already finding a feasible timetable is NP-hard. Often, such starting solutions are required as part of a local improvement method, such as the modulo network simplex.

In this paper, we developed a new method to find feasible timetables that is inspired by the phase I of the classic simplex method for linear programs. By adding virtual activities to a given event-activity network, we construct an alternative PESP instance for which a feasible solution is trivial to provide. We then solve this extended instance to find a solution that is feasible for the original problem.

We discussed different possibilities of adding virtual activities and conducted an extensive analysis of all combinations of extending methods and PESP algorithms on a set of problems taken from the *LinTim* library. Our results suggest that it is important to differentiate between feasible and infeasible instances when comparing algorithmic performances. While the new phase I approach has a higher success rate on feasible instance, the classic MIP solvers are noticeably better on infeasible instances. For best results, two algorithms may be started in parallel, as proposed in [2]: One to find a feasible solution and one to prove infeasibility.

Future research could focus on developing new extending methods, on the algorithms used for the phase I, and their combination. Furthermore, the behavior of a phase II should be further studied: how does the structure of starting solutions derived from different algorithms impact the subsequent optimization step? Finally, it would be interesting to use the starting solutions also for improving approaches (as in [18]) for integrating timetabling and routing.

References

- 1 Ralf Borndörfer, Heide Hoppmann, Marika Karbstein, and Niels Lindner. Separation of cycle inequalities in periodic timetabling. *Discrete Optimization*, 35:100552, 2020.
- 2 Ralf Borndörfer, Niels Lindner, and Sarah Roth. A concurrent approach to the periodic event scheduling problem. *Journal of Rail Transport Planning & Management*, 15:100175, 2020.
- 3 Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

- 4 Marc Goerigk and Christian Liebchen. An improved algorithm for the periodic timetabling problem. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017.
- 5 Marc Goerigk, Michael Schachtebeck, and Anita Schöbel. Evaluating line concepts using travel times and robustness: Simulations with the lintim toolbox. *Public Transport*, 5(3):267–284, 2013.
- 6 Marc Goerigk and Anita Schöbel. Engineering the modulo network simplex heuristic for the periodic timetabling problem. In *International Symposium on Experimental Algorithms*, pages 181–192. Springer, 2011.
- 7 Marc Goerigk and Anita Schöbel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers & Operations Research*, 40(5):1363–1370, 2013.
- 8 Peter Großmann, Steffen Hölldobler, Norbert Manthey, Karl Nachtigall, Jens Opitz, and Peter Steinke. Solving periodic event scheduling problems with sat. In *International conference on industrial, engineering and other applications of applied intelligent systems*, pages 166–175. Springer, 2012.
- 9 Sabrina Herrigel, Marco Laumanns, Jacint Szabo, and Ulrich Weidmann. Periodic railway timetabling with sequential decomposition in the pesp model. *Journal of rail transport planning & management*, 8(3-4):167–183, 2018.
- 10 Leo Kroon, Dennis Huisman, Erwin Abbink, Pieter-Jan Fioole, Matteo Fischetti, Gábor Maróti, Alexander Schrijver, Adri Steenbeek, and Roelof Ybema. The new dutch timetable: The or revolution. *Interfaces*, 39(1):6–17, 2009.
- 11 Christian Liebchen. *Periodic Timetable Optimization in Public Transport*. dissertation.de – Verlag im Internet, Berlin, 2006.
- 12 Christian Liebchen. Periodic timetable optimization in public transport. In *Operations research proceedings 2006*, pages 29–36. Springer, 2007.
- 13 Christian Liebchen. The first optimized railway timetable in practice. *Transportation Science*, 42(4):420–435, 2008.
- 14 Niels Lindner and Julian Reisch. Parameterized complexity of periodic timetabling. Technical Report ZIB Report 20-15, Zuse Institute Berlin, 2020.
- 15 Karl Nachtigall and Jens Opitz. Solving periodic timetable optimisation problems by modulo simplex calculations. In *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS’08)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.
- 16 Jens Opitz. *Automatische Erzeugung und Optimierung von Taktfahrplänen in Schienenverkehrsnetzen*, volume 1. Springer, 2009.
- 17 Julius Pätzold and Anita Schöbel. A matching approach for periodic timetabling. In *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016.
- 18 Philine Schiewe and Anita Schöbel. Periodic timetabling with integrated routing: Towards applicable approaches. *Transportation Science*, 54(6):1714–1731, 2020.
- 19 Anita Schöbel, Alexander Schiewe, Sebastian Albert, Julius Pätzold, Philine Schiewe, and Jochen Schulz. Lintim: An integrated environment for mathematical public transport optimization. Technical report, Documentation. Technical Report 2020.02, 2020.
- 20 Paolo Serafini and Walter Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989.