CORE
Metadata, citation and similar papers at core.ac.uk
Provided by CWI's Institutional Repository

Tree Decomposition Methods for the Periodic Event Scheduling Problem

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

Abstract

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

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

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

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

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.6

1 Introduction

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

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

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


© Irving I. van Heuven van Staereling;
licensed under Creative Commons License CC-BY
18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018).
Editors: Ralf Borndörfer and Sabine Storandt; Article No. 6; pp. 6:1–6:13

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

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

Overview

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

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

2 Problem description

2.1 The Periodic Event Scheduling Problem

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

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

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

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

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

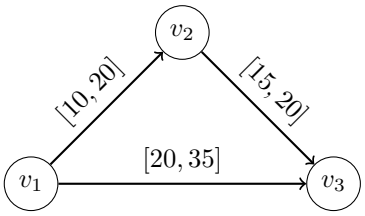

Figure 1 Example of a PESP instance visualized in a graph ($T = 60$).

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

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

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

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

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

2.2 Complexity of PESP

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

2.3 Handling the modulo operator

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

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

at the cost of one extra integer variable p_{ij} per constraint (in similar other models, p_{ij} can also be a binary variable). Here, $p_{ij} \in \mathbb{Z}$ indicates the cycle difference between i and j. In

these constraints, p_{ij} is also referred to as the modulo parameter of the constraint. The model now has become suitable for Mixed Integer Linear Programming (MIP) methods.

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

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

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

2.4 Cost optimization

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

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

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

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

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

2.5 Related work

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

Exact methods

A large part of the methods in the current literature focus on the PESP as a feasibility problem, rather than an optimization problem. One of the first solution methods in a railway timetabling context has been implemented by [15] by solving the Mixed Integer Linear Program (MILP) using constraints of the type as in Equation 3. With the aid of the

commercial optimization software package CPLEX, solutions for practical railway timetabling instances can be found with the aid of searching algorithms and adjustable parameters within the software package. Other papers that focus on solving the MILP can be found in [11], [12], [7] and [13], using cutting planes and similar other mathematical optimization techniques. A relevant approach, but different perspective is presented in [14], where feasible railway timetables can be found with minimal deviations from the original constraints in case no feasible timetable exists.

Heuristics

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

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

3 State- and search space reduction techniques

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

3.1 Intersecting feasible intervals

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

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

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

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

3.2 Eliminating variables

Variables can be eliminated in two ways.

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

3.3 Propagating constraints

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

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

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

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

4 The Restricted Periodic Event Scheduling Problem

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

4.1 Motivation

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

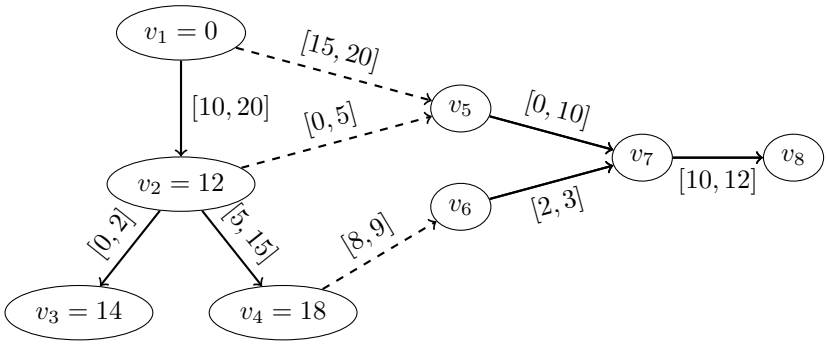

Figure 2 Example of restrictions while integrating components.

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

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

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

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

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

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

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

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

4.2 Problem description

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

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

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

Restricted Periodic Event Scheduling Problem (RPESP)	
Given:	A directed, cycle-free graph $D = (V, A)$, a cycle time T, a feasible interval $\Delta_{ij} \subseteq \{0, \dots, T-1\}$ for all $(i, j) \in A$ and variable restrictions $X_i \subseteq \{0, \dots, T-1\}$ for all $i \in V$.
Goal:	Find a $v \in [0, T)^n$ such $v_j - v_i \in \Delta_{ij}$ for all $(i, j) \in A$ and $v_i \in X_i$ for all $i \in V$, or state infeasibility.

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

4.3 Optimizing RPESP

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

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

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

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

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

with initialization for the leaves as:

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

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

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

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

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

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

5 Tree decomposition heuristics

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

5.1 Decomposing a PESP graph into trees

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

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

5.2 Requirements for partial solutions

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

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

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

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

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

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

5.3 Identifying non-extendable partial solutions

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

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

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

5.4 Fixing non-extendable partial solutions

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

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

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

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

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

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

6 Experimental results

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

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

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

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

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

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

7 Conclusions and future work

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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