Train Routing Algorithms: Concepts, Design Choices, and Practical Considerations*

Luzi Anderegg† Stephan Eidenbenz‡ ¶ Martin Gantenbein† Christoph Stamm†
David Scot Taylor§ ¶ Birgitta Weber† Peter Widmayer†

Abstract

Routing cars of trains for a given schedule is a problem that has been studied for a long time. The minimum number of cars to run a schedule can be found efficiently by means of flow algorithms. Realistic objectives are more complex, with many cost components such as shunting or coupling/decoupling of trains, and also with a variety of constraints such as requirements for regular maintenance. These versions of the problem tend to be $\mathcal{NP}$-hard, and thus the standard powerful tools (e.g. branch-and-bound, branch-and-cut, Lagrangian relaxation, gradient descent) have been used. These methods may guarantee good solutions, but not quick runtimes. In practice, two major railway companies, German Railway and Swiss Federal Railway, do not use either approach. Instead, their schedulers create solutions manually, modifying solutions from the previous year.

In this paper, we advocate an intermediate, semi-automatic approach. In reality, not all constraints or objectives can be easily formulated mathematically. To allow for interactive human modification of solutions, the system must work rapidly, allowing a user to save desired subsolutions, and modify (or just start over on) the rest. After careful examination to find which constraints and costs we can easily integrate into a flow model approach, we heuristically modify network flow based solutions to account for remaining constraints. We present experimental results from this approach on real data from the German Railway and Swiss Federal Railway.

*Work partially supported by the Swiss Federal Office for Education and Science under the Human Potential Programme of the European Union under contact no. HPRN-CT-1999-00104 (AMORE)

†Institute of Theoretical Computer Science, ETH Zentrum, Zürich, Switzerland **{lastname}@inf.ethz.ch**

‡Los Alamos National Laboratory, **eidenben@lanl.gov**
LA-UR:02-7640

§Department of Computer Science, San Jose State University, **taylor@cs.sjsu.edu**

¶Based on work performed while at ETH Zurich

1 Introduction

Train companies face many algorithmically challenging questions. One of the most important problems in terms of economic importance is to minimize the number of trains needed to fulfill a given schedule of train routes. This problem, known as the *fleet size*, *rolling stock rostering*, or *vehicle routing* problem, was modeled as a minimum cost circulation problem long ago by Dantzig and Fulkerson [3], for a survey see Desrosiers et al [4]. In practice, the problem is still solved by hand, because this flow model ignores many real-world constraints and objectives such as shunting, coupling/decoupling, and maintenance constraints.

In this paper, we report on realistic rolling stock rostering, as it occurs for large train operators such as German Railway (DB) and Swiss Federal Railway (SBB). We consider the standard minimum cost flow model, but we more carefully address what different objectives and constraints it can represent. We focus on finding efficient solutions, because in practical situations, all constraints are not explicitly formulated, and all costs are not well defined. Realistic problem instances are so complex that it would be useful to have a system which allows people to interact with proposed solutions, until a satisfactory solution is obtained. For instance, the train companies prefer "robust" solutions, which avoid propagating delays. Unfortunately, they cannot characterize the rules for robustness. With manual solutions, this robustness comes from a combination of intuition by the planners, and from the fact that tested, robust portions of previous solutions are reused.

Following a brief introduction to the problem (Sections 1.1, 1.2) is a description of the standard and the network flow approach (Sections 1.3, 1.4, 1.5). Next, Section 2 contains more extensive discussion of real-life problem variants. We consider additional requirements which can be modeled by flow in Section 3: these include more complex costs, different train types, and various station constraints. In Section 4, we address maintenance, which does not seem to easily fit into the standard flow model, and thus requires additional effort. We describe our heuristic modifications to the standard flow model to satisfy maintenance constraints. In Section 5, we describe our experiments and results using data provided by the German Railway (DB) and the Swiss Federal Railway (SBB).

1.1 Terminology We begin with some basic definitions. For our purposes, the smallest interesting entity of a train is a *train unit*: one or more railcars or locomotives which are linked together. Train units are routed and maintained as an entity, never to be broken into smaller units in the problems considered here. Depending on different kinds of railcars and locomotives and on the formation of the single cars, there can be different *train unit types*. A *train* consists of a number of train units coupled together. The *length* of a train is the number of train units which form the train. (The physical length of the train, a distance measurement, is also implicitly used in some of the train scheduling constraints.) Some of the train units can move in only one direction, whereas others can move in both directions. These bidirectional units are called *push-pull* trains.

A *route* is a path between two given end stations. Each route has a distance and a travel time associated with it. For our purposes we do not need to consider details such as intermediate stations, number of tracks between the two stations etc. A *ride* is a train unit on a certain route with distinct departure and arrival time. We distinguish between two types of rides: scheduled rides correspond to all the entries from a timetable, unscheduled rides are additional rides needed to operate the rail network. Unscheduled rides are further divided into three types: *empty, piggy-back* and *maintenance* rides. Empty and piggy-back rides are used to transfer train units without passengers from one station to another. Piggy-back rides refer to train units coupled to scheduled rides, while empty rides (also known as deadheading rides) refer to units riding alone to new locations. Maintenance is performed at maintenance stations, which may or may not be the end station of a route. Maintenance rides are used to shuttle the trains to and from maintenance stations, and can also represent maintenance itself. Usually piggy-back rides are cheaper than empty rides because of the costs saved for the crew and track reservation. A *rotation* is the order of rides performed by one train unit or a number of train units coupled together. A rotation has a *duration* and *length*. If a rotation takes one week, the train unit serving this rotation repeats the same rides every week. The length of a rotation is the total distance a train unit travels during a single cycle through the rotation. A *circulation* is the set of rotations for all train units.

1.2 Problem description The goal of *rolling stock rostering* is to determine a circulation for given scheduled rides. The input to the simplest version of this problem (rolling stock rostering with one train type and without maintenance) includes the scheduled rides with passenger demand, just one train unit type, and some cost functions for using specific train units of that type. The output assigns tasks to specific train units. This also defines how many train units are used for each ride, which empty and piggy-back rides are used to move train units between stations, and which cars will perform each of these tasks.

A general goal is to minimize the cost of the assignment. Of course, realistic costs are determined by a long list of factors, including the total number of train units needed, how much mileage they cover, crews used, number of couplings and decouplings, etc. The number of train units used is clearly one of the most important costs, and has been the primary focus of much previous work.

Besides these basic requirements, there are numerous additional requirements and constraints that may differ for each train company. We present some of these requirements for the specific cases of the German Railway (DB) and Swiss Federal Railway (SBB). Depending on the constraints considered, there are many versions of the problem, and these have rarely been treated in the literature. One of the most important constraints has to do with train maintenance: trains need regular inspections, maintenance work, and cleaning, all of which must be done at special facilities. Any solution that ignores maintenance requirements is useless to train companies.

1.3 Flow model Even the simplest version of the rolling stock rostering problem is usually broken into two separate phases: first, one solves the *train length* problem, in which the lengths of the trains on each route (including empty and piggy-back rides) are determined. The main goal here is to ensure that there are always enough train units available at each station for all scheduled rides departing from that station. In this phase rides are only assigned to train unit types, but not to specific train units.

In the second phase, the *train assignment* problem is solved: specific train units are assigned to the scheduled rides. For each train unit a rotation is determined.

A standard (and longtime) approach (see [2, 11]) for modeling the train length problem phase is to use a flow model: a periodic directed graph is used to model the scheduled rides. Each vertex in the graph represents one station at a specific time, that is, an $\langle s, t\rangle$ pair with station s at time t. A ride is represented by an edge, which leaves a vertex representing its departure

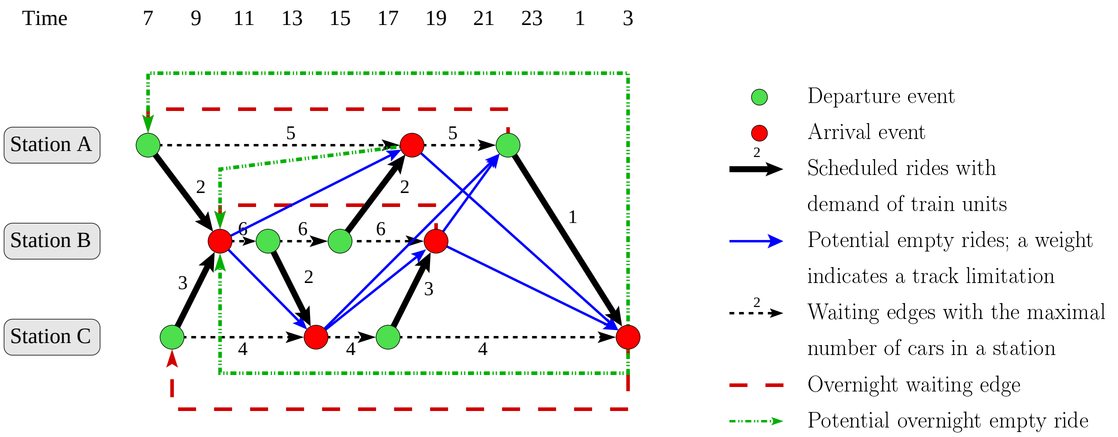

Figure 1: Example periodic graph.

Departure		Arrival		Demand of
Station	time	Station	time	train units
A	7:00	B	10:00	2
C	8:00	B	10:00	3
B	12:00	C	14:00	2
B	15:00	A	18:00	2
C	17:00	B	19:00	3
A	22:00	C	3:00	1

Table 1: Example timetable.

(station and time), and enters a vertex representing its arrival (station and time). Each edge has a lower and upper bound for the required flow where the number of flow units corresponds to the number of train units. The lower bound corresponds to the passenger demand for the associated ride, while the upper bound corresponds to track/station limitations and thus a maximum number of allowed train units. Moreover, edges are added to the graph to represent trains waiting in a station after their arrival and potential empty rides.

Example. A small one day schedule for a single train unit type is given in Table 1. We have three stations A, B, and C and seven scheduled rides. One row represents one ride, with departure, arrival station and times, and certain demands of train units. The corresponding graph is given in Figure 1. The number of train units that can wait within a station is five for station A, six for station B, and four train units can wait in station C. (To simplify the diagram, we have not put capacities on the overnight edges.)

Generally, this graph model is used to solve the train length problem, with the flow on each edge representing the number of train units. The periodicity of our graph model implies that some solutions can be repeated each period: for these solutions, each period will begin with the same number of train units in a station as the period before. Train assignment is done by breaking the train length solution into cycles, and assigning physical train units to each cycle. A cycle corresponds to a rotation for one specific train unit. These rotations may overlap each other, or contain an edge more than once when multiple train units are required for a route. For these “long train” edges, it is convenient to think of multiple edges, each of flow one, to distinguish among them (i.e. the first and second units of a train can be considered as separate rides, although they travel together).

As a result of this approach we get a *one period solution*. A one period solution does not imply that each of the rotations needs one period to complete, they can take much longer (they do need *at least* one period). Instead, it means that once any train unit consecutively performs two rides, then in every period those two rides will both be consecutively serviced by one train unit. If a rotation needs x periods to complete, it will use x different train units to serve the rides of its cycle within each period.

Example. Figure 2 shows a possible solution for the small example from Figure 1. The circulation consist of two rotations with three train units per rotation. Another representation of this circulation is pictured in Figure 3. Every rotation of one train unit is represented by one horizontal line. This representation is often used by train companies.

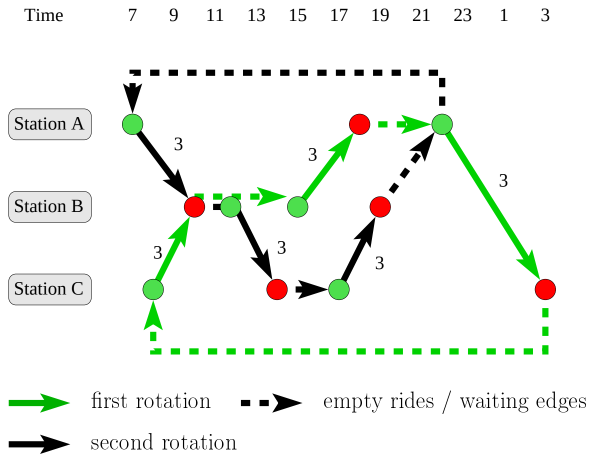

Figure 2: Two rotations.

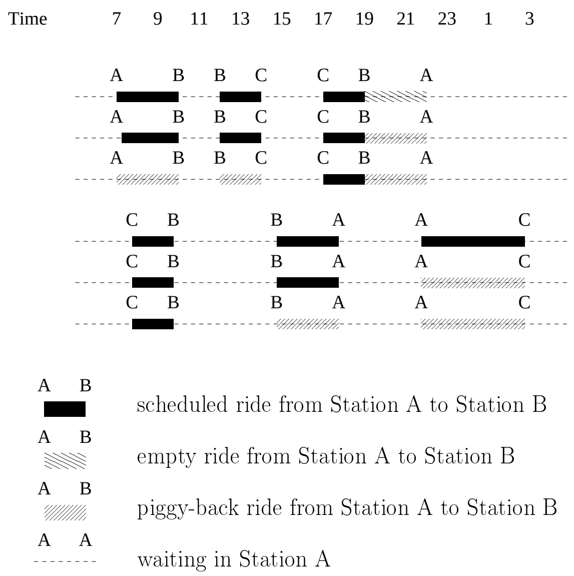

Figure 3: Interval representation of circulation.

1.4 Model limitations The flow model has several strong limitations, even for solving simple instances with no maintenance constraints. In theory it is possible for one period solutions to cost more than solutions that are allowed to span more than one period (see also [6, 7] for further description of these problems and the ones below). DB and SBB prefer one period solutions for their conceptual simplicity, so will only consider one period solutions as feasible. This simplifies our approach considerably: it allows us to ignore unexpected (and often overlooked) complexities inherent in other solutions.

In the past, purely theoretical approaches have overlooked other model limitations as well. For instance, depending on what types of empty ride edges have been used in the graph, past work has sometimes overlooked optimal solutions, while at other times it has allowed for solutions which break station capacity constraints. (Unless intermediate route stations are given, it is impossible to check station capacity constraints anyway.) Here, we do not claim to find optimal solutions: with realistic problems, such solutions seem well beyond our current abilities.

Finally, the train length and assignment problems should both be solved at the same time. Because the train length problem does not assign tasks to specific train units, many specific constraints (e.g. coupling times, maintenance) cannot be considered. Thus, a solution to the train length problem may not have any feasible train assignment solution, if all constraints are followed. For some constraints (such as coupling), we can the train length problem instances by adding coupling times to the routes, to ensure that coupling will not affect the feasibility of a train assignment. Unfortunately, this padding time may sometimes eliminate the optimal problem solution. Starting with Section 2, we describe some of these real world constraints and our approach which finds feasible solutions on realistic data.

1.5 Previous work The flow approach to this problem is intuitive, and mentioned as early as 1954 [3]; it is standard enough to be included in textbooks [1] and surveys [4], yet it is also still being studied in more recent articles [2, 11]. As for more recent work it is shown in [6] that the rolling stock rostering problem is $\mathcal{NP}$-hard to approximate arbitrarily closely; this result even holds for problems with highly simplified maintenance constraints, with all train lengths set to 1, and the costs equal to the number of trains needed. In [7], a variant of a proof from [9] shows that the train length problem is also $\mathcal{NP}$-hard, if arbitrary fixed costs are allowed. In this case (the train length problem), maintenance is not even a consideration, but instead more complex costs are allowed. In general, every realistic problem variant is difficult, but here we concentrate more on getting good solutions to real problem instances.

Periodicity of schedules has been studied for the simple case of minimizing the number of trains needed to implement the given schedule. Polynomial time solutions for the periodic case are known that follow very different approaches. Orlin [10] uses a periodic version of Dilworth's Theorem [5], whereas Gertsbakh and Gurevich [8] use the concept of a “deficit function”. Later, Serafini and Ukovich [12] propose a very general framework for periodic scheduling problems which also implies this result.

2 Real-life issues

This section further describes some of the real world problems which arise in rostering. As in Section 1.2, the primary data for all problems is a set of scheduled train routes, and the output must specify a set of trains and how they will be used to fulfill the input schedule. However, unless the basic model is extended to include additional costs and constraints, the solutions produced will be of little value to railway companies like DB and SBB.

Clearly, some of these changes require additional input: specific knowledge for each route (such as length, traveling time, whether or not the track will accommodate electric locomotives) is important. For every type of train unit, maintenance requirements and infrastructures, cost per kilometer to run the train unit, cost to maintain a train unit, crew costs, and coupling/decoupling costs must also be considered. In the following subsections, we describe some of the most important problem variants, constraints, and costs which arose in discussions with DB and SBB.

2.1 Requirements Here, we list two types of requirements which the standard model currently ignores. In Section 3 we show how to account for these additional requirements.

- *Non-identical train unit types:* Each scheduled ride must be assigned to one train unit type out of a predefined set of possibilities. Typically, each ride can be performed by three or four alternative train unit types, each of which may have a suitability ranking. Any solution must assign a suitable train unit type to each scheduled ride. It is, however, possible to append train units as piggy-back units to a scheduled ride, even if the piggy-back units are of not suitable types for the scheduled ride, as long as there are enough suitable units in the scheduled ride.

  Train unit types model the different parts of a train: if, for example, a certain train route goes from station A to B starting at time t, and the train consists of a locomotive, three first-class cars, six second-class cars, and a dining car, we would model this as four different rides. Each has the same station endpoints and times, but one ride must be satisfied by a locomotive train unit, while another requires first class cars, etc. In our graph model, this will result in a multi-edge with multiplicity four.

- *Minimum station turn around times:* If a train reaches its arrival station, it might have to be decomposed into its train units by decoupling and shunting; the train units are subsequently combined into new trains to carry out their next rides. The time needed to complete this procedure depends on the train unit types involved, on whether only coupling or decoupling or both are needed, and on the station topology.

  The time needed for this procedure is called station turn around time. These times are given as part of the input for each possible combination of coupling mode, train unit type, and station. Therefore, a solution of the train length problem may not have any feasible train assignment solution, since the turn around time at some station is violated. Any feasible solution must fulfill these timing requirements.

2.2 Maintenance requirements A feasible rostering solution must satisfy all maintenance requirements. The input information concerning maintenance consists of three parts:

- *Maintenance types:* Each train unit must have certain types of maintenance. The most common types are: refuel for diesel locomotives (T), interior cleaning (I+E), exterior cleaning (ARA), scheduled repairs (INST), and technical check-ups (V+A). If a train has to wait at a station for a longer period of time, in certain stations it is possible to park the train on a separate track. This is called siding (ABST), and will also be treated as maintenance.

- *Maintenance interval:* Each maintenance type must be performed on a train unit at certain intervals. These intervals can depend on elapsed time (since last maintenance), distance driven (since last maintenance), or on both. For example, interior cleaning should be done every 24 hours, exterior cleaning should be done before the train has run for 1,000 kilometers, and a technical check-up needs to be performed, whenever one week has passed or 10,000 kilometers have been driven (whichever occurs first). This last requirement is both time- and distance-dependent. Of course, it is always possible to perform maintenance before the interval is reached, but this increases costs.

- *Maintenance stations:* Maintenance stations are scattered rather scarcely all over the network. For each maintenance station we are given information of which types of maintenance can be performed for each train unit type, and hours of operation for the station. Moreover, capacity constraints are also given as part of the input and it is specified how long each maintenance type takes.

2.3 Costs Typically, railway companies have many "hazy" objectives concerning the rotations. For example, the train unit movement per period (measured in moving time and distance) should be more or less equal for all train units. This objective ensures an almost equal aging for all train units, but it does not imply that all rotations should be of the same period length. Another possible objective is the minimization of the number of different stations during one period or in one rotation. This may increase the chance that a specific train unit will move along the same line during one period and thus increase the regularity of a rotation.

More precisely, the overall objective of our problem is to minimize costs. Although many properties such as the ones above are desired, their costs are not well defined by the railway companies. Here, we consider costs that are easier to define and thus can be given as part of the input. We distinguish between three cost types (this list is not exhaustive):

1. *Fixed costs:* Fixed costs occur for each train unit that is used at some stage in the rostering. Fixed costs include all costs that are incurred by simply owning the train unit without using it at all; these costs include several items, but depreciation is the most important one. Traditionally, fixed costs are considered to be the most crucial. The quality of a train rostering solution is very often measured simply by the number of train units used. Train units are very expensive: aside from the capital investment to buy the equipment, even unused trains require thousands of dollars per year to maintain.

2. *Costs for each ride:* Each ride, whether scheduled, empty or piggy-back, incurs a cost that is composed of different cost factors. The most important cost factors are the following:

   - *Cost of energy:* Each train unit type has a certain cost associated to it for each kilometer that it runs and for each ton that is transported on this kilometer. Cost examples include power, fuel usage, or wear and tear of rails.
   - *Cost per time:* Each train unit type has a certain cost associated to it for each hour that it runs. Cost examples include heating, lighting.

   These cost factors are different for each type of ride, i.e., different for scheduled, empty, or piggy-back rides, with the empty rides being cheaper than scheduled, but more expensive than piggy-back rides. Moreover, these costs can occur either for each train unit in a train or for a train as a whole. For example, cost per time must be paid for a train as a whole to reserve rails for the train's passage.

3. *Maintenance costs:* Whenever maintenance is carried out on a train unit, costs are incurred. These costs depend on the maintenance type and on the maintenance station that performs the work.

3 Flow model modifications

Because of the numerous additional requirements that a real-life problem poses as explained in the previous section, our straightforward two-phase approach of first solving the train length problem by building a graph and finding a minimum cost flow on it and then solving the train assignment problem by extracting rotations from the minimum cost flow may no longer be a viable approach. In this section, we consider some of the additional requirements which *can* be treated by this approach by making modifications to the graph built for the train length problem.

3.1 Multiple train unit types We integrate the concept of having several train unit types by first determining an order in which the different train unit types will be processed. We schedule the non-locomotive train unit types before mixed and locomotive train units; within these categories, inexpensive train unit types are scheduled first.

So, given the train unit ordering, we simply iterate the standard flow model on the cheapest unit first, for all routes where that car type can be used. Once we have found a solution for a certain train unit we then iterate on the remaining routes.

Many complications are hidden here: for instance, if the available number of a certain type of train unit is limited, perhaps there are not enough to cover all routes for which that train unit is useful. This will leave more routes for the next iteration, where a "better" unit may be used to satisfy the demand. Also, the number of non-locomotive train units on a route may determine the type and number of locomotives needed to pull those cars. Finally, in order to offer inexpensive piggy-back opportunities for a train unit type, edges from previous iterations, with appropriate weights, will be included.

3.2 Minimum station turn around times Station turn around times can be integrated into the flow model as follows: after having obtained a valid rostering from our approach, we check in the solution to see whenever minimum station turn around times have been violated. If they are violated, we artificially delay all incoming edges by the amount of time needed to carry out a coupling operation and thus create a new vertex; this has the effect that the train unit on this incoming edge then has a higher probability of linking to another ride without violating minimum station turn around times. This yields a new graph, and we iterate this procedure until no violations occur.

The number of iterations is bounded in our instances because the minimum station turn around time is always less than four couplings. Thus, each vertex in the underlying graph will be delayed at most four times. In our experiments, to save calculation time, we added the maximum needed delay to vertices with violations after the first iteration.

We do not simply add the delay to all stations, as at some stations, the delay is quite high ($\sim$1 hour), and to do so would require extra train units in the rostering. An other important point is that station turn around times vary not only by station but also by type (coupling or decoupling).

3.3 Additional costs We can model fixed costs of owning each train unit by simply making one unit of flow on the overnight-edges cost as much as the fixed costs for one train unit. Similarly, we can compute the costs for each each ride by computing and combining the costs per distance, time and ton kilometer for each train unit type. However, the costs that are incurred only for a train as whole (e.g. coupling/decoupling costs) and not for each train unit type are harder to model exactly. As an approximation, we allocate these train costs to a specific unit of the train, if possible the locomotive. This can lead to extra costs in our calculations if two or more locomotives pull a train, but these should be small compared to others.

4 Maintenance

Compliance with maintenance requirements is crucial to making routing solutions feasible: while ignoring certain real-world costs may lead to suboptimal solutions, solutions which ignore maintenance are invalid altogether. Maintenance is ignored by the standard minimum cost flow algorithm, yet it is of topmost concern to the railway companies, greatly affecting the cost and practicality of scheduling trains. Lack of maintenance in solutions produced by automated systems may be the primary reason that railway companies still schedule trains manually.

Maintenance is performed at special maintenance-capable stations, with limited hours of operation and bounded work capacity. Even in practice, maintenance is not always scheduled in advance: it is not practical to expect future plans to be followed precisely indefinitely. Unexpected changes such as extra trains scheduled for special events, or train break-downs, happen frequently enough that trains occasionally get shuffled to cover for each other. It is more pragmatic to only schedule more frequent maintenance in advance.

Maintenance frequency may not match that of a rotation very well: it may be that a natural schedule for a train covers the same routes each week, for 5,000 total km, yet the train might need to be maintained once every 10,000 km. Although it seems clear that the train should just be maintained once every 2 weeks, this may disrupt the scheduled 1 week rotation. This adds to the complexity of scheduling maintenance.

When considering maintenance, the straightforward two-phase approach seems to break down. While many of the additional requirements can be accounted for by making modifications to the underlying graph in the network flow phase (as discussed in Section 3), the crucial maintenance requirements seem evasive to integration into the two-phase approach. We thus introduce a third phase into our approach that we call maintenance insertion. There are several approaches possible as to how to integrate maintenance in this third phase: some iterate over all three phases of our solution approach, others simply add a single phase after the two other phases have been completed. We propose three different approaches :

1. If the maintenance requirements are simple to perform (e.g. cleaning) — they are available in most stations and do not take long — it is possible to schedule them locally within a station. This approach and its weaknesses are presented in Section 4.1.

2. Train companies do not normally consider maintenance to be part of the rotation itself. Instead, they keep an over-stock of approximately 10% of trains, and try to keep these units maintained, and available within maintenance stations. They are swapped into a rotation as an unmaintained train is swapped out of it. This gives them flexibility to not fully schedule all maintenance far in advance. This approach is described in Section 4.2.

3. The last approach combines the first two approaches and is described in Section 4.3.

Approaches 1 and 2 work iteratively. It is necessary to recalculate the minimum cost flow circulation and extract new cycles after fixing certain maintenance constraints. The complexity of the problem instance will determine which approach is most useful.

4.1 Locally added maintenance If the underlying network structure is rather simple, i.e., if it has abundant, well dispersed maintenance stations, we can hope for the best: the rotations obtained from the standard two-phase approach may already fulfill all or most maintenance requirements as the train units sometimes remain at maintenance stations for long enough time periods to perform required maintenance. In this case, a promising approach for integrating maintenance is to add "maintenance edges" (described below) into our our network flow graph. Using this approach, we repeat the two-phase solution on the new modified underlying graph until all needed maintenance is performed.

This "local fix" approach iterates the following basic steps:

1. Solve the train length problem using network flow.
2. Compute a train assignment to the train length problem.
3. Test the solution for maintenance requirement violations.
4. Introduce new maintenance edges into the graph.

Once the solution no longer violates any maintenance requirements, the algorithm stops. All steps except for the introduction of maintenance edges work exactly as in the basic approach described earlier. We introduce new maintenance edges into the graph as follows. Each rotation R consists of events $v_1, \ldots, v_{|R|}$. Each event v_i is a pair $\langle s, t \rangle$, where s is a station and t is a time. Considering the rotation from v_1, we determine at each event whether or not any maintenance requirements of a particular train unit have been violated. If there is a violation at event v_i, we consider if any of the following hold for v_{i-k} and k:

1. Maintenance performed at the station of event v_{i-k} will prevent any maintenance violations at event v_i.
2. The station of event v_{i-k} is capable of performing the maintenance required by event v_i. Besides being an appropriate maintenance station, it must also have enough capacity at the given time.
3. The train spends enough at the station of event v_{i-k} to perform the required maintenance.

If all these points can be answered affirmatively, we schedule maintenance at event v_{i-k}. We can use the first point above to decide whether it makes sense to check back further (that is, check higher k values) in the rotation. If it no longer makes sense to go back in the rotation, we find the maintenance station closest to event v_{i-1}. This station has to have the capacity to carry out the required maintenance and should be reachable from event v_{i-1} without violating the maintenance requirement. Given such a station, we introduce a new event v'_{i-1} at this station and connect v_{i-1} to it by an empty-ride edge (thus implicitly determining its time); we then introduce a new event v''_{i-1} at the maintenance station at a later point in time such that an edge from v'_{i-1} to v''_{i-1} allows enough time to carry out the necessary maintenance. Finally, we connect event v''_{i-1} to the next possible event on the original rotation by a third edge. If we cannot find a legally reachable maintenance station from event v_{i-1}, we consider v_{i-2} (or v_{i-k} as needed).

To encourage the minimum cost flow algorithm to use this maintenance detour through events v'_{i-1} and v''_{i-1} into the solution in the next iteration, all three edges have cost zero, and the minimum flow requirement for edge (v'_{i-1}, v''_{i-1}) is set to one.

Example. An example of this approach is illustrated in Figure 4. In Station B at 12:00 the maintenance require-

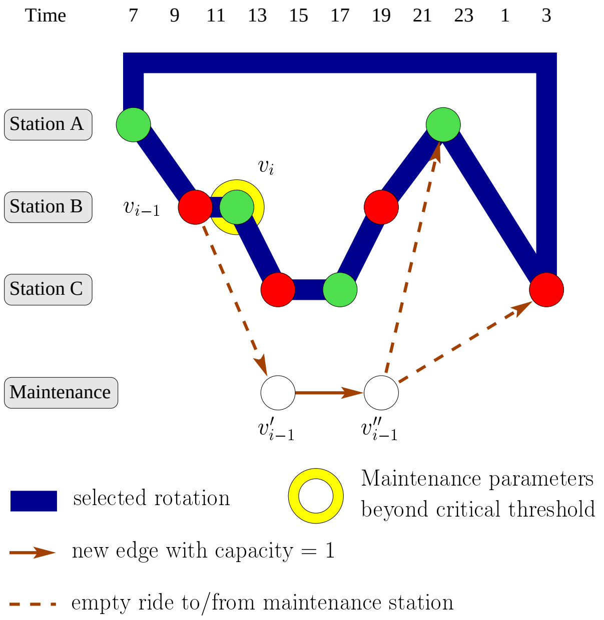

Figure 4: Locally added maintenance.

ments are not fulfilled. New edges to a maintenance station are introduced to the graph and an additional edge within a maintenance station has capacity one. After the maintenance is done, edges to every station — representing possible empty rides — are introduced as well.

This local fix approach seems to work quite well for problem instances in which maintenance really is not the central issue, with simple to follow maintenance constraints. In these cases, minor detours will allow all needed maintenance. Unfortunately, maintenance requirements are rarely so simple.

4.2 Extra trains for maintenance In this approach, we try to model a strategy of the railway companies: always try to keep a fully serviced spare train unit (of any type needed) available at each maintenance station. These trains will be swapped into rotations as needed to eliminate maintenance violations. The first and foremost goal is to use as few spare trains as possible. This approach is non-iterative and consists of solving the train length and assignment problems without maintenance, and then adding extra trains into the assignment to alleviate maintenance requirements.

To add these extra trains, we proceed as follows: For a selected rotation R we order all vertices in the graph according to their time, which results in an ordered list R of vertices $v_1, \ldots, v_{|R|}$. We then process the resulting list R vertex by vertex, where we check at vertex v_i, whether the maintenance requirements of the train units going through vertex v_i have reached some critical threshold with respect to maintenance types. If this is not the case, we proceed to the next vertex; if it is the case, we replace the current train unit by a replacement unit, which was swapped out of a rotation when it needed service sometime in the past. The train unit still requiring service is moved to a maintenance station and serviced. After that, it will be routed to some other station, where it becomes the replacement for another unit requiring service. If no replacement unit can be routed to a rotation when needed, an additional "service train" is added to the system, increasing the total number of extra units used.

After processing all vertices all vertices from list R, we link each train at the end of the period to a vertex v_j at the beginning of the period such that the resulting maintenance parameters will satisfy the maintenance requirements of vertex v_j. This will result in feasible rotations that satisfy all maintenance requirements.

In this approach, we keep the original cycles intact, which mimics a strategy of DB and SBB. It can be thought of as solving the rostering problem with virtual trains, needing no maintenance. The number of extra trains needed to maintain this illusion increases with the difficulty of the maintenance constraints. While conceptually simple, examination of implementation details shows hidden complexities, especially concerning how to best link the start and end of the rotations. Simply linking the first and last stations in a cycle would not allow to "start" the cycle with a maintained train, and some care must be taken to keep feasible solutions while not requiring an extra maintenance.

4.3 Iteratively fix rotations with maintenance Our last approach to maintenance combines some aspects of both preceding strategies. As in Section 4.1, we try to fix violations by adding maintenance into cycles for free when possible, and by routing trains to maintenance stations when it is not. Once maintenance is completed on the train, we move this train back into the same cycle, allowing the cycle to continue later on with a freshly maintained train as in Section 4.2. This altered cycle will not service every route on the original cycle, but the routes it does cover will be serviced with maintained trains. These routes are removed from the schedule, and solutions for the remaining routes, excluded from the altered cycles, will be iteratively found by starting over.

Example. An example of this is shown in Figure 5. In Station B a maintenance edge is added after the

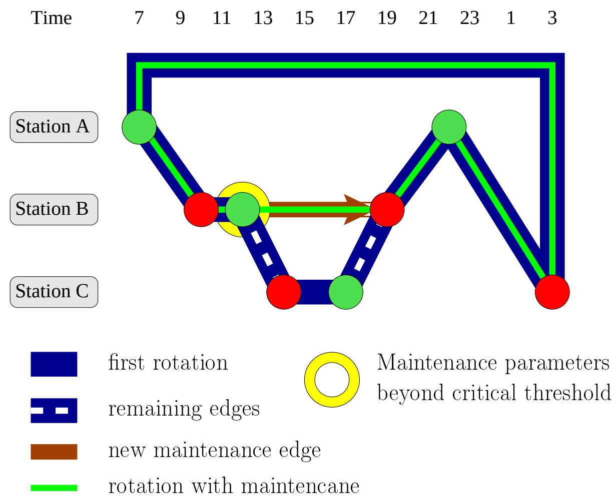

Figure 5: Fixing maintenance constraints.

maintenance parameter got beyond a critical threshold. The rotation with maintenance meets the old rotation again in Station B at a later point in time. The remaining edges will be covered in one of the next iterations. In practice the number of iterations should be small.

Of the three approaches, the final one worked best overall for our test data. This approach evolved from the first two: during implementation it became clear that the first two approaches would not work well on complex real-world data. Therefore, only the third approach was implemented and used for all experiments.

5 Experiments

Here we discuss our test data and experiments. The major goal of these experiments was to carefully revise our model, test its practicality, and to ensure that important details were not being glazed over as they had been in previous, purely theoretical, treatments.

The overall goal in each experiment is the same: the minimization of the total costs. However, some of the test data given to us by the railway companies was chosen to test the minimization of train units, while other test sets were designed to test whether or not our maintenance strategies give feasible solutions.

Although the overall goal is the minimization of the total costs, our results summarized in Table 2 do not show these cost values. The railway companies have not disclosed their precise current solutions or their costs. Without the real values a comparison is not possible. We do compare the number of train units, which is the most important cost factor.

5.1 Test sets We used five different test sets in our experiments, four of them were supplied to us by DB and one by SBB. To simplify further discussions on these test sets we enumerate them:

- *BR112:* This set contains only locomotives of the same train unit type, called Baureihe 112. A train unit in this test set is a single locomotive. The given schedule is a subset of the DB timetable with 2098 routes (each specified only by its end stations). While this test set is quite large, the maintenance requirements are not overwhelming, and the main objective is to minimize the number of train units.

- *BR612:* This test set only contains diesel railcars (self-locomotive cars) of the same train unit type Baureihe 612. Thus, the train unit consists of just one railcar. The given schedule is a subset of the regional timetable Saarland-Westpfalz in Germany with 332 routes. The difficulty here lies in the frequent refueling requirements of these locomotives, as there are only a small number of fueling stations, and they have limited capacity.

- *BR411:* This test set consists of two different Intercity Express (ICE) train units. The train unit type Baureihe 411 is a composition of seven cars including a locomotive. The train unit type Baureihe 415 is a similar but shorter composition. Two train units of the same type can be coupled together, but a substitution of a train unit of one type by the other is rarely possible. Both train types use the same maintenance infrastructures with capacity and time limitations. Testing maintenance feasibility is a main goal. The schedule is a subset of the international timetable between Germany and its neighbor countries with 496 rides. 330 of these rides must be conducted with train units of type Baureihe 411.

- *BR218:* Our last DB test consists of all locomotives of the train unit type Baureihe 218 and their associated passenger cars in the region Schleswig-Holstein. The given schedule contains 7666 routes of up to 32 different train unit types. Some of these 32 train unit types are only used as possible substitutions in case of car capacity problems. Here, we must schedule both train cars and locomotives, adding to the complexity of the problem. Additionally, the number of some train unit types are limited, and we must find appropriate substitutions in case of car capacity problems.

- *BR1210:* Our last test set consists of all self-locomotive train compositions of the Zurich S-Bahn. A train unit in this test is a composition of several cars with locomotive. S-Bahn trains are either one or two coupled push-pull train units. The given subset of the SBB schedule contains 6151 routes. This test set is the only one which supplies information about the type of turn within a station. Because all train units are composed the same way, the additional information allows the computation of the first class car positions within each station. Besides minimizing the number of train units used, the position of the first class cars within the station should be the same from day to day on any route. Maintenance information was not supplied.

5.2 Results The most interesting results of these test scenarios are the rolling stock rosters, but they are much too large to present here. We show a cutout of one of the rosters, and some values to summarize how efficient our results were.

Our software is built of 20,000 lines of C++ code. Besides standard libraries we only used LEDA (Library of Efficient Data types and Algorithms)[1]. We have run our tests on a PC laptop with a 1.6 GHz Mobile Pentium 4 processor, 512 MByte of RAM under Windows XP.

In Figure 6 you see a typical cutout of a graphical representation of a rolling stock roster. Several different depictions are meaningful; this format matches the one used by SBB. On the x-axis we see the time between 4:00 of the third day in period until 3:00 of the next day. Each line represents one train unit over one period and the bars on that line show the planned rides of the train unit. On the second line, for example, we see four

[1]LEDA has been developed at the Max-Planck-Insitut für Informatik, Saarbrücken (http://www.mpi-sb.mpg.de/LEDA/) and is available at http://www.algorithmic-solutions.com/.

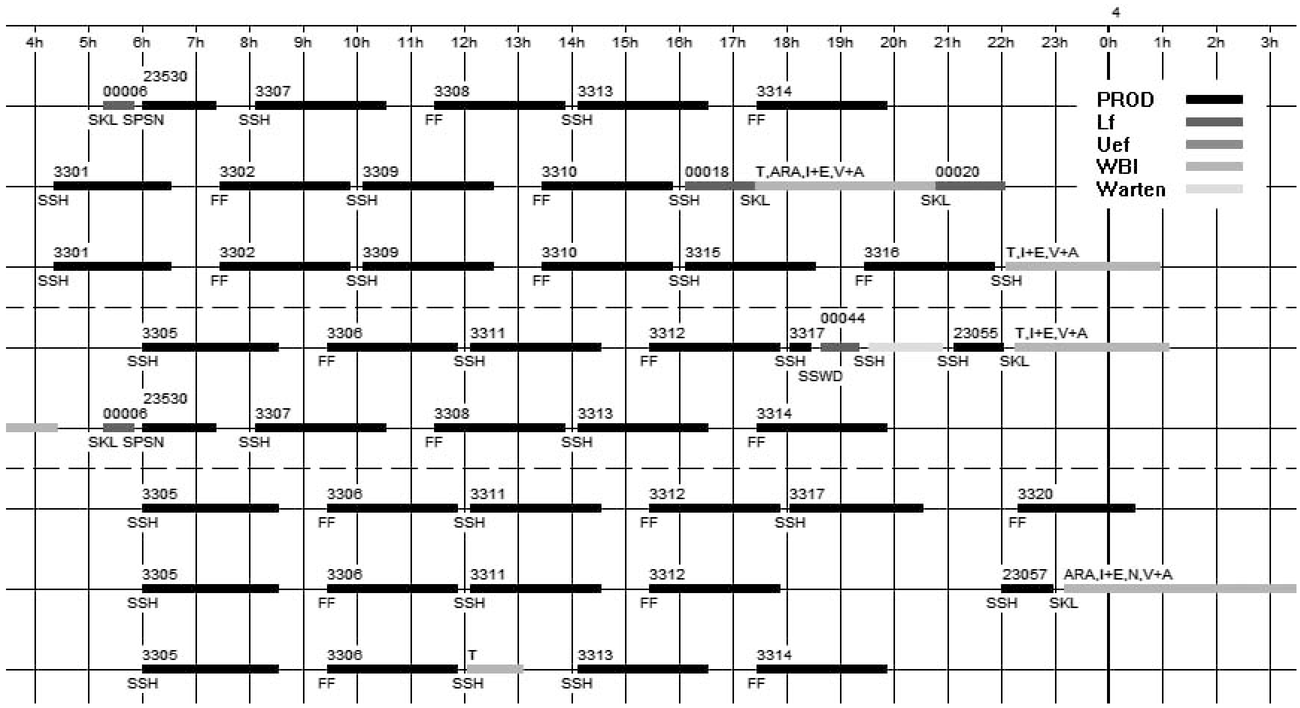

Figure 6: Graphical representation of a rolling stock roster.

consecutive productive rides starting and ending in SSH followed by an empty movement to SKL where the train unit is maintained. Several maintenance operations (T, ARA, I+E, V+A) are planned between 17:30 and 20:45. Finally, the train unit makes an empty movement to the next station where it is needed. Above the bars we see train identification numbers, labeling which train units belong to the same train. For example, the train 3305 starts at 6:00 in SSH and ends at 8:30 in FF. This train consists of four separate train units. The dashed horizontal lines separate train units of the same rotation. For example, the first three train units are in the same rotation of length three.

In Table 2 we summarize our results of the five test sets. As would be expected, ignoring maintenance constraints allows the automatic construction of a solution at least as good as the ones in use. Unfortunately, our results with maintenance are not as good. In the two test sets with the most maintenance constraints, BR612 and BR411, our results with maintenance suffer the most, 33% worse than the current solution of the railway companies. Especially in the case of BR612, this is not surprising: refueling is needed so frequently that it must be incorporated very efficiently into the schedule. Pulling a train out of rotation and traveling 50 km extra for each 10,000 km maintenance is very different than doing it for a 500 km tank of fuel.

These results indicate that our approach to maintenance is still not sophisticated enough. The extra train units for our system come from several sources. First, the rides removed from a rotation very often contain multi-edges. Each removed ride may then require an additional train unit. Next, the current approach does not check whether remaining routes can be combined with existing rotations. The integration of these two points into the existing *Iteratively fix rotations with maintenance* approach is part of future work.

This approach to maintenance attempts to fully automate the rostering process. Our program would need additional modifications to be used in an interactive, semi-automated way. This would be a natural "first step" to take before trying to use it in a fully automatic way to plan all rostering decisions, and our approach has been to make the program fast to make this interaction possible. Using the flow model, it is also simple to "freeze" parts of the solution and change others.

6 Conclusion

We have presented an extended and therefore much more practical version of the standard fleet size problem. We have shown that many extensions of the rolling stock rostering problem can be well integrated into the standard flow model approach. For a constraint which we cannot easily integrate (maintenance), we

Test set	Graph: $\lvert V\rvert$, $\lvert E\rvert$	Number of train units: in real solution	our solution without maintenance	our solution with maintenance	Runtime in seconds maintenance version
BR112	12521, 90050	66	52	55	35
BR612	1025, 165266	9	9	11	14
BR411	411: 3602, 1066810	28	27	35	20
	415: 1124, 214768	8	8	10	6
	Total	36	35	45	26
BR218	C1: 3005, 11982		6	7	6
	C2: 3655, 14075		9	10	29
	C3: 6891, 40459		10	10	10
	C4: 304, 623		1	1	0
	C5: 6273, 22383	68	6	8	8
	C6: 3013, 12703		5	6	6
	C7: 7048, 30324		11	12	8
	C8: 574, 1351		2	3	2
	C9: 8508, 62435		16	19	27
	218: 22628, 368991	72	57	84	81
	Total	140	123	160	177
BR1210	36895, 319211	120	79		79

Table 2: Table of results.

have developed some heuristics to modify the standard flow approach.

We have implemented our three phase approach and tested it with real data from the German Railway and Swiss Federal Railway. Although our solutions are worse than the rostering solutions already used by those railway companies, our experiments are encouraging. It would be overly optimistic to assume that the first attempt of a fully automated system would improve upon long tested and used solutions. Each year, many person-years of work are used to refine rostering solutions for minor scheduling changes from the previous year. Over the years, it may well be that minor scheduling changes have also been made to accommodate the rostering solutions. Although finding improved fully automated solutions is an ultimate goal, a more immediate goal was to build a system which could be used interactively, by scheduling personnel, to help in their jobs. Given an automated solution, the scheduling personnel can extract partial solutions which they believe look promising, and then run the system iteratively on the parts which looked worse. Given our quick runtime, on modest equipment, this interactivity is quite feasible.

7 Acknowledgements

We would like to thank German Railway and Swiss Federal Railway for sharing their rostering problems and data. Special thanks go to Frank Wagner, Martin Neuser, Jean-Claude Strueby, and Daniel Hürlimann for their support.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows - theory, algorithms, and applications.* Prentice Hall, 1993.

[2] P. Brucker, J.L. Hurink, and T. Rolfes. Routing of railway carriages: A case study. In *Memorandum No. 1498, Fac. of Mathematical Sciences.* University of Twente, 1999.

[3] G. Dantzig and D. Fulkerson. Minimizing the number of tankers to meet a fixed schedule. *Naval Research Logistics Quarterly*, 1:217–222, 1954.

[4] J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. In *Handbooks in OR & MS*, volume 8, pages 35–139. Elsevier, 1995.

[5] R. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51:161–166, 1950.

[6] T. Erlebach, M. Gantenbein, D. Hürlimann, G. Neyer, A. Pagourtzis, P. Penna, K. Schlude, K. Steinhöfel, D. Taylor, and P. Widmayer. On the complexity of train assignment problems. In *ISAAC: International Symposium on Algorithms and Computation, LNCS.* Springer-Verlag, 2001.

[7] M. Gantenbein. The train length problem. Diploma Thesis, Department of Computer Science, ETH Zürich, 2001.

[8] I. Gerthsbak and Y. Gurevich. Constructing an opti-

mal fleet for a transportation schedule. *Transportation Science*, 11:20–36, 1977.

[9] S. Hochbaum and A. Segev. Analysis of a flow problem with fixed charges. *Networks*, 19:291–312, 1989.

[10] J.B. Orlin. Minimizing the number of vehicles to meet a fixed periodic schedule: an application of periodic posets. *Operations Research*, 30:760–776, 1982.

[11] A. Schrijver. Minimum circulation of railway stock. *CWI Quarterly*, 6(3):205–217, 1993.

[12] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM J. Discrete Mathematics*, 2(4):550–581, 1989.