



FELIX PRAUSE¹, RALF BORNDÖRFER², BORIS GRIMM,
ALEXANDER TESCH

Approximating the RSRP with Predictive Maintenance

¹  0000-0001-9401-3707

²  0000-0001-7223-9174

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: <http://www.zib.de>

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Approximating the RSRP with Predictive Maintenance

Felix Prause, Ralf Borndörfer, Boris Grimm, and Alexander Tesch

Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
{prause, borndoerfer, grimm, tesch}@zib.de

Abstract. We study the solution of the rolling stock rotation problem with predictive maintenance (RSRP-PM) by an iterative refinement approach that is based on a state-expanded event-graph. In this graph, the states are parameters of a failure distribution, and paths correspond to vehicle rotations with associated health state approximations. An optimal set of paths including maintenance can be computed by solving an integer linear program. Afterwards, the graph is refined and the procedure repeated. An associated linear program gives rise to a lower bound that can be used to determine the solution quality. Computational results for two instances derived from real world timetables of a German railway company are presented. The results show the effectiveness of the approach and the quality of the solutions.

1 Introduction

Scheduling the maintenance of rolling stock is one of the key tasks of every railway operator. The main objectives are to ensure the safety and the comfort of passengers, to save costs, and to satisfy all legal regulations. The progress in computing technology, the increasing availability of sensor data, and the recent theoretical advances in the field of machine learning allow, for the first time in history, to not only monitor and predict the health state of rolling stock, but also to use this information to develop methods for predictive maintenance in the railway sector. Tailor-made maintenance will ideally lead to a better health state at a lower cost.

Traditionally, preventive maintenance regimes, like time- or distance-dependent strategies, are applied in rolling stock rotation planning (RSRP), see, for example, Grimm et al. [6], Andrés et al. [1], and Giacco [5]. A first approach to predictive maintenance for rolling stock was presented by Herr et al. [9]. They propose a method for threshold-based maintenance considering the remaining useful lifetime (RUL) of the vehicles. They assume that the vehicles can be maintained at every station, and model the RUL by point estimates subject to a linear degradation behavior. Their goal is to jointly assign the vehicles to trips and to schedule their maintenance. Here, no costs are considered in the model and the objective is to maximize the minimal degradation value of the vehicles before maintenance. This is done by solving a mixed-integer linear program (MILP) based on a linearization of a quadratic program. A more general setting

involving sets of trips instead of individual trips is considered by Herr et al. [10]. The individual trips do not have to start and end at maintenance facilities, but only the first and the last trip of a set. Again, point estimates are used to model the RUL subject to a linear degradation behavior.

Bougacha [3] argues that the RUL should not be assumed to be a deterministic value, but rather be represented in terms of a probability distribution to reflect its uncertainty. In this vein, Bougacha et al. [4] improve the approach of Herr et al. [10] by assuming the health states to be uncertain and by modeling the degradation as a gamma process. Their main goal is to study and optimize the duration of the decision horizon for rolling stock planning. They propose a genetic algorithm as well as two other heuristics to solve the resulting non-linear formulation. The tasks that need to be operated are also sequences of trips starting and ending at a depot, and the maintenance gets scheduled whenever the predicted health values exceed some predefined threshold.

According to Wu and Lai [13], two-parameter Weibull distributions are suitable for modeling the RUL of rolling stock. They give a MILP formulation using bigM constraints for minimizing the costs of maintenance, and the expected cost of failures and of operation. The Weibull distribution is discretized by dividing the days since the last maintenance into intervals and assigning a number of expected failures to each. As the earlier mentioned authors, they aim at assigning sets of trips to vehicles, but their main focus lies on combining different types of maintenance. They show that combining time-based preventive and predictive maintenance reduces the total costs of operation, in their example by about 14%.

Finally, Rokhforoz and Fink [12] also state that point estimates are insufficient for modeling real world applications. They assume the RUL to be a random variable and use normal distributions to represent the uncertainty. They consider maintenance scheduling for railway wagons in a slightly different setting in which passenger demands have to be met and the goal is to maximize the revenue. Furthermore, the health states of the wagons are considered their private information leading to a non-cooperative game among them. In their approach, they use an exponential penalty function to enforce maintenance and a MILP formulation with bigM constraints to linearize a quadratic formulation. They solve the problem by applying a hierarchical learning approach.

We propose a novel approach to the RSRP with predictive maintenance (RSRP-PM) in this article. Its main feature is to treat the RUL by means of a propagation of distribution parameters. This makes it possible to deal with realistic large-scale scenarios, and it provides lower bounds that allow to assess the solution quality. As in the classical RSRP, we assign vehicles to trips and schedule their maintenance, but we additionally assume that the maintenance is scheduled depending on the health states of the vehicles; for a detailed introduction to RSRP, we refer to Reuther [11]. Furthermore, we assume that the health states are uncertain and represented by a probability distribution function (PDF), and that the degradation affecting the vehicles is randomly distributed. Different from the literature, the maintenance is planned w.r.t. the failure probability of the vehicles instead of applying thresholds. Additionally, we assume that each

trip can be operated individually and is not restricted to begin or end at a depot or maintenance facility, in order to avoid restricting the solution space. Finally, non-linear degradation behavior should not be excluded. Indeed, predictions on the degradation of mechanical components are always uncertain, and usually possess a non-linear degradation pattern, see Heng et al. [8]. A major problem is to propagate the random variables representing the health states of the vehicles. This becomes complicated when considering non-linear degradation functions or non-stable distributions. To overcome these problems, we propose the use of a state-expanded event-graph (SEG) approximating the original problem and give an integer linear programming (ILP) formulation for solving the resulting approximation. Then, by iteratively refining model, we are able to obtain closer and closer approximations of the original RSRP-PM instance.

2 Problem Formulation

In this section, we give a mathematical definition of RSRP-PM. First, let Π_Θ denote a parametric family of PDFs with parameter space Θ , i.e., for each $\theta \in \Theta$ we obtain a PDF of Π_Θ . An example would be the the parametric family of normal distributions with fixed variance, e.g., $\mathcal{N}(\mu, \sigma^2)$ with parameter space $\{(\mu, \sigma^2) \mid \mu \in \mathbb{R}, \sigma^2 = 0.1\}$. Note that we assume Π_Θ to be a one-parameter family in the following, i.e., $\Theta \subseteq \mathbb{R}$.

Next, let \mathcal{V} be a set of vehicles. Each vehicle $v \in \mathcal{V}$ has an initial health status that is a random variable distributed by a PDF from Π_Θ , which can be characterized by its parameter $\theta_v^0 \in \Theta$. We assume that the health states take values in $[0, 1]$, where zero indicates a status that is as good as new, and one indicates a completely deteriorated component. Furthermore, we are given a timetable or set of trips \mathcal{T} , where each trip $t \in \mathcal{T}$ has its individual start and end location, departure and arrival time, and a monotonic increasing degradation function Δ_t determining how the health value parameter of the vehicle operating this trip gets altered. Next, \mathcal{L} is the set of all occurring locations with its subset of maintenance locations $\mathcal{L}_M \subseteq \mathcal{L}$. We further associate costs with different operations: operation costs for each vehicle that is used in the solution, trip costs associated with the operation of the given trips, deadhead costs that occur when a vehicle is moved from one location to another, maintenance costs, and failure costs occurring when a vehicle breaks down. Finally, we demand the balancedness of the vehicle rotations meaning that the numbers of vehicles located at each location $l \in \mathcal{L}$, at the start and the end of the considered time horizon, need to coincide.

Given these definitions, the task of RSRP-PM is as follows: We need to assign sequences of trips, deadhead trips, and maintenance operations to the vehicles s.t. all given trips are operated. Furthermore, we want the assignment to be balanced, and to have minimal total costs. Here, the total costs consist of operation costs, trip costs, deadhead costs, maintenance costs, and expected failure costs, i.e., the product of the failure costs with the probability of breakdowns occurring.

3 The State-Expanded Event-Graph

After defining RSRP-PM, we present a state-expanded version of an event-graph (SEG) that provides an approximation to the problem. This approximation will then serve as a basis for formulating an ILP that is used throughout the solution process.

3.1 Nodes and Layers

First, we generate the nodes of the SEG. To this purpose, we construct layers, where each layer represents another parameter value for the same set of events. Note that this corresponds to considering a discretization of the parameter values. For example, if we consider a real-valued parameter $\theta \in [0, 1]$, $\mathcal{D} = \{0, 0.1, \dots, 0.9, 1\}$ could be such a discretization, giving rise to a graph with eleven layers.

Considering an individual layer, a node $v := (l, t, \theta)$ is defined by a location $l \in \mathcal{L}$, a time point t within the time horizon, and the parameter value θ of its layer. In order to obtain the set of all nodes of a layer, we iterate over all trips and create a node for its start location and departure time, as well as for its end location and arrival time. Additionally, we add start and end nodes for each location, i.e., $(l, 0, \theta)$ and (l, ∞, θ) for all $l \in \mathcal{L}$. This construction is repeated for each parameter value of the discretization. Furthermore, the set of all nodes is denoted by V , while $V(\theta)$ denotes the set of all nodes of the layer associated with parameter value θ . Additionally, we define the set of nodes corresponding to a location l_c and a time t_c to be $V(l_c, t_c) := \{(l, t, \theta) \in V \mid l = l_c, t = t_c, \theta \in \mathcal{D}\}$.

Next, we describe how to construct the arcs of the graph. They can be subdivided into layer internal arcs and arcs between layers. Here, layer internal arcs connect nodes with equal parameter value, while the nodes connected by arcs between layers have differing parameters. Note that all arcs are directed.

3.2 Layer Internal Arcs

We start with the layer internal arcs. As already mentioned, the incident nodes of these arcs have the same parameter value, i.e., traversing them does not alter the parameter value and hence no degradation occurs.

First, we add arcs representing vehicles just waiting at their current locations. To this purpose, we sort all nodes of each location by time and add an arc between each pair of consecutive nodes. This yields a timeline for each location $l \in \mathcal{L}$, starting with $(l, 0, \theta)$ and ending with (l, ∞, θ) .

Next, we add arcs corresponding to possible deadhead trips between differing locations. These can be obtained by iterating over all nodes $v \in V(\theta)$ of the current layer and determining the nodes on the other locations' timelines with a time point that is less than or equal to the time point of v minus the necessary travel time. Afterwards, we add an arc from the latest of these nodes to v .

3.3 Arcs Between Layers

After adding the arcs that do not change the parameter values and thus stay within the same layer, we add the operations that can alter these values, i.e., trips and maintenance services.

First, we generate the trip arcs. Let l_d and t_d be the departure location and time, and l_a and t_a be the arrival location and time of trip $s \in \mathcal{T}$. Then, we iterate over all corresponding departure nodes, i.e., nodes in $\{(l_d, t_d, \theta) \mid \theta \in \mathcal{D}\}$, and apply the degradation function of trip s to their parameter values. Let $v_c = (l_d, t_d, \theta_c)$ be one of these departure nodes, then the parameter value after operating s is $\theta_{new} = \Delta_s(\theta_c)$. But θ_{new} does not need to be contained in \mathcal{D} . Therefore, we need to map it to one of its values or decide that there should not be a trip arc outgoing of the considered departure node v_c . This can be done by rounding θ_{new} to the nearest value in \mathcal{D} by applying either $\lceil \cdot \rceil_{\mathcal{D}}$ or $\lfloor \cdot \rfloor_{\mathcal{D}}$ defined as follows:

$$\lceil \theta \rceil_{\mathcal{D}} := \begin{cases} \emptyset & \text{if } \{x \in \mathcal{D} \mid x \geq \theta\} = \emptyset \\ \min\{x \in \mathcal{D} \mid x \geq \theta\} & \text{else} \end{cases}$$

$$\lfloor \theta \rfloor_{\mathcal{D}} := \begin{cases} \emptyset & \text{if } \{x \in \mathcal{D} \mid x \leq \theta\} = \emptyset \\ \max\{x \in \mathcal{D} \mid x \leq \theta\} & \text{else} \end{cases}$$

If we decide to build the graph by using $\lceil \cdot \rceil_{\mathcal{D}}$, we apply this function to the updated parameter θ_{new} . If this value is an element of \mathcal{D} , e.g., $\theta_{to} = \lceil \theta_{new} \rceil_{\mathcal{D}} \in \mathcal{D}$, we add a trip arc between the respective departure and arrival nodes, i.e., $v_c = (l_d, t_d, \theta_c)$ and $v_{to} = (l_a, t_a, \theta_{to})$, otherwise we omit adding an arc. This procedure is repeated for all trips $s \in \mathcal{T}$ and all considered parameter values $\theta \in \mathcal{D}$. Note that our graph is constructed by consistently applying either $\lceil \cdot \rceil_{\mathcal{D}}$ or $\lfloor \cdot \rfloor_{\mathcal{D}}$.

Next, we add the maintenance arcs. These arcs reset the parameters to some predefined value and can be seen as replenishment. Let therefore θ_M be the parameter value after maintenance, and define $\theta_m := \lceil \theta_M \rceil_{\mathcal{D}}$. Then, we iterate over all tuples (l, t) for that a node exists and over all maintenance locations $m \in \mathcal{L}_M$. For each combination, we add a new node $v_M = (m, t + t_{l,m} + t_M, \theta_m)$ corresponding to the maintenance service. Here, $t_{l,m}$ is the travel time from l to m and t_M is the duration of the service. Afterwards, we add an arc from all nodes corresponding to location l and time t , and any choice of $\theta \in \mathcal{D}$, i.e., all nodes of $V(l, t)$, to the just generated maintenance node v_M . As a final step, we add the layer internal arcs, i.e., deadhead and waiting arcs, outgoing of v_M as described above.

Finally, we add artificial nodes and arcs for ensuring the balancedness of the vehicle rotations and to incorporate the initial health states of the vehicles. Therefore, we add a node v_l for any location where a vehicle is located. Afterwards, for each of these vehicles we add an arc from v_l to $(l, 0, \lceil \theta_v^0 \rceil_{\mathcal{D}})$, where θ_v^0 is the parameter value of the initial health state of v . Similarly, we add a node for each location $l \in \mathcal{L}$ symbolizing the end of the time horizon, and add an arc from every node (l, ∞, θ) , for all $\theta \in \mathcal{D}$, to this node.

Generating and collecting all the nodes in V and arcs in A as described above, yields the SEG $G = (V, A)$. Note that G is generated either completely w.r.t. $\lceil \cdot \rceil_{\mathcal{D}}$ or completely w.r.t. $\lfloor \cdot \rfloor_{\mathcal{D}}$, and is always dependent on an initial discretization \mathcal{D} . Finally, we want to highlight that also non-linear degradation functions can be used during the construction of the presented graph. This gives us the possibility to address and incorporate a greater variety of degradation behaviors, which is advantageous over direct MIP formulations of the problem. Additionally, it should be mentioned that the presented graph can be easily adapted for distance- or time-based preventive maintenance scenarios using thresholds. In such applications, one chooses the parameter value to represent distance or time, ensures that the greatest value of the discretization is less than or equal to the desired threshold, and defines the update function of the trips to sum the already traveled distance or time with the value associated to the trip.

3.4 Example

An illustrative example for a graph resulting from this construction procedure, based on the trips given in Table 1, can be found in Figure 1. Here, we assume that maintenance takes three hours, that the considered time horizon is from 00:00 to 23:59, and that a deadhead trip between A and B takes one hour. Furthermore, the trips are assumed to increase the parameter values by one, while the maintenance operations completely reset them to zero. Additionally, maintenance operations are possible at both locations. Finally, we assume both vehicles to have an initial parameter value of zero, one vehicle to be located at A, and the other one to be located at B.

Start	End	From	To
00:00	09:00	A	B
11:00	20:00	B	A

Table 1. Two exemplary trips.

3.5 Arc Costs

Finally, we need to determine the costs of each arc $a \in A$. The first type of arc that we want to consider are the waiting arcs, which we assign arc costs of zero. Next, the deadhead arcs are associated with costs depending on the traveled distance between their start and end location. The costs of the artificial arcs heading to and from the artificial start and end nodes are determined like the costs of deadhead trips, but we additionally add the operation costs of a vehicle to the starting arcs. The two types of maintenance arcs, i.e., the arcs heading to a maintenance node and the arcs pointing away from them get assigned different costs. The first one gets assigned the sum of the maintenance costs plus the

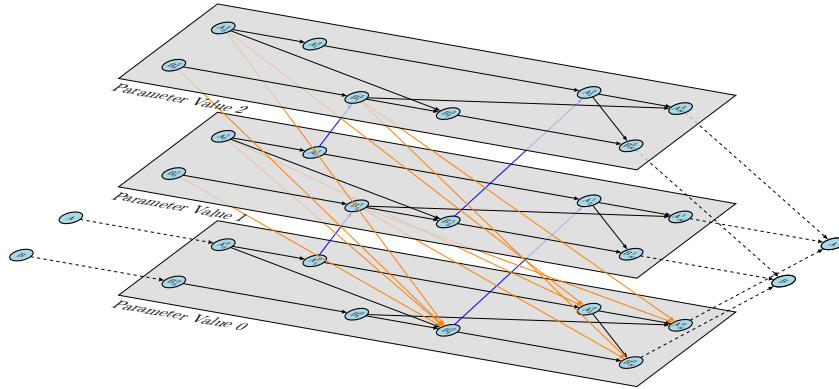


Fig. 1. Illustration of a SEG w.r.t. the exemplary trips given in Table 1. The different layers are shaded in gray, and their height corresponds to the value of their parameters. Here, the black arcs represent waiting or deadheads, the blue arcs represent the trips, and the orange arcs represent the maintenance arcs that reset the parameter value to zero. Furthermore, the dashed arcs are the artificial ones necessary for enforcing the balancedness and setting the initial parameter values of the vehicles. Note that we combined each pair of consecutive maintenance arcs, i.e., the arc heading to the maintenance node and the arc leaving it, into one arc for reasons of presentability.

deadhead costs between the start and the end location of the arc, while the second one gets assigned only the deadhead costs from the maintenance facility to the arrival location.

The trip costs also consist of two different sub-costs. First, we assume costs that are inherent to each trip and which might be time- or distance-dependent as well. Furthermore, we are going to consider the expected failure costs, i.e., costs that are associated with a possible breakdown of the vehicle. These costs depend on the health states of the vehicles, and thus on the parameters associated with them since these characterize the PDF of the potential failure. Thus, in order to determine the failure probability, we need to determine the cumulative distribution function (CDF) of the random variable depicting the vehicles health status. Therefore, we have to determine an integral of the PDF depending on the parameters as follows:

$$\mathbb{P}[v \text{ has a failure}] = \mathbb{P}[H_v > 1] = 1 - \mathbb{P}[H_v \leq 1] = 1 - \int_{-\infty}^1 \Pi_\theta(x) dx =: \mathbb{P}_f(\theta) \quad (1)$$

where H_v is the random variable representing the vehicle's health status. Here, a health value of zero is assumed to indicate a status that is as good as new, while one depicts the maximal degradation of vehicle. Recall that the random variable H_v is distributed by the vehicle's PDF $\Pi_\theta \in \Pi_\Theta$, and that this PDF is characterized by the parameter θ .

Regarding our graph, this means that a trip arc $a = (v_1, v_2)$ connecting $v_1 = (l_1, t_1, \theta_1)$ and $v_2 = (l_2, t_2, \theta_2)$ takes a vehicle, which health status is a random variable distributed by PDF Π_{θ_1} and alters the health state s.t. it is a random variable distributed by Π_{θ_2} when the trip is finished. Thus, the parameter value of the arrival node of a trip characterizes the failure probability of a vehicle during the trip. Hence, the expected failure costs can be determined by multiplying the costs associated with a failure with $\mathbb{P}_f(\theta_2)$. We assume the failure probability depending on the parameter θ , i.e., $\mathbb{P}_f(\theta)$, to be a monotonic increasing function in the following. Note that the presented approach has the advantage of determining the failure probability of the trips a priori when constructing the SEG, and does not have to evaluate \mathbb{P}_f in the objective function like non-parameter-expanded MIP formulations.

4 Algorithm

In the following, we propose an algorithm for solving RSRP-PM. Our approach is based on solving the problem induced by the graph-theoretic approximation presented in the previous section. Here, a solution of RSRP-PM corresponds to a set of paths in the approximating SEG covering each trip exactly once. Furthermore, each path has to start and end at an artificial node, and the goal is to find a set with minimal costs.

4.1 An Exact ILP Formulation for the Approximate Problem

Suppose we are given an instance of RSRP-PM, and have constructed an approximating SEG $G = (V, A)$ w.r.t. a discretization \mathcal{D} and one of the presented rounding functions, i.e., $\lceil \cdot \rceil_{\mathcal{D}}$ or $\lfloor \cdot \rfloor_{\mathcal{D}}$. Then, a solution to the approximate problem can be determined by solving the following ILP formulation (AP). The formulation uses the variables, parameters, and sets given in Table 2.

$$(AP) \quad \min \quad \sum_{a \in A} c_a x_a \quad (2)$$

$$\text{s.t.} \quad \sum_{a \in A(s)} x_a = 1 \quad \forall s \in \mathcal{T} \quad (3)$$

$$\sum_{a \in \delta^{in}(v)} x_a = \sum_{a \in \delta^{out}(v)} x_a \quad \forall v \in V \setminus V_{art} \quad (4)$$

$$\sum_{a \in \delta^{out}(v_0(l))} x_a = \sum_{a \in \delta^{in}(v_\infty(l))} x_a \quad \forall l \in \mathcal{L} \quad (5)$$

$$x_a \in \mathbb{Z}_{\geq 0} \quad \forall a \in A \setminus A_{art}^0 \quad (6)$$

$$x_a \in \{0, 1\} \quad \forall a \in A_{art}^0. \quad (7)$$

Object	Description
$x_a \in \mathbb{Z}_{\geq 0}$	Integer variable determining how many vehicles are using arc $a \in A$
$c_a \in \mathbb{R}_{\geq 0}$	Cost parameter of arc $a \in A$
$\delta^{in}(v)$	Incoming arcs of node $v \in V$
$\delta^{out}(v)$	Outgoing arcs of node $v \in V$
$v_0(l)$	Artificial start node corresponding to location $l \in \mathcal{L}$
$v_\infty(l)$	Artificial end node corresponding to location $l \in \mathcal{L}$
A	Set of all arcs of the graph
A_{art}^0	Set of artificial start arcs
$A(s)$	Set of all arcs that correspond to trip $s \in \mathcal{T}$
\mathcal{T}	Set of all trips of instance
V	Set of all nodes of the graph
V_{art}	Set of all artificial nodes, i.e., start and end nodes
\mathcal{L}	Set of all locations of the instance

Table 2. Variables, parameters, and sets used in ILP formulation (AP).

The objective function (2) aims at minimizing the total costs, i.e., the sum of deadhead, operation, maintenance, trip operation, and expected failure costs. Constraints (3) ensure that each trip is operated exactly once, while (4) guarantee the flow conservation, i.e., that any vehicle entering a node also leaves it. Finally, the balancedness, i.e., the fact that the same amount of vehicles is located at each location at the beginning and the end of the considered time horizon, is ensured by constraints (5), while (6) and (7) define the domains of the arc variables.

Lemma 1. *Let \mathcal{I} be a RSRP-PM instance and G be a SEG for \mathcal{I} based on $\lceil \cdot \rceil_{\mathcal{D}}$ and any discretization \mathcal{D} . Furthermore, let (AP) be the ILP formulation for the approximate problem given by G . Then any incumbent or solution of (AP) is feasible w.r.t. to \mathcal{I} .*

Lemma 1 relies on the observation that all parameter values get overestimated by applying $\lceil \cdot \rceil_{\mathcal{D}}$. Hence, some solutions of the original problem might be eliminated because the degradation of the vehicles is overestimated. Thus, the vehicles would get maintained before it is necessary, but no new feasible solutions are added, unlike when considering the SEG w.r.t. $\lfloor \cdot \rfloor_{\mathcal{D}}$, where the parameters of the health states get underestimated.

4.2 An ILP-Based Refinement Heuristic

The construction of the SEG involves roundings of parameter values to the considered discretization, which produces approximation errors. Thus, the solutions to (AP), for the induced approximate problem represented by the SEG, are not necessarily “good” solutions for the original RSRP-PM instance. However, the errors can be reduced by iteratively refining the discretization and thus increasing the approximation quality of the solution. In this vein, we propose the following solution method for RSRP-PM relying on Lemma 1.

It is plausible that the initial discretization determines the complexity of the instance since the size of the resulting ILP formulation depends on the number of vertices and arcs contained in G . Hence, we start with a coarse discretization \mathcal{D} leading to a probably inaccurate approximation of the problem. Afterwards, we refine \mathcal{D} and repeat this procedure until no further improvement is made or a given time limit is met, see Algorithm 1. Within this algorithm, $\underline{\theta} \in \mathbb{R}$ and $\bar{\theta} \in \mathbb{R}$ are a lower and an upper bound on the parameter values of the initial discretization, with $\underline{\theta} < \bar{\theta}$, and $s \in \mathbb{R}_{>0}$ is its step-size. Additionally, $\lambda \in (0, 1)$ is a decay parameter for the step-size, and $i_{\max} \in \mathbb{Z}_{>0}$ determines how many iterations without improvement are allowed.

Note that the value of a heuristically generated solution x w.r.t. the original RSRP-PM instance, i.e., $v(x)$, can be obtained by simply tracking the paths in G given by x . These paths correspond to the schedules of the vehicles, and if we apply and propagate the original parameter values along them, we determine their values w.r.t. the original instance.

Algorithm 1: Heuristic for RSRP-PM

Data: RSRP-PM instance, $\underline{\theta}$, $\bar{\theta}$, s , λ , i_{\max}
Result: Solution to the given RSRP-PM

```

1  $x \leftarrow \infty$ 
2  $i \leftarrow 0$ 
3 while time limit not reached and  $i < i_{\max}$  do
4    $\mathcal{D} \leftarrow \{\underline{\theta}, \underline{\theta} + s, \underline{\theta} + 2s, \dots, \bar{\theta}\}$ 
5    $G \leftarrow$  SEG based on  $\mathcal{D}$  w.r.t.  $\lceil \cdot \rceil_{\mathcal{D}}$ 
6    $c \leftarrow$  solution of (AP) for  $G$  and  $\mathcal{D}$ 
7   if  $v(c) < v(x)$  then
8      $x \leftarrow c$ 
9      $i \leftarrow 0$ 
10  else
11     $i \leftarrow i + 1$ 
12  end
13   $s \leftarrow \lambda s$ 
14 end
15 return  $x$ 

```

4.3 Bounds

Consider an instance of RSRP-PM and any approximating SEG G . Then, the costs associated with any operation, i.e., with any arc, are equal except the expected failure costs since these depend on the parameter values associated with the trips. During the construction of G w.r.t. to either $\lceil \cdot \rceil_{\mathcal{D}}$ or $\lfloor \cdot \rfloor_{\mathcal{D}}$, the parameter values get rounded and thus, the approximation always over- or underestimates them compared to their values in the original RSRP-PM instance. But the false estimations are all done consistently.

Since we consider a monotonic increasing failure probability \mathbb{P}_f , the solution of (AP) always over- or underestimates this probability, and hence the failure costs. Therefore, the value of an optimal solution to (AP) gives either an upper or a lower bound on the objective value of the original instance. Note that a similar approach is also possible when other PDFs are considered. Then, $\lceil \cdot \rceil_{\mathcal{D}}$ and $\lfloor \cdot \rfloor_{\mathcal{D}}$ have to be modified s.t. they round the parameter values in the direction of $\nabla \mathbb{P}_f$ and $-\nabla \mathbb{P}_f$, respectively.

Thus, independent of the choice of \mathcal{D} , we are able to obtain an upper bound on the optimal objective value of a RSRP-PM instance by determining any solution to (AP), for a graph constructed w.r.t. $\lceil \cdot \rceil_{\mathcal{D}}$. Analogously, a lower bound can be determined by an optimal solution to (AP), for a graph constructed w.r.t. $\lfloor \cdot \rfloor_{\mathcal{D}}$. Furthermore, since we are solving a minimization problem, the value of an optimal solution to the linear program (LP) relaxation of (AP), i.e., $(\text{AP})_{\text{LP}}$, gives a lower bound on the value of an optimal solution of (AP). Hence, we are able to obtain a second lower bound by an optimal solution to $(\text{AP})_{\text{LP}}$, for a graph constructed w.r.t. $\lfloor \cdot \rfloor_{\mathcal{D}}$. These bounds will be abbreviated by $\text{UB}_{(\text{AP})}$, $\text{LB}_{(\text{AP})}$, and $\text{LB}_{(\text{AP})\text{-LP}}$, and the following proposition applies.

Proposition 1. *Let x^* be an optimal solution to a given RSRP-PM instance, then it holds*

$$\text{UB}_{(\text{AP})} \geq x^* \geq \text{LB}_{(\text{AP})} \geq \text{LB}_{(\text{AP})\text{-LP}}$$

Note that the tightness of the bounds does not have to increase with the granularity of the applied discretization \mathcal{D} . Finally, an approach for determining lower bounds on an RSRP-PM instance can be derived by modifying the just given heuristic. The resulting approach is given by Algorithm 2 and uses the same parameters as Algorithm 1.

5 Computational Results

In this section, we describe the setup and the test instances that we used to evaluate our iterative refinement heuristic and our lower bounds. Afterwards, we report on the results we obtained by applying both algorithms.

5.1 Computational Setup

We ran all our computations on a machine with Intel(R) Xeon(R) Gold 5122 @ 3.60GHz CPUs, eight cores, and 32GB of RAM. All algorithms were implemented with Julia v1.8.2 using Gurobi v9.5.1 as internal MIP solver, see [2] and [7]. Note that we set the focus of Gurobi on finding feasible solutions, chose the barrier method for solving the continuous models, and set the amount of time spend in MIP heuristics to 10%.

Algorithm 2: Lower Bound Heuristic for RSRP-PM

Data: RSRP-PM instance, $\underline{\theta}$, $\bar{\theta}$, s , λ , i_{\max}
Result: Lower bound for the given RSRP-PM

```

1  $lb \leftarrow 0$ 
2  $i \leftarrow 0$ 
3 while time limit not reached and  $i < i_{\max}$  do
4    $\mathcal{D} \leftarrow \{\underline{\theta}, \underline{\theta} + s, \underline{\theta} + 2s, \dots, \bar{\theta}\}$ 
5    $G \leftarrow$  SEG based on  $\mathcal{D}$  w.r.t.  $\lfloor \cdot \rfloor_{\mathcal{D}}$ 
6    $c \leftarrow$  optimal objective value of  $(AP)_{LP}$  for  $G$  and  $\mathcal{D}$ 
7   if  $c > lb$  then
8      $lb \leftarrow c$ 
9      $i \leftarrow 0$ 
10  else
11     $i \leftarrow i + 1$ 
12  end
13   $s \leftarrow \lambda s$ 
14 end
15 return  $lb$ 

```

5.2 Test Instances

The test instances that we used to conduct our computations are based on real timetables of a private German railroad company, and thus give realistic scenarios for testing our approaches. In the following, we describe the assumptions we made concerning the instances, in order to determine meaningful examples for predictive maintenance of non-safety relevant technology. Thus, we will consider components that are not subject to official maintenance regulations. A failure of such a component would cause additional costs related to the damage that needs to be repaired and to the caused inconvenience but would not lead to an abortion of a trip. As an example, we have therefore chosen the doors of the train.

Hence, we assume the degradation parameter of each trip $t \in \mathcal{T}$ to depend on the number of opening-closing-cycles of the train doors. We suppose that, at each stop of t , the number of these cycles is an integer uniformly taken from $\{1, \dots, 4\}$, i.e., $\mathcal{U}\{1, 4\}$. Furthermore, we assume that each door can complete about 500 of these cycles on average before a first failure occurs. Thus, the parameter value associated with a trip containing n stops is $\frac{n}{500} \cdot \sum_{i=1}^n X_i$, where $X_i \stackrel{\text{iid}}{\sim} \mathcal{U}\{1, 4\}$. Next, we assume the degradation function to be $\Delta_t(\theta_v) = \theta_v + \alpha \cdot \theta_t$, where θ_v is the parameter value of the vehicle before the operation of the trip and θ_t is the parameter value corresponding to the degradation of t . Additionally, $\alpha = 1.02$ is an aging coefficient depicting the fact that used components deteriorate faster than new ones. Maintenance is assumed to reset the parameter value to $\theta = 0.05$ and we further assume that also the initial parameters of the health states are equal to $\theta = 0.05$ for all vehicles. Furthermore, the family of normal distributions with variance $\sigma^2 = 0.1$, i.e., $\mathcal{N}(\theta, 0.1)$, will be considered as the

parametric family of PDFs in the following. Note that using this PDF leads to a failure probability function \mathbb{P}_f that is monotonic increasing. Finally, we use the discretization $\mathcal{D} = \{0, s, 2s, \dots, 1\} \cup \{0.05\}$ depending on step-size s , where 0.05 is added to ensure that the parameter values after maintenance as well as the initial values are already contained within \mathcal{D} .

The two timetables considered in the following are taken from publicly available sources of a private German railroad company. The RSRP-PM instances derived from these timetables, are abbreviated by T and R and consist of 180 and 585 trips, respectively. All of their trips take place within one week and are operated between 18 locations, of that three are maintenance facilities. Additionally, they possess 9 and 17 vehicles, respectively.

Furthermore, the deadhead costs are assumed to be 6 units per kilometer, the trip costs are 3 units per kilometer, and the maintenance costs are 2,000 units at two of the workshops and 1,000 units at the third one. Finally, the failure costs are 50,000 units per breakdown and the operation costs are 3,300,000 units per year and vehicle, i.e., 63,288 units for using a vehicle in the considered time horizon of one week.

5.3 Results

Throughout this section, we present, describe, and evaluate the computational experiments we conducted on the just introduced test instances for RSRP-PM. Therefore, we applied the heuristic and the lower bound approach, see Algorithm 1 and 2, with different instantiations of s and λ . The detailed results can be found in Tables 3 - 6 given in the Appendix.

Lower Bounds First, we describe the results obtained by the lower bound approach presented in Section 4.3. Therefore, we ran Algorithm 2 using different instantiations of the step-size $s \in \mathbb{R}_{>0}$ and the decay parameter $\lambda \in (0, 1)$. All computations had a time limit of one hour and the results can be found in Tables 3 and 4. Throughout these tables, the first two columns contain the values of the step-size s and the decay parameter λ used for instantiating the approach. The third column contains the best obtained lower bound by the considered instantiation. In the fourth one, the gap of the current lower bound to LB_{\max} is given in percent, where LB_{\max} is the best obtained lower bound among all choices of s and λ . Finally, the last two columns contain the time required for obtaining the lower bound, and the time to obtain an ε -good bound, i.e., a bound that has a value of at least $(1 - \varepsilon) \text{LB}_{\max}$. Note that the values of the lower bounds as well as the computation times are rounded to the nearest integer. Furthermore, the best bound is marked in bold, and the runs marked with an asterisk are stopped due to an excess of the provided memory. Whenever we occur a 3,600 in the column of computation time, the current LP was stopped and its best lower bound was taken as a solution.

The detailed results for instance T can be found in Table 3. For this instance, we obtained a best lower bound with value $\text{LB}_{\max} = 1,100,971$, while the worst

lower bound determined by any instantiation has a value of 1,099,944. Therefore, the gap among all runs is only 0.1%. Here, more aggressive decay, i.e., smaller choices of λ , seem to have a good influence on the performance of the approach since the instantiations using $\lambda \in \{0.8, 0.9\}$ were never able to obtain a result with value LB_{\max} . Furthermore, four of the five best results are obtained by choosing an initial step-size of $s \in \{0.5, 0.6, 0.7\}$. However, seven of the eight memory exceeds are caused by these small choices of s since the resulting graph approximations get too fine too fast resulting in computationally intractable LPs. Nevertheless, all instantiations were able to obtain high quality lower bounds within short time since all determined a lower bound with value greater than 99.5% of LB_{\max} in under 250 seconds. The development of the lower bound over time, for instantiations with step-size $s = 0.07$, are visualized by the graphs coming from below in Figure 2.

Next, we describe the results w.r.t. instance R , which can be found in Table 4. For this instance, we obtained a best lower bound with value $LB_{\max} = 2,889,913$. Here, the lower bound determined by the worst performing instantiation has a value of 2,860,158 having a gap of 1%. Choosing $\lambda \in \{0.6, 0.7, 0.8\}$ leads to the fastest determination of a solution with a value of 98% of LB_{\max} or higher, and also the best bounds are obtained by using a medium aggressive decay, i.e., $\lambda = 0.7$. Applying this choice of λ , the time to ε -good solutions is lower than 950 seconds, i.e., 16 minutes. Next, choosing $\lambda = 0.9$ resulted, except for a step-size of $s = 0.07$, in the slowest approaches, and applying $\lambda \in \{0.5, 0.6\}$ almost always led to an exceed of the memory limit causing ten out of 13 excesses. Furthermore, the step-size does not seem to have such a big influence on the quality of the lower bound than previously. Nevertheless, the best results are obtained when a bigger step-size like 0.1 or a medium step-size like 0.07 is applied. Since these choices lead to coarser discretizations, we obtain smaller graph approximations resulting in smaller LP formulations. However, the results generated by $s = 0.06$ are the best on average.

Summarizing, the influence of the instantiations behaves as expected since the parameters s and λ should be chosen as fine as possible to speed up the computation of high quality lower bounds. Nevertheless, the parameters should also be increased with an increasing size of the considered instances in order to avoid exceeding the memory limit, due to the size of the LP formulations resulting from small choices of the parameters.

Heuristic Solutions After presenting the obtained lower bounds, we discuss the results determined by the heuristic described in Section 4.2. Therefore, we applied Algorithm 1 to both test instances using different choices of the step-size $s \in \mathbb{R}_{>0}$ and the decay parameter $\lambda \in (0, 1)$. Again, a time limit of one hour was used and the results can be found in Tables 5 and 6. As previously, the first two columns of these tables contain the choices of s and λ , while the third and fourth ones contain the best obtained solution as well as the gap of the current best solution to x_{best} , which is the best solution obtained among all instantiations. Next, the fifth column contains the time required to determine the best solution,

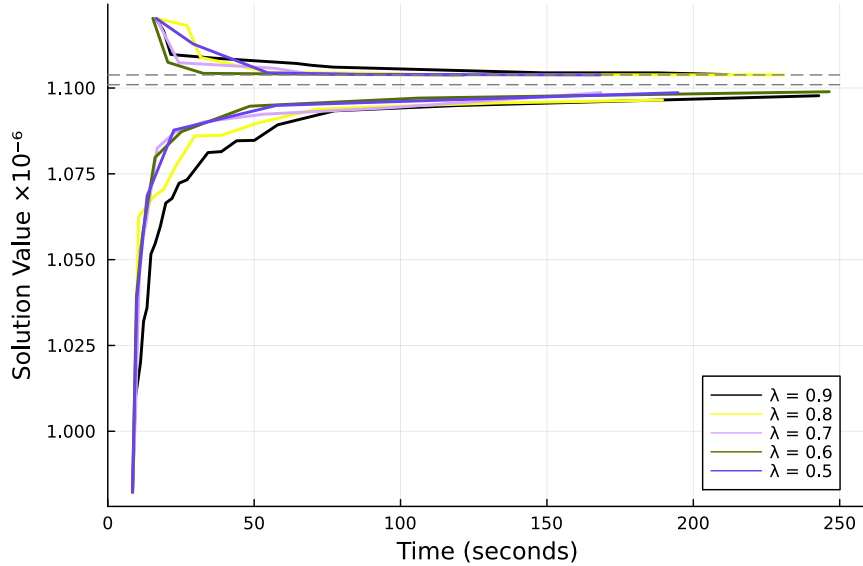


Fig. 2. Results obtained by the heuristic and the lower bound approach for instance T during the first 250 seconds, for a step-size of $s = 0.07$. The dashed gray lines represent the best obtained lower bound and the best obtained heuristic result, respectively.

and the sixth one contains the number of generated solutions. Furthermore, the table containing the results of instance T has an additional seventh column containing the time required to obtain an ε -good solution, i.e., a solution with value at most $(1 + \varepsilon)x_{best}$. This column was not added for instance R since many instantiations obtained only one solution. Additionally, a computation time of 3,600 seconds indicates that the currently active ILP solution process was stopped and the best incumbent was taken as solution. Finally, note that we used the best obtained solution so far as an initial solution for future iterations.

First, we comment on the results for instance T , which can be found in Table 5. Here, all instantiations obtained the same result with value $x_{best} = 1,103,819$. Thus, we decided to additionally evaluate the time until an ε -good solution, for $\varepsilon = 0.1\%$, is found. Obviously, the instantiations finding good solutions early on, are the ones with the smallest solution pools. The fastest instantiations obtaining ε -good results are the ones using $\lambda \in \{0.5, 0.6, 0.7\}$, while the ones that determine x_{best} first rely on $\lambda \in \{0.6, 0.7, 0.8\}$. Here, the biggest choice of $\lambda = 0.9$ leads to slow convergence towards x_{best} because the solution space is explored more precisely. This is an expectable behavior since the discretization is refined to a lesser extent and thus more often compared with the more aggressive choices of λ . On the other hand, choosing $\lambda = 0.5$ leads to a memory excess in four out of six cases, and especially the combination of a small step-size $s = 0.05$ with aggressive decay can result in exceeding the memory

after already 55 seconds. Here, the discretizations get too fine too fast resulting in big ILP formulations. Apart from this, the choice of s does not have a great influence on the performance. A quite surprising observation is that the biggest solution pool is obtained by the choice of $(s, \lambda) = (0.07, 0.9)$ although the choice of $(0.1, 0.9)$ should have converged more slowly to a discretization with a step-size approaching zero, and thus should have led to more solutions. This could be the case since a solution obtained at an early stage by the latter one tends to be optimal w.r.t. the following generated discretizations. Therefore, the solution processes of the individual ILPs are faster and a smaller solution pool is generated. An illustration for the trajectories of the heuristic's results can be found in the graph coming from above in Figure 2, where different instantiations using a step-size of $s = 0.07$ are depicted.

Next, we evaluate the solutions obtained for instance R , see Table 6. Here, we want to highlight the relationship between the number of determined solutions and the time until the best solution is obtained. Whenever only one solution is found, it is always found before the time limit is met. Therefore, we conclude that the ILP formulation generated during the second iteration, after refining the initial discretization, is not able to obtain an incumbent with better solution value before reaching the time limit. Hence, the second ILP seems to be computationally intractable in these cases. But in the cases where two or three solutions are found, the best solution is always the incumbent of the last ILP run. This can be seen as these solutions are all obtained after 3,600 seconds, i.e., when the last run was aborted. This indicates that the considered ILPs can be solved efficiently, which is especially the case for instantiations with $s \in \{0.06, 0.07\}$, i.e., small choices of s . Furthermore, the decay parameter λ does not have a big effect when solving instance R since the discretization gets refined at most twice. Hence, the instantiations starting with a small step-size are advantageous since their approximation errors tend to be smaller. Since we observe at most three solutions, and do not find any memory excesses, it seems to be meaningful to combine a small initial step-size with an aggressive choice of λ when solving the bigger instance R . However, the best solution overall is obtained by applying $\lambda = 0.8$.

Summarizing, and in comparison to the results of the lower bound, we can state that the heuristic does not exceed the memory limit since it is much harder to solve the ILP than the LP. Therefore, more time is spent for solving the formulations, and only one to three iterations are made for instance R and four to 14 for instance T . Hence, it seems to be reasonable to use a small or medium-sized decay parameter λ together with a step-size that is decreasing with the increasing size of the instance. Hereby, we are able to obtain close approximations without too many refinement steps.

Combined Results After discussing the results obtained by the heuristic and by the lower bound approach separately, they need to be put in context with each other. For instance T , the best lower bound is 1,100,971, while the best solution is 1,103,819 leading to a gap of 0.258%. A similarly tight gap of 1.026% can be

obtained for instance R , where we determined a lower bound of 2,889,913 and a solution of 2,919,862. This shows the effectiveness of the presented approaches, since we are able to obtain feasible solutions and show their high quality within two hours of combined computation time for both real world instances. Finally, if we just consider the time necessary to determine ε -good solutions, we are able to determine a feasible solution and a lower bound with a gap of 0.856% for instance T within 100 seconds.

6 Conclusion

In this article we introduced RSRP-PM, a modified variant of the well known RSRP, which can be applied for integrated rolling stock planning with predictive maintenance scheduling. We presented an approximation to the defined problem by a state-expanded event-graph and gave an ILP formulation for solving the induced graph problem. From this formulation, we derived a primal heuristic as well as an approach to determine lower bounds for RSRP-PM relying on iteratively refining the approximating graph. Afterwards, we reported on the results obtained by applying these approaches to two test instances derived from real world timetables of a private German railway company.

These results, and the gaps resulting from combining the obtained solutions and lower bounds, show the effectiveness of our approaches for real world instances. We are able to find solutions and lower bounds, with a gap of 1% or less, within two hours of combined computation time for instances with a time horizon of one week. Although we considered a one-parameter probability distribution for modeling the failure probability of the vehicles, it should be mentioned that our approaches are also capable of handling time- or distance-based preventive maintenance instances. Therefore, the considered parameter needs to be exchanged with the traveled distance or the passed time. Finally, we want to highlight that the presented approaches do not rely on the linearity of the degradation functions yielding the possibility to incorporate more complex update functions as well as non-stable PDFs.

Possible next steps for future research, are a comparison of the given approaches to today's state-of-the-art approaches although they mainly tackle preventive maintenance scenarios with linear update functions. This is of particular interest, since Bougacha [3] states that authors in the field of predictive maintenance compare their work only to other maintenance regimes instead of comparing it to the approaches of other authors. Additionally, we aim to generalize our approaches to more than one dimension in order to apply more sophisticated probability distributions for modeling the failure probabilities. Finally, it would be meaningful to investigate the influence and the interaction of the cost coefficients, especially of the failure costs and maintenance costs.

References

1. Andrés, J., Cadarso, L., Marín, Á.: Maintenance scheduling in rolling stock circulations in rapid transit networks. *Transportation Research Procedia* **10**, 524–533 (2015). <https://doi.org/10.1016/j.trpro.2015.09.006>
2. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A fresh approach to numerical computing. *SIAM review* **59**(1), 65–98 (2017). <https://doi.org/10.1137/141000671>
3. Bougacha, O.: Contribution to post-pronostic decision: a new framework based on prongnostic/decision interaction. Ph.D. thesis, Université Bourgogne Franche-Comté (2020)
4. Bougacha, O., Varnier, C., Zerhouni, N.: Impact of decision horizon on post-prognostics maintenance and missions scheduling: a railways case study. *International Journal of Rail Transportation* **10**(4), 516–546 (2022). <https://doi.org/10.1080/23248378.2021.1940329>
5. Giacco, G.L.: Rolling stock rostering and maintenance scheduling optimization. Università degli studi Roma Tre (2014)
6. Grimm, B., Borndörfer, R., Reuther, M., Schlechte, T.: A cut separation approach for the rolling stock rotation problem with vehicle maintenance. In: 19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2019)
7. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual, Version 9.5.1. <http://www.gurobi.com> (2022)
8. Heng, A., Zhang, S., Tan, A.C., Mathew, J.: Rotating machinery prognostics: State of the art, challenges and opportunities. *Mechanical systems and signal processing* **23**(3), 724–739 (2009). <https://doi.org/10.1016/j.ymsp.2008.06.009>
9. Herr, N., Nicod, J., Varnier, C., Zerhouni, N., Cherif, M., Fnaiech, F.: Joint optimization of train assignment and predictive maintenance scheduling. In: International Conference on Railway Operations Modelling and Analysis (2017)
10. Herr, N., Nicod, J.M., Varnier, C., Zerhouni, N., Dersin, P.: Predictive maintenance of moving systems. In: 2017 Prognostics and System Health Management Conference (PHM-Harbin). pp. 1–6. IEEE (2017)
11. Reuther, M.: Mathematical optimization of rolling stock rotations. Technische Universität Berlin (Germany) (2017)
12. Rokhforoz, P., Fink, O.: Hierarchical multi-agent predictive maintenance scheduling for trains using price-based approach. *Computers & Industrial Engineering* **159**, 107475 (2021). <https://doi.org/10.1016/j.cie.2021.107475>
13. Wu, M.J., Lai, Y.C.: Train-set assignment optimization with predictive maintenance. In: RailNorrköping 2019. 8th International Conference on Railway Operations Modelling and Analysis (ICROMA), Norrköping, Sweden, June 17th–20th, 2019. pp. 1131–1139. No. 069, Linköping University Electronic Press (2019)

Appendix

Table 3. Results for the lower bounds on instance T using the lower bound approach with different instantiations for the step-size s and the decay parameter λ . The best lower bound is marked in bold and an asterisk $*$ indicates that the heuristic stopped due to exceeding the memory limits. Additionally, LB_{\max} is the best lower bound obtained among all instantiations.

s	λ	LB	Gap to LB_{\max} in %	Time in Secs.	Time to 99.5% LB_{\max} in Secs.
0.1	0.9	1,099,944	0.093	2,549	187
0.1	0.8	1,100,634	0.031	3,510	104
0.1	0.7	1,100,466	0.046	3,306	169
0.1	0.6	1,100,761	0.019	*3,134	129
0.1	0.5	1,100,633	0.031	*1,664	90
0.09	0.9	1,099,944	0.093	2,597	186
0.09	0.8	1,100,403	0.052	1,510	122
0.09	0.7	1,100,401	0.052	*1,690	110
0.09	0.6	1,100,296	0.061	1,135	160
0.09	0.5	1,100,568	0.037	2,125	120
0.08	0.9	1,100,308	0.060	3,600	226
0.08	0.8	1,100,413	0.051	2,244	109
0.08	0.7	1,100,351	0.056	2,875	126
0.08	0.6	1,100,640	0.030	1,802	84
0.08	0.5	1,100,285	0.062	1,987	146
0.07	0.9	1,100,090	0.080	3,193	243
0.07	0.8	1,100,450	0.047	3,069	134
0.07	0.7	1,100,466	0.046	3,375	169
0.07	0.6	1,100,699	0.025	2,867	106
0.07	0.5	1,100,971	0.000	3,600	195
0.06	0.9	1,100,035	0.085	*2,591	213
0.06	0.8	1,100,708	0.024	2,583	125
0.06	0.7	1,100,441	0.048	*2,292	69
0.06	0.6	1,100,761	0.019	3,154	135
0.06	0.5	1,100,033	0.085	*1,042	217
0.05	0.9	1,100,015	0.087	3,240	138
0.05	0.8	1,100,450	0.047	2,354	164
0.05	0.7	1,100,945	0.002	2,748	149
0.05	0.6	1,100,371	0.054	*1,487	182
0.05	0.5	1,100,633	0.031	*1,577	83

Table 4. Results for the lower bounds on instance R using the lower bound approach with different instantiations for the step-size s and the decay parameter λ . The best lower bound is marked in bold and an asterisk $*$ indicates that the heuristic stopped due to exceeding the memory limits. Additionally, LB_{\max} is the best lower bound obtained among all instantiations.

s	λ	LB	Gap to LB_{\max} in %	Time in Secs.	Time to 98% LB_{\max} in Secs.
0.1	0.9	2,861,667	0.977	1,521	909
0.1	0.8	2,872,270	0.611	2,133	384
0.1	0.7	2,889,913	0.000	3,392	737
0.1	0.6	2,875,657	0.493	*1,274	469
0.1	0.5	2,869,314	0.713	*850	850
0.09	0.9	2,861,667	0.977	1,544	918
0.09	0.8	2,871,319	0.643	2,834	339
0.09	0.7	2,875,879	0.486	2,092	512
0.09	0.6	2,863,917	0.900	*1,866	241
0.09	0.5	2,876,335	0.470	*1,104	250
0.08	0.9	2,868,637	0.736	*1,942	938
0.08	0.8	2,872,270	0.611	2,187	395
0.08	0.7	2,874,611	0.529	2,672	557
0.08	0.6	2,868,919	0.726	*2,363	280
0.08	0.5	2,880,863	0.313	*1,499	326
0.07	0.9	2,860,158	1.030	2,674	675
0.07	0.8	2,864,283	0.887	*1,181	498
0.07	0.7	2,889,913	0.000	*3,516	741
0.07	0.6	2,877,641	0.425	*2,883	341
0.07	0.5	2,869,972	0.690	*2,166	484
0.06	0.9	2,883,609	0.218	3,537	1,282
0.06	0.8	2,880,167	0.337	2,569	692
0.06	0.7	2,879,384	0.364	2,234	298
0.06	0.6	2,875,657	0.493	*1,306	482
0.06	0.5	2,879,012	0.377	3,139	585
0.05	0.9	2,863,404	0.917	1,661	1,120
0.05	0.8	2,863,906	0.900	2,499	944
0.05	0.7	2,868,584	0.738	1,546	366
0.05	0.6	2,875,376	0.503	*1,895	257
0.05	0.5	2,869,314	0.713	*827	827

Table 5. Results for the solutions to test instance T using the heuristic with different instantiations. The best result is marked in bold and runs stopped due to exceeding the memory limit are marked with an asterisk *. By ε -good solution we refer to a solution having a value less than $(1 + \varepsilon)$ times the best obtained solution, for $\varepsilon = 0.1\%$. Additionally, x_{best} is the best solution obtained among all instantiations.

s	λ	Best Sol.	Gap to x_{best} in %	Time in Secs.	# Sols	Time to ε -good Sol. in Secs.
0.1	0.9	1,103,819	0.0	216	11	71
0.1	0.8	1,103,819	0.0	100	6	45
0.1	0.7	1,103,819	0.0	246	7	63
0.1	0.6	1,103,819	0.0	845	8	75
0.1	0.5	1,103,819	0.0	*685	6	36
0.09	0.9	1,103,819	0.0	214	10	69
0.09	0.8	1,103,819	0.0	2,516	9	68
0.09	0.7	1,103,819	0.0	132	6	67
0.09	0.6	1,103,819	0.0	97	5	60
0.09	0.5	1,103,819	0.0	268	5	42
0.08	0.9	1,103,819	0.0	236	9	118
0.08	0.8	1,103,819	0.0	89	5	37
0.08	0.7	1,103,819	0.0	410	9	127
0.08	0.6	1,103,819	0.0	210	5	29
0.08	0.5	1,103,819	0.0	*153	5	87
0.07	0.9	1,103,819	0.0	398	14	149
0.07	0.8	1,103,819	0.0	304	8	54
0.07	0.7	1,103,819	0.0	427	8	68
0.07	0.6	1,103,819	0.0	576	6	33
0.07	0.5	1,103,819	0.0	444	6	55
0.06	0.9	1,103,819	0.0	1,199	11	100
0.06	0.8	1,103,819	0.0	118	8	81
0.06	0.7	1,103,819	0.0	76	5	39
0.06	0.6	1,103,819	0.0	3,600	8	70
0.06	0.5	1,103,819	0.0	*81	4	43
0.05	0.9	1,103,819	0.0	220	9	124
0.05	0.8	1,103,819	0.0	*139	6	46
0.05	0.7	1,103,819	0.0	260	8	60
0.05	0.6	1,103,819	0.0	*55	4	37
0.05	0.5	1,103,819	0.0	*238	4	38

Table 6. Results for the solutions to test instance R using the heuristic with different instantiations. The best result is marked in bold and runs stopped due to exceeding the memory limit are marked with an asterisk *. Additionally, x_{best} is the best solution obtained among all instantiations.

s	λ	Best Sol.	Gap to x_{best}	in %	Time in Secs.	# Sols
0.1	0.9	2,944,973		0.853	3,600	2
0.1	0.8	2,952,887		1.118	3,087	1
0.1	0.7	2,952,887		1.118	2,989	1
0.1	0.6	2,952,887		1.118	3,075	1
0.1	0.5	2,938,617		0.638	3,600	2
0.09	0.9	2,945,547		0.872	2,428	1
0.09	0.8	2,945,547		0.872	2,461	1
0.09	0.7	2,945,547		0.872	2,454	1
0.09	0.6	2,938,252		0.626	3,600	2
0.09	0.5	2,945,547		0.872	2,471	1
0.08	0.9	3,001,313		2.714	3,600	2
0.08	0.8	3,065,579		4.753	3,224	1
0.08	0.7	2,964,094		1.492	3,600	2
0.08	0.6	2,928,834		0.306	3,600	2
0.08	0.5	2,982,498		2.100	3,600	2
0.07	0.9	3,004,134		2.805	3,600	2
0.07	0.8	2,957,688		1.279	3,257	2
0.07	0.7	2,926,225		0.217	3,600	2
0.07	0.6	2,961,882		1.419	3,600	2
0.07	0.5	2,933,881		0.478	3,600	3
0.06	0.9	2,932,479		0.430	3,600	2
0.06	0.8	2,919,862	0.000		3,600	2
0.06	0.7	2,958,056		1.291	3,600	2
0.06	0.6	2,924,767		0.168	3,600	3
0.06	0.5	2,924,037		0.143	2,837	2
0.05	0.9	2,938,857		0.646	3,004	1
0.05	0.8	2,938,857		0.646	3,003	1
0.05	0.7	2,938,857		0.646	3,006	1
0.05	0.6	2,924,198		0.148	3,600	2
0.05	0.5	2,938,857		0.646	2,989	1