Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

PNA

Probability, Networks and Algorithms

Probability, Networks and Algorithms

Maintenance Routing for Train Units: the Scenario Model

G. Maróti, L.G. Kroon

REPORT PNA-E0414 AUGUST 2004

CWI is the National Research Institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organization for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-3711

Maintenance Routing for Train Units: the Scenario Model

ABSTRACT
Train units need regular preventive maintenance. Given the train units that require maintenance in the forthcoming 1--3 days, the rolling stock schedule must be adjusted so that these urgent units reach the maintenance facility in time. We present an integer programming model for this problem, give complexity results, suggest solution methods and report our computational results on practical instances of NS Reizigers, the main Dutch operator of passenger trains.

2000 Mathematics Subject Classification: 90B06; 90B10; 90C35
Keywords and Phrases: Railway transportation; maintenance routing

Maintenance Routing for Train Units: the Scenario Model

Gábor Maróti
CWI, Amsterdam and
NS Reizigers, Utrecht
P.O.Box 94079, 1090 GB Amsterdam, The Netherlands
E-mail: G.Maroti@cwi.nl

Leo Kroon
NS Reizigers, Utrecht and
Erasmus University, Rotterdam
P.O.Box 1738, 3000 DR Rotterdam, The Netherlands
E-mail: L.Kroon@fbk.eur.nl

Abstract

Train units need regular preventive maintenance. Given the train units that require maintenance in the forthcoming 1–3 days, the rolling stock schedule must be adjusted so that these urgent units reach the maintenance facility in time. We present an integer programming model for this problem, give complexity results, suggest solution methods and report our computational results on practical instances of *NS Reizigers*, the main Dutch operator of passenger trains.

AMS Subject classification: 90B06, 90B10, 90C35
Keywords: Railway transportation, maintenance routing.

1 Introduction

The Dutch passenger railway operator *NS Reizigers* runs a number of train units in order to carry out its timetable services. These units need regular preventive maintenance checks, say every 30,000 km: units that travelled that far since their last maintenance check must go to the maintenance facility. These checks are arranged by maintenance routing planners. They modify the rolling stock

schedule so that the units that need maintenance in the next couple of days arrive in time at a maintenance facility.

As yet, NS Reizigers does not use a decision support system for this maintenance routing. The goal of this research is to provide mathematical programming models and solution methods that can serve as basis for such a decision support system. In this paper, we present an integer programming model for maintenance routing. Implementations for test data give promising results: we find good solutions fast.

The description of the maintenance routing problem is given in Section 2 and our integer programming model is presented in Section 3. In Section 4 we prove some complexity results and propose a heuristic solution method. Section 5 contains computational results.

2 The maintenance routing problem

2.1 The planning in practice

NS Reizigers uses train units; each train consists of one or more train units. The *regular plan* contains all rolling stock movements between different stations but does not involve movements inside stations. These are described by shunting plans created by local shunting crew.

A *task* is a smallest indivisible movement in the regular plan to be carried out by a single train unit. So, if a movement is carried out by a train with more than one unit, it corresponds to several tasks. A *regular duty* is the sequence of tasks planned to be carried out by the same train unit. Two consecutive tasks in a regular duty form a *regular transition*.

Some of the tasks in the regular plan are *maintenance tasks*. If a train unit carries out such a task, it undergoes a maintenance check. (Maintenance checks can only take place at specialised stations.) The regular plan assigns maintenance tasks arbitrarily, irrespective of which unit needs to be maintained and when. The actual maintenance routing is handled shortly before execution. NS Reizigers uses a fixed planning horizon for maintenance routing, usually 3 days. Input is the regular plan and a list of *urgent* train units: those that need a maintenance check within the planning horizon. They may have different urgencies, expressed by a deadline (for instance saying that the unit should be maintained within 2 days). The part of the regular plan that lies within the planning horizon must be adjusted so that each urgent unit is routed to a main-

tenance facility while still being used for timetable services. This maintenance planning process is carried out daily, with a rolling horizon.

The maintenance planners work as follows. They consider a (small) number of regular transitions (t_i, t_i') (that is, pairs of consecutive tasks) such that the arrival station of the tasks t_i is the same, and replace these transitions simultaneously by a collection of new transitions between the t_i's and the t_i''s. We call such a simultaneous interchange a *changing scenario*. The maintenance planners look for changing scenarios that lead each urgent unit to a maintenance task within its deadline. Figure 1 shows a small example for a single urgent train unit. The thick lines represent the tasks, the dashed lines indicate the regular transitions. The arrows show the new transitions that route the urgent train unit to the maintenance task.

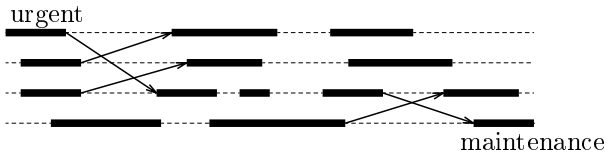


Figure 1

The changing scenarios may require adjusted shunting plans at the stations. Generating shunting plans is a difficult problem in itself, even for just a single middle-sized station. So, the maintenance routing planners themselves cannot decide the practical feasibility of a modified plan. Therefore, they check with the local shunting crew if the preferred changing scenarios can be carried out.

If the maintenance planners do not find a solution, they may decide that one or more urgent train unit goes from one station to another as an extra empty train. This is quite expensive and may conflict with the schedules of the train drivers or with the capacity of the infrastructure. Therefore, this is to be avoided as much as possible.

Communication between the maintenance planners and the local shunting crew takes time, so the maintenance routing planners cannot test too many possibilities. Therefore they are usually satisfied with the first feasible solution found.

2.2 Previous work

Several related papers can be found in the literature. Lingaya et al. [7] describe a model for operational management of locomotive-hauled railway cars. They seek for a maximum expected profit schedule that satisfies various constraints, among them also maintenance requirements.

Some other papers focus on aircraft maintenance routing. For example, Barnhart et al. [2], Clarke et al. [3], Feo and Bard [5], Gopalan and Talluri [6] and Talluri [8] deal with this problem. Anderegg et al. [1] describe models for railway applications that are similar to the aircraft routing models.

These models cannot be applied directly to maintenance routing at NS Reizigers. The models consider maintenance routing as a part of the medium- and long-term vehicle scheduling problem, and specify the duties of the individual vehicles during the entire planning period. The models above create the vehicle schedule from scratch, without taking shunting issues into account. Therefore in our railway application there is no guarantee that the output of these models can be carried out in practice.

Furthermore, disturbances and delays make it unlikely that the rolling stock schedule can be carried out exactly as planned in a period longer than a couple of days. On the other hand, the Dutch railway network contains a large number of frequently operated relatively short train lines. This provides many exchanging possibilities for the train units, therefore a couple of days is mostly enough to route an urgent train unit to a maintenance facility. This explains why the maintenance planners consider planning periods of length 1–3 days, and only a small number of train units to be routed.

2.3 Our approach

In a decision support system for maintenance routing, we should be able to determine the difficulty of the solutions it creates. We cannot instantly check whether the modifications in the regular plan can be implemented in practice. Yet, we can rely on the practical feasibility of the regular plan. If we apply only a small number of changes, the modified plan will be close to the regular (feasible) plan and we may hope that the local shunting crew will be able to carry out the new plan.

The set of all possible changing scenarios can be very large. In our model we have a (relatively small) set of "elementary changing scenarios" and use these as building blocks for the entire solution. We assume that each elementary chang-

ing scenario is feasible in itself and that we know a cost value that estimates the difficulty to carry it out. These cost values are used to predict the feasibility of a certain combination of elementary changing scenarios. As measure of how hard it is to carry out a combination, we basically use the sum of the costs of the scenarios in it.

Mind that whatever method for maintenance planning is used, in practice each solution needs approval of the local shunting crew before it can be implemented. Therefore the final goal of this research is to develop an *interactive* decision support system that proposes changing scenarios to the maintenance planner. The planner can accept the proposal or run the system again with modified specifications. Results on single instances can also be used to update the allowed changing scenarios or the associated costs.

2.4 Formalising maintenance routing

In this section, we formalise the data that describe the maintenance routing problem.

Timetable data

We denote the set of tasks during the planning period by T. For each task $t \in T$, the arrival time, arrival station, departure time and departure station are denoted by $\mathsf{Arr\_time}(t)$, $\mathsf{Arr\_stat}(t)$, $\mathsf{Dep\_time}(t)$ and $\mathsf{Dep\_stat}(t)$. The regular plan is described by the functions $\mathsf{next}()$ and $\mathsf{prev}()$. For a task $t \in T$, $\mathsf{next}(t)$ is a task in T such that $(t, \mathsf{next}(t))$ is a regular transition. Similarly, $\mathsf{prev}(t)$ is a task in T such that $(\mathsf{prev}(t), t)$ is a regular transition. Let $\mathsf{duty}(t)$ denote the regular duty containing task t. The set of urgent train units is denoted by U. For any $u \in U$, let $M(u)$ be the set of those maintenance tasks that can be assigned to u.

Transitions, changing scenarios

Now we formalise the options the planner can choose from. Some regular transitions have to be replaced by other transitions. In order to reserve time for the necessary shunting operations and to protect against spreading of delays, we require some buffer time: A pair (t, t') of tasks is a *transition* if it either is

a regular transition, or if

$$\mathsf{Arr\_stat}(t) = \mathsf{Dep\_stat}(t') \text{ and}$$
$$\mathsf{Arr\_time}(t) \leqslant \mathsf{Dep\_time}(t') - \mathsf{Buffer\_time}.$$

The buffer time might depend on t and t'. For simplicity, we set it uniformly to 10 minutes, based on discussions with the planners. If we wish to allow some empty train movements as well, we can do this by introducing some more general transitions between tasks at different stations; the buffer time for such a transition then also has to account for the travel time between the two stations.

A *changing scenario* is a set $S = \{(t_i, t_i') : i \in I\}$ of transitions with distinct tasks t_i such that $\{t_i' : i \in I\} = \{\mathsf{next}(t_i) : i \in I\}$. It is meant as a candidate to simultaneously replace the collection of regular transitions $\{(t_i, \mathsf{next}(t_i)) : i \in I\}$. We may well assume that none of the transitions (t_i, t_i') in a changing scenario is regular and that all tasks t_i in it arrive at the same station (unless the scenario involves empty train rides). The simplest kind of a changing scenario has the form $\{(t_1, \mathsf{next}(t_2)), (t_2, \mathsf{next}(t_1))\}$, it interchanges two train units.

Elementary scenarios

The notion of changing scenario is still a bit too unstructured to be of much use; for instance if you consider a potential solution, then the collection of its new transitions at a single station over the entire planning period is a changing scenario. Therefore we will restrict ourselves to elementary changing scenarios and use these as building blocks for our solutions.

For changing scenario $S = \{(t_i, t_i') : i \in I\}$, we denote $\max_i \mathsf{Arr\_time}(t_i)$ by $\alpha(S)$ and $\min_i \mathsf{Dep\_time}(t_i')$ by $\omega(S)$. We say that S is an *elementary scenario* if $\omega(S) - \alpha(S) \geqslant \mathsf{Buffer\_time}$. That is, if the train units spend at least $\mathsf{Buffer\_time}$ minutes together at the given station.

A large part of the changing scenarios that turned out to be important in practice is in fact elementary. We note that the notion of elementariness can be extended in order to cover some other changing scenarios. Then the graph representation in Section 3 and the correctness proof of the heuristic approach in Section 4.2 must be adjusted accordingly.

By confining ourselves to elementary changing scenarios we limit our scope: not every changing scenario is decomposable into elementary ones. Still in our study we will only consider solutions that are composed from a given list of elementary changing scenarios which are assumed to be feasible individually.

Two elementary changing scenarios can be chosen simultaneously in a solution when they can be carried out at such times that they do not interfere with each other.

Shunting difficulty

The shunting difficulty of a changing scenario S is measured by a non-negative value associated with S. In real-life applications, the list of changing scenarios as well as the cost evaluation should be created or at least approved by the local shunting planners. The following factors play an important role when defining the cost values:

1. The number of extra shunting movements, the physical distance between the tracks, whether or not extra storage tracks are required for carrying out the changing scenario; these parameters depend on the positions of the units in the train compositions as well as on the infrastructure of the stations.
2. The length of the time interval between the arrival and departure events: the shorter this interval, the higher the cost.
3. Stations with heavy traffic provide less changing possibilities.
4. It is very hard to apply changes during morning and evening rush hours; changes at night and during off-peak hours are easier.

3 The Scenario Model

The input to our model contains the set T of tasks together with the functions $\mathsf{Arr\_time}()$, $\mathsf{Arr\_stat}()$, $\mathsf{Dep\_time}()$, $\mathsf{Dep\_stat}()$, $\mathsf{next}()$, $\mathsf{prev}()$ and $\mathsf{duty}()$, moreover, a list $\mathcal{S}$ of elementary changing scenarios with the cost values estimating the shunting difficulty of each changing scenario $S \in \mathcal{S}$.

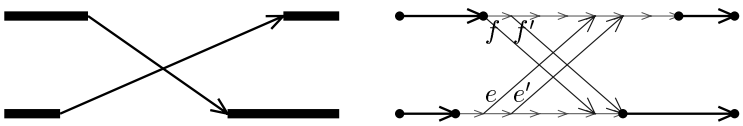
The graph representation

We give a flow-type model for the maintenance routing problem, thus we first need a graph. We create a node for every pair of a duty d and a minute m in the planning such that m does not lie strictly inside any of the intervals $[\mathsf{Dep\_time}(t), \mathsf{Arr\_time}(t)]$ where t is a task in duty d. For a node $v = (d, m)$, let $\mathsf{time}(v) = m$. We identify each maintenance task t with the node

(duty(t), Dep_time(t)). We call the nodes corresponding to the begin and the end of the planning period *first* and *last* nodes, respectively.

For any task v, we insert a *task arc* between (duty(v), Dep_time(v)) and (duty(v), Arr_time(v)). Moreover, for each duty d and for each minute m in the planning period, we join the node (d, m) to $(d, m+1)$ if these nodes exist and if they are not joined by a task arc. We call them *regular arcs*. So far, we have a collection of disjoint directed paths consisting of task arcs and regular arcs.

Carrying out a changing scenario takes Buffer_time minutes. So, an elementary scenario $S = \{(t_i, t_i') : i \in I\}$ can be carried out at several time moments. Therefore we implement the scenario S by several *changes*, one for each integer m in $[\alpha(S), \omega(S) - \text{Buffer\_time}]$. A change is a set of arcs containing an arc from the node (duty(t_i), m) to the node (duty(t_i'), $m + \text{Buffer\_time}$), for every $i \in I$. A change is thus a changing scenario *plus* an exact time specification when the scenario should be carried out. An example is given in Figure 2. The left hand side of the figure shows a schematic representation of the tasks and the transitions in the scenario. The corresponding part of the graph is shown on the right hand side of the figure. The scenario is represented by the changes $\{e, f\}$ and $\{e', f'\}$.



Elementary scenario Two changes

Figure 2

The arcs occurring in the changes are called *change arcs*. Note that when a transition belongs to more than one scenario, it corresponds to more than one change arc, so the graph may have parallel arcs. We denote the graph by $G_{\text{scen}} = (V, A)$ and the set of changes by $\mathcal{C}$.

The cost of a change is the cost of the corresponding changing scenario. Define the function $c\colon A \to \mathbb{R}_+$ as follows. Let $c(a) = 0$ for regular arcs and task arcs. For a change arc a that belongs to the change C, let $c(a)$ be the cost of the changing scenario from which C was derived. That is, the cost of each change arc fully represents the difficulty of the corresponding scenario.

Executing a change $C = \{(v_i, w_i) : i \in I\}$ means replacing the regular arc paths that connect the nodes v_i to the nodes w_i by the change arcs (v_i, w_i).

We pointed out in Section 2.4 that elementary changing scenarios can only be combined in a solution if their shunting operations do not interfere. We can take care of that in our model as follows. Suppose that a change C contains an arc leaving the node (d, m) and an arc entering the node (d, m'). When executing C, no train unit is present in duty d during the time interval $I_C(d) = [m, m']$. So we cannot combine the change C with another change C' at the same station if $I_C(d)$ and $I_{C'}(d)$ meet. Other pairs of changes can be combined. Therefore we call changes C and C' *independent* if they take place at different stations, or if $I_C(d)$ and $I_{C'}(d)$ are disjoint for each duty d.

According to the regular plan, the train units follow the regular arcs. Executing some pairwise independent changes yields a system of node-disjoint paths, each path indicating the route of a train unit.

Now we formulate the

Scenario Model: *Select a collection of pairwise independent changes such that, when executing these changes, the urgent units are routed to a node representing an appropriate maintenance task. Minimise the total cost of the solution, defined as the sum of the individual costs of those arcs that are used by the urgent train units until they reach a maintenance task.*

By inserting the changing scenarios multiple times, the Scenario Model has the possibility to carry out several changes for a train unit between two consecutive tasks. That is, the model can use all those more complex changing scenarios in the solutions that are built up from the elementary changes in $\mathcal{S}$. A small example is given in Figure 3. The left hand side shows a non-elementary changing scenario, that can yet be obtained by using two changes after each other.

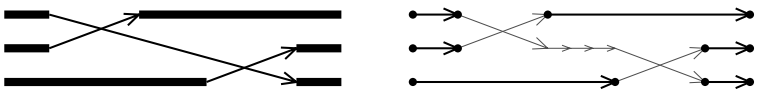


Figure 3: A non-elementary scenario

The integer programming formulation

We formulate the model as a binary linear program.

The variables are:

$$y\colon \mathcal{C} \to \{0,1\}, z\colon A \to \{0,1\}, \text{ and, for each } u \in U,\ x_u\colon A \to \{0,1\}. \quad (1)$$

The variables y_C are binary decision variables for the changes: $y_C = 1$ if and only if change $C \in \mathcal{C}$ is selected in the solution. The functions x_u are network flows, indicating the paths of the urgent units to the maintenance tasks, while the flow z is the collection of the paths of all train units.

The constraints are:

1. flow conservation for z at every node v except for the first and last nodes; moreover, the nodes have capacity 1 in order to obtain node-disjoint paths (δ^{in} and δ^{out} denote the set of entering and leaving arcs, resp.):

$$z\left(\delta^{\text{in}}(v)\right) = z\left(\delta^{\text{out}}(v)\right) \leqslant 1; \quad (2)$$

2. flow conservation for each flow x_u at every node v except for the first node in the duty of u and for the end nodes of the tasks in $M(u)$:

$$x_u\left(\delta^{\text{in}}(v)\right) = x_u\left(\delta^{\text{out}}(v)\right); \quad (3)$$

3. the flows x_u must form a subsystem of the paths defined by z:

$$\text{for every arc } a\colon \sum_{u \in U} x_u(a) \leqslant z(a); \quad (4)$$

4. for every $u \in U$, the first node v in the duty of u has out-flow 1 in x_u; every first node v' has out-flow 1 in z:

$$x_u\left(\delta^{\text{out}}(v)\right) = 1; \qquad z\left(\delta^{\text{out}}(v')\right) = 1; \quad (5)$$

5. a maintenance node v has neither any out-going flow in any of the urgent flows, nor any in-going flow in a flow x_u with $v \notin M(u)$:

$$\sum_{u \in U} x_u(\delta^{\text{out}}(v)) = 0; \qquad \sum_{u \in U : v \notin M(u)} x_u\left(\delta^{\text{in}}(v)\right) = 0; \quad (6)$$

6. for any change C and any arc $a \in C$, the arc a is used by z if and only if C has been selected:

$$z(a) = y_C. \tag{7}$$

The objective function is:

$$\text{minimise} \sum_{a \in A} c(a) \cdot \sum_{u \in U} x_u(a). \tag{8}$$

Note that the objective function is *not* the sum of the costs of the selected changes. If more than one urgent train unit use a change, the cost of the change is counted multiple times in (8). The strategy that several urgent train units should not appear in the same train composition may increase the robustness of the solution against disturbances.

Several other objective functions could be supported by practical arguments. We shall see in Section 5 that our choice enables us to compute good lower bounds for measuring the performance of heuristic solution methods.

To see that the binary linear program is in fact a formulation of the Scenario Model, we prove the following proposition.

Proposition 1. *Consider any feasible solution to the Scenario Model, let C be a change with $y_C = 1$. Let $\{v_i : i \in I\}$ and $\{w_i : i \in I\}$ denote the sets of tails and heads of the arcs in C, respectively, so that v_i and w_i belong to the same duty. Then there does not exist any node v such that v lies on the interior of the regular arc path between v_i and w_i for some i, and such that v is traversed by the flow z.*

Proof. Suppose there exists such a change C and a node v lying between v_i and w_i (see Figure 4). Choose them so that $\mathsf{time}(v)$ is as small as possible. Then the nodes strictly between v_i and v have no throughput in z. Let e be the arc entering v with $z(e) = 1$. The arc e cannot be a regular arc or a task arc, so $e \in C'$ for some change C'. This change must contain an arc f with $z(f) = 1$ that leaves a node on the duty of v earlier than $\mathsf{time}(v)$. The tail of f is traversed by z, therefore $\mathsf{time}(\mathsf{tail}(f)) < \mathsf{time}(v_i)$. Then C' and v_i form a counterexample, thereby contradicting the choice of C and v. □

Therefore in each feasible solution to the binary linear program, the set of arcs with $z(a) = 1$ arises from the regular transitions by executing the changes with $y_C = 1$, and these changes are pairwise independent. That is, every feasible solution to the binary linear program has a corresponding feasible solution to the Scenario Model with the same objective value. One can easily prove the other direction: every solution to the Scenario Model corresponds to a solution to the binary linear program.

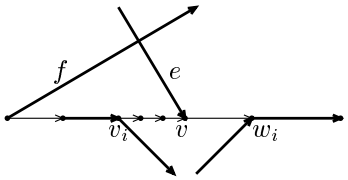


Figure 4

4 Complexity results for the Scenario Model

4.1 NP-completeness of the model

First we show that the feasibility problem of the Scenario Model is NP-complete. The size of the graph depends on the number of tasks and on the size of the list $\mathcal{C}$ of changes. The construction shows that the problem is NP-complete if $|\mathcal{C}| = O(d)$ where d denotes the number of duties (i.e. train units) and the longest path in our acyclic graph has length 3.

Theorem 2. *It is NP-complete to decide whether or not the Scenario Model has a feasible solution.*

Proof. We show that 3SAT can be polynomially reduced to our problem.

Consider a conjunctive normal form

$$\varphi = \bigwedge_{i=1}^{k} (x_{a_i}^{\varepsilon_{a_i}} \vee x_{b_i}^{\varepsilon_{b_i}} \vee x_{c_i}^{\varepsilon_{c_i}})$$

in the Boolean variables $x_1, \ldots, x_\ell$ where $\varepsilon_j \in \{\pm 1\}$ and

$$x_j^{\varepsilon_j} = \begin{cases} x_j & \text{if } \varepsilon_j = +1, \\ \neg x_j & \text{if } \varepsilon_j = -1. \end{cases}$$

It is well known that deciding whether or not φ can be satisfied is NP-complete (Cook [4]).

For any conjunctive normal form φ, we build up an instance of the Scenario Model such that the required paths in the graph exist if and only if φ is satisfiable. The construction is polynomial in the size of φ.

Let $d = 6k$. We start from an empty graph and an empty list $\mathcal{C}$. For each clause $x_{a_i}^{\varepsilon_{a_i}} \vee x_{b_i}^{\varepsilon_{b_i}} \vee x_{c_i}^{\varepsilon_{c_i}}$, we create the nodes s_i, s_i', s_i'' and t_i, t_i', t_i''.

For every occurrence of a literal x_j or $\neg x_j$ (say, in the clause i), we create a *box* consisting of 6 new nodes u, v, w, z, a, b and the arcs uv, wz, uz, aw. We also draw the arcs s_iu, s'_iu, s''_iu. If the non-negated literal x_j appears in clause i, we also draw the arcs vt_i, vt'_i, vt''_i, zb. If $\neg x_j$ appears in clause i, we insert the arcs zt_i, zt'_i, zt''_i, vb (see Figure 5). We declare aw as well as vb or zb "horizontal".

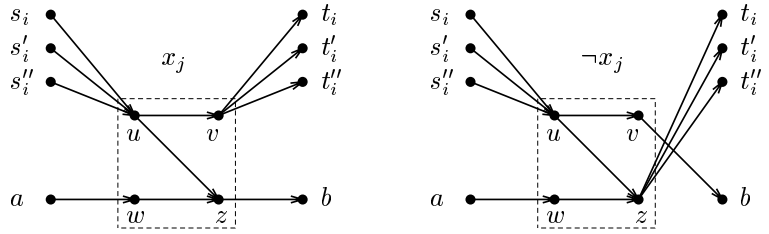


Figure 5

We join the boxes corresponding to the same Boolean variable by fixing any cyclic ordering on them and drawing an arc from the w-node of any box to the v-node of the next box in the ordering. Then we obtain a graph like in Figure 6. In this figure, the dashed rectangles contain the u, v, w, and z-nodes of the boxes, while the a and b-nodes are omitted.

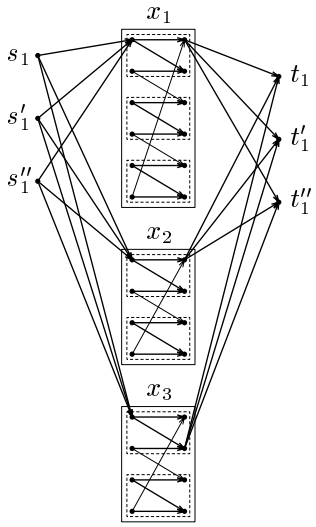


Figure 6 : $\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge \ldots$

For each variable x_j, we create a change $C \in \mathcal{C}$ consisting of the arcs of type uz and wv in the x_j-boxes. We also declare uv and wz "horizontal".

For every clause i, consider the nodes s_i, s_i', s_i'' and the 3 u-nodes they are connected to. These 6 nodes span a complete bipartite subgraph. Decompose these 9 arcs into 3 disjoint perfect matchings, declare the arcs in one of the matchings to be "horizontal", and add the other two matchings as arc sets of cardinality 3 to $\mathcal{C}$. Then do the same at the other side: at t_i, t_i', t_i'' and their neighbours.

This gives a possible instance of the Scenario Model. Everything happens at the same station, each node represents a task with a length of zero minutes. The "horizontal" arcs are the regular transitions, the nodes s_i are the first tasks for the urgent train units, t_i is the (unique) maintenance task assigned to s_i, and $\mathcal{C}$ is the list of changes. Note that the longest path in this graph contains 3 arcs, and that $\mathcal{C}$ has $\ell + 4k$ members.

We claim that the Scenario Model for this graph has a feasible solution if and only if φ can be satisfied.

Suppose there exists an assignment of the Boolean variables making φ true. If $x_j = 1$, select the horizontal arcs inside all the x_j-boxes, otherwise select the uz and wv arcs inside or between the x_j-boxes. For every clause i, choose a literal making it true, and consider the box corresponding to this occurrence of the variable. If the arc e from s_i to the u-node of this box is not "horizontal", select e as well as the two other arcs that appear in the same member of $\mathcal{C}$ as e does. If e is horizontal, select the horizontal arcs incident to s_i, s_i', s_i''. Select also the arc leading from the box to t_i together with the other two arcs defined similarly.

Finally, select for every box the arc leaving the a-node of the box, and the arc entering the b-node of the box. Then the selected arcs form a required path system: we only use horizontal (i.e. regular) arcs and all arcs of some changes.

Conversely, suppose there exists such a path system. Assign the value "true" to a variable if and only if the horizontal arcs are used in the x_j-boxes. Then for any i, the $s_i - t_i$ path shows that the ith clause is satisfied. □

4.2 The case of one urgent train unit

Each feasible solution to the Scenario Model contains a path from the first node of each urgent unit to a maintenance node. We show that such a path can always be extended to a solution to the Scenario Model. That is, the case of

one urgent train unit is reduced to a shortest path problem.

Theorem 3. *An optimal solution to the Scenario Model with one urgent train unit can be found in $O(|A|)$ time.*

Proof. Let P be a shortest path from the first node of the urgent unit to a maintenance node w.r.t. the cost function c. Then $c(P)$ is a lower bound for the objective value of any solution to the Scenario Model. Let C_1, C_2 be changes containing the change arcs $a_1, a_2 \in P$, respectively. Then C_1 and C_2 correspond to disjoint time intervals. Therefore C_1 and C_2 are independent. Applying all the changes that contain a change arc in P, we obtain a feasible solution to the Scenario Model, and its cost is $c(P)$. Since G_{scen} does not contain directed cycles (the arcs are directed according to the time), the shortest path algorithm can be implemented very efficiently, the running time being proportional to the number of arcs. □

4.3 A heuristic algorithm

We have seen in Section 4.2 that the case of one urgent train unit is easy to solve. The following *Iterated Shortest Path Heuristic* (ISPH) is a natural approach to find solutions for more than one urgent unit.

We fix an order of the set U of urgent units. We iterate on U, applying the following steps for the actual $u \in U$.

1. *We look for a shortest path P from the first node in the duty of u to the maintenance nodes that are allowed for u. We stop with an error if there is no path from u to $M(u)$.*

2. *We delete the arcs that are incident to the nodes in P except for the arcs of P. For each change arc a in P, let $C = \{(v_i, w_i) : i \in I\}$ be the change that contains a. Consider the regular arc paths that join the nodes v_i and w_i. We delete all their internal nodes together with the arcs incident to these internal nodes. Moreover, we delete the arcs leaving the nodes v_i or entering the nodes w_i, except for the arcs (v_i, w_i). In any case when a change arc a' is to be deleted, we also remove those arcs that occur in the same change with a'. Having updated the graph, we continue iterating on U.*

Theorem 3 implies that, in each iteration of the algorithm ISPH, the path for the actually considered urgent unit can be obtained by applying independent changes. Afterwards we explicitly forbid all those changes that are in conflict with the already selected changes. Therefore the algorithm ISPH provides a feasible solution to the Scenario Model if it terminates without an error.

It is not specified yet in which order the algorithm ISPH should process the urgent units. We can simply run the process we described so far on several different orders, and select the best solution.

We can compare the cost values of the heuristic solutions to the following natural lower bound. Let S_i be the set of first nodes of those urgent units that have maintenance on day i, let T_i be the set of maintenance nodes on day i. Let $\mathsf{MF}(i)$ denote the minimum cost of a network flow of value $|S_i|$ in G_{scen} with sources S_i, sinks T_i and with node capacities 1. Then

$$\mathsf{FlowBound} = \sum_{i=1}^{k} \mathsf{MF}(i)$$

is a lower bound on the cost of any feasible solution.

5 Computational results

5.1 The test case

We implemented the model for the train unit type 'Sprinter'. There are 47 train units serving about 800 tasks per day. The typical length of the train compositions is 1 or 2 units, there are only a few exceptional trains with 3 units, this makes the cost estimates for the changes easier. The regular plan contains two maintenance tasks per day, all taking place at Leidschendam station starting in the early morning.

For our test, we collected and evaluated the changing scenarios by applying some simple rules to the regular plan. We computed about 800 elementary changing scenarios per day. The cost values of the scenarios have been chosen between 0 and 1000. An arc with cost value higher than 100 corresponds to a changing scenario which is considered to be quite difficult in practice.

We also allowed empty train movements with cost value of 1000 since without using such movements, some test instances turned out to be infeasible.

5.2 Experiments

We generated 1000 instances each containing randomly chosen urgent units. We assumed that the urgent units had already been assigned to the maintenance tasks. We set the planning horizon to $k = 2$, 3, 4, and 5 days. We always start on Monday morning, so depending on k, the planning period lasts till the late evening of Tuesday, Wednesday, Thursday, or Friday. We have $2k$ urgent train units: 2 units assigned to the maintenance tasks starting early morning on Tuesday, 2 starting early morning on Wednesday, etc. The cases $k = 2$ and 3 correspond to the most frequently used planning periods in the maintenance planning process at NS Reizigers.

We applied two solution methods: (*i*) running the heuristic algorithm in Section 4.3, implemented in C++, and (*ii*) solving the binary linear program by the modelling software ILOG OPL Studio 3.7 and the integer programming solver ILOG Cplex 9.0. The computation has been carried out on a PC with an Intel P4 3.0 GHz processor and 512 MB internal memory. As mentioned above, we can choose any order in the ISPH algorithm for processing the urgent train units. In our implementation we ran ISPH for each instance with 24–52 orders, depending on the planning horizon k, and we selected the best solutions. We refer to the solution methods and their solution values as 'Opt' and 'ISPH'.

Since the graph G_{scen} we defined in Section 3 is quite large, the binary linear program could not be solved at all. We obtained a much smaller representation as follows. An elementary changing scenario has been represented as changes starting at every minute between $\alpha(S)$ and $\omega(S) - \mathsf{Buffer\_time}$. However, if this interval is very long, like $1 - 2$ hours, we insert only changes that start at minutes divisible by 10. The size of this new graph G^{10}_{scen} is dramatically smaller, as Table 1 shows. Nevertheless, when solving the same instances on both graphs by the heuristic method, the solutions turned out almost identical. We found in less than 1% of the test instances any difference between the objective function values in G_{scen} and G^{10}_{scen}. The largest difference was 6, it occurred once among the 1000 test instances, and this difference is less than 5% of the corresponding objective function value. So we do not loose substantial information by working with G^{10}_{scen}. Therefore in what follows, we only present the results for this smaller graph.

graph	# nodes	# arcs
G_{scen}	111,892	692,706
G^{10}_{scen}	21,358	88,351

Table 1

k	# inst.	avg. gap	max. gap	avg. abs. gap	max. abs. gap
2	1000	1.9%	71%	1.1	39
3	1000	5.6%	89%	4.9	54
4	1000	8.2%	85%	8.9	70
5	1000	14.3%	82%	19.1	103

Table 2 : ISPH vs. FlowBound

5.3 Performance of the algorithms

The ISPH gave a feasible solution for every test instance and for every planning horizon $k = 2$, 3, 4, and 5.

First we compare the flow bounds to the heuristic solutions. The absolute gap is defined by $\mathsf{ISPH} - \mathsf{FlowBound}$, while the relative gap is defined by $\frac{\mathsf{ISPH}-\mathsf{FlowBound}}{\mathsf{FlowBound}}$. The average and maximum value of the relative gaps and of the absolute gaps are presented in Table 2. Observe that the largest absolute gap is never higher than the cost of just one considerably expensive arc, although we computed paths for up to 10 urgent train units.

The test instances could also be solved by Cplex, despite the large size of the binary linear programs (see Table 3). The solution process required very few nodes in the branch and bound tree, quite often the linear programming relaxation turned out to have an integral optimal solution. Some instances required, however, a longer solution process taking up to 2 hours for proving optimality. Even in those cases, an almost optimal solution was found after a couple of branchings.

For comparing the heuristic solutions to the optimal solutions, we used the absolute gap $\mathsf{ISPH} - \mathsf{Opt}$ and the relative gaps $\frac{\mathsf{ISPH}-\mathsf{Opt}}{\mathsf{Opt}}$. Table 4 shows the

	$k = 2$	$k = 3$	$k = 4$	$k = 5$
Number of variables	265,053	353,404	441,755	530,106
Number of constraints	74,961	125,850	183,213	250,976

Table 3 : Size of the binary linear programs

k	# inst.	avg. gap	max. gap	avg. abs. gap	max. abs. gap
2	1000	0.83%	42.19%	0.51	27
3	1000	2.53%	56.60%	2.37	37
4	1000	3.79%	53.42%	4.39	44
5	1000	6.04%	48.05%	8.70	54

Table 4: ISPH vs. Opt

k	# inst.	avg. gap	max. gap	avg. abs. gap	max. abs. gap
2	1000	0.82%	30.77%	0.61	39
3	1000	2.43%	40.00%	2.57	39
4	1000	3.52%	36.36%	4.53	44
5	1000	6.66%	36.09%	10.42	50

Table 5 : FlowBound vs. Opt

average and maximum values of these performance indicators. The quality of the flow bounds is examined in Table 5, the measures being the absolute gap $\mathsf{Opt} - \mathsf{FlowBound}$ and the relative gap $\frac{\mathsf{Opt}-\mathsf{FlowBound}}{\mathsf{Opt}}$.

Table 6 shows the average running times for computing the flow bounds, for the heuristic algorithm, and for solving the binary linear programs by Cplex. We can see that the heuristic method is in fact very fast. The solution times of Cplex were also acceptable for the practically important planning horizons $k = 2$ and 3 (though a bit longer than convenient in an interactive decision support application). We expect that for other train unit types, which are much larger than our test case, solving the binary linear program becomes very difficult while the heuristic method still performs well.

	$k = 2$	$k = 3$	$k = 4$	$k = 5$
FlowBound	–	–	–	4.91 s
ISPH: 1 order	0.08 s	0.13 s	0.21 s	0.27 s
# orders	24	28	36	52
ISPH: all orders	2.23 s	3.97 s	7.43 s	15.03 s
Binary program	20.41 s	87.36 s	296.44 s	1062.99 s

Table 6 : Average running times (in seconds)

5.4 Conclusions

The good performance of the heuristic solution method can be explained by the structure of the problem. The number of urgent train units is small compared to the total number of train units (especially in the case of a shorter planning horizon, such as $k = 2$ or 3), so the urgent train units do not have too much interaction with each other. On the other hand, the rolling stock schedule provides a large enough number of changing possibilities, yielding several almost optimal paths for an urgent unit.

The most critical problem is to route the two most urgent train units to the maintenance station till Tuesday morning when the planning is carried out on Monday morning. In the test instances this was the only reason to use very expensive arcs. The lower bound proved in any of these cases that there is no solution without empty train movements.

The computational results indicate that when solving a sequence of problems every day, and using each day's output for the next day, then the system would very rarely use expensive arcs. However, disturbances and delays may prevent the local shunting crew to carry out the desired changes. This may lead in practice to using more expensive changes, sometimes even empty train movements.

6 Summary

In this paper we described the Scenario Model for supporting the routing of passenger train units towards a maintenance facility. We analysed the computational complexity of this maintenance routing problem, and we suggested a heuristic solution method. In our computational tests optimal or nearly optimal solutions could be found in short running times.

The model takes a lot of details about the shunting process into account. This increases the chance that the solutions are acceptable in practice. Reliable technical information is, however, difficult to collect. Moreover, it is time-consuming to maintain the data if the timetable and the rolling stock schedule changes. Therefore, in our future research we focus on developing alternative models dealing with less details of the technical possibilities, but yet providing solutions that help the work of the maintenance routing planners.

Acknowledgement

This research was supported by the Human Potential Programme of the European Union under contract no. HPRN-CT-1999-00104 (AMORE).

We would like to thank Bert Gerards for many valuable discussions and comments concerning the present paper.

References

[1] L. Anderegg, S. Eidenbenz, M. Gantenbein, Ch. Stamm, D.S. Taylor, B. Weber, P. Widmayer, Train routing algorithms: Concepts, design choices, and practical considerations, *Proceedings of ALENEX'03*, `www.siam.org/meetings/alenex03/Abstracts/LAnderegg3.pdf`.

[2] C. Barnhart, N.L. Boland, L.W. Clarke, E. Johnson, G.L. Nemhauser, R.G. Shenoi, Flight string models for aircraft fleeting and routing, *Transportation Science* 32 No. 3 (1998), pp. 208–220.

[3] L. Clarke, E. Johnson, G. Nemhauser, Z. Zhu, The aircraft rotation problem, *Annals of Operations Research* 69 (1997), pp. 33–46.

[4] S.A. Cook, The complexity of theorem-proving procedures, in: *Conference Record of Third Annual ACM Symposium on Theory of Computing* The Association for Computing Machinery, New York, 1971, pp 151–158.

[5] T.A. Feo, J.F. Bard, Flight scheduling and maintenance base planning, *Management Science* 35 No. 12 (1989), pp. 1514–1532.

[6] R. Gopalan, K.T. Talluri, The aircraft maintenance routing problem, *Operations Research* 46 No. 2 (1998), pp. 260–271.

[7] N. Lingaya, J.F. Cordeau, G. Desaulniers, J. Desrosiers, F. Soumis, Operational car assignment at VIA Rail Canada, *Transportation Research Part B* 36 (2002) pp. 755–778.

[8] K.T. Talluri, The four-day aircraft maintenance routing problem, *Transportation Science* 32 No. 1 (1998), pp. 43–53.