Car Routing Algorithms: Concepts, Design Choices, and Practical Considerations*†

David Scot Taylor Martin Gantenbein Luzi Anderegg Birgitta Weber
Peter Widmayer

Institute for Theoretical Computer Science, ETH Zürich.‡

Abstract

Routing cars of trains, or other vehicles, is a problem that has been studied from the first days of operational research. It has been observed early on that a minimum number of cars to run a given schedule can be found efficiently, by means of flow algorithms. In practice, the objectives are more complex, with lots of cost components such as coupling or decoupling of trains or shunting, but also with a variety of constraints such as requirements for regular maintenance. Most of the realistic problems, defined by an objective and constraints, turn out to be NP-hard in general. Hence, it is natural that an abundance of powerful techniques for solving hard problems have been put to use, such as integer linear programming, branch and bound, branch and cut, Lagrangian relaxation, or gradient descent methods, to name but a few. These methods guarantee (or tend) to deliver good solutions, but might run for a long time to get them.

In this paper, we advocate a different approach altogether. The reason is that in reality, not all constraints and not all objectives can be easily formulated mathematically. Instead, the most important ones are expressed, and then a human user plays interactively with the optimization program, modifying the problem instance repeatedly, or modifying and freezing some part of the solution, before repeating the optimization for the rest. A system that supports this mode of interaction must run the individual optimization step rapidly. We therefore choose to base the optimization of each and every objective under any constraints whatsoever on network flow and circulation solutions. We show that this approach is worthwhile, by inspecting real problem instances as to their characteristics, and by studying under what conditions a flow based algorithm creates a good solution for a problem that is NP-hard in general.

1 Introduction

We consider the problem of assigning trains to implement scheduled train routes. Such problems are typically solved by hand, though the core of the problem has been modeled as a

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

†E-mail: `mgantenb,anderegg,weberb,widmayer@inf.ethz.ch` Contact: `dstaylor@alum.mit.edu`

‡Building CLW, Clausiusstrasse 49, CH-8092 Zürich, Switzerland.

minimum cost circulation problem long ago by Dantzig and Fulkerson [DF54]; for a survey see Desrosiers et al [DDSS95]. Since then, many have approached the problem, both with theoretical papers and with experimental systems, yet these systems are still rarely used in practice. The reason is that in real life systems, a multitude of constraints come into the picture, and the objective of the optimization is not as simple as theory would like. Therefore, solutions from theory either use fairly general and inefficient techniques, such as linear programming based methods, or they can be applied in very special practical situations only.

We attempt to bridge this disturbing gap between theoretical results and practical systems by formally examining the shortcomings of some current theoretical models that aim at efficiency, and by showing in detail which constraints and costs can still be handled efficiently, and what others lead to hard problems. Even for theoretically hard problems, practical instances might still be solvable within reasonable time limits. We will discuss examples for which this is the case.

We are focusing on the *efficiency* of our solutions, simply because we feel that in practical situations, not all constraints will be formulated explicitly, and not all ingredients in the objective of the optimization will be given in advance. The real problem is just too complex, and a system should allow people to interact and play with variants of the problem instance, until a satisfactory solution is obtained. For instance, "robust" solutions are sought, that is, if one train is delayed, hopefully the delays will not propagate through the system. Unfortunately, rules for robustness are not so well characterized. With hand solutions, this robustness comes from a combination of intuition by the planners, and from the fact that tested, robust portions of previous solutions can just be reused.

The reasons why such systems are still not automated are complex. Besides the normal communication gap between theory and practice, there is a certain inertia within industry which makes it difficult to change a system which has worked well traditionally. As the saying goes, "if it ain't broke, don't fix it". In truth, the railways do not seem to know if their solutions are "broke": with only hand-made solutions, there are no comparisons to be made. There are no estimates for the cost of optimal solutions.

To change this, large obstacles must be overcome. Risk is inherent in any change, and any method which changes how trains are assigned to routes may have unwanted side effects. Part of the problem is that the objective functions to be optimized are not well defined. While clearly one wants to minimize costs, other constraints are not so well defined, or even well understood. Consider that from year to year, train schedules tend to only change incrementally. If a rostering solution to a schedule from a past year works well, one would expect that a similar solution to the new year may also work well. This is without even understanding what properties the previous schedule held.

Even with these risks and complexities, the time is right for change. Increasingly, railways are being run by private companies. Previously, much of what drove the railways was a question of what services can be provided to the customers, but as companies try to make profits (and governments try to save money), the focus is moving towards lowering costs for those services. This, however, asks for a system that responds immediately, thus allowing the planner to interact.

We will consider a standard model which uses flow algorithms to solve the problem, and consider what costs and constraints each model can represent. We start with a theoretical view, and show that for limited constraints and costs, the circulation formulation gives an optimal solution. Next, we show how these bounds can be used in specific problem instances to prove performance of a heuristic system which accounts for more costs and more constraints. While the more complex system can model $\mathcal{NP}$-hard problems, we can still show that when applied to data from the Swiss Federal Railway (SBB), our heuristic gives good performance. Although the problems are provably hard to optimally solve for arbitrary instances, real problem instances are not nearly so malicious and appear ripe for automated, near-optimal solutions.

Even though some of the flow algorithms are "efficient", some of the railway systems are large enough that the tremendous increase in computing power of the past decades was a necessary precursor to automated systems. It is time to better figure out *what problems the various theoretical approaches can and cannot model.* To do this, we need to work interactively with railway companies, with real data, to avoid missing important underlying issues: optimal solutions to current flow models do not necessarily optimally solve the train assignment problem, because the models are not powerful enough to account for all costs and constraints within the system. These shortcomings are often ignored, thus no guarantees of true performance for real systems are given. Currently, we are working with both the Swiss (SBB) and German (DB) railway companies to determine how well flow based approaches to the problem work; we report our experiences with them throughout the paper, but they especially influence the discussions of Sections 2 and 4.

1.1 Problem description

The goal of *rolling stock rostering* is to assign movements to a set of trains which will fulfill a predetermined schedule of train routes. The input will include the scheduled routes, and the output will assign trains to those routes, and will determine how many trains should be sent on each route, which specific trains will perform each route, and which empty (deadheading) movements are used to move trains between stations. A general goal is to minimize the cost of the assignment, where the costs are determined by a long list of factors, including the total number of trains needed, how much mileage they cover, crews used, number of couplings and decouplings, etc. The number of trains used is clearly one of the most important costs, and has been the primary focus of much of previous work, in which the problem is also known as the *fleet size* problem, or the *vehicle routing* problem (see also [EGH$^+$01a, EGH$^+$01b]).

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 cleanings, all of which must be done at special facilities. Any solution which fails to address this is of limited usefulness to the train companies.

The problem is often broken into two separate parts: first, one can solve the *train length* problem, in which the length of the trains on each route (including deadheading movements) are determined. The main goal here is to make sure that there are enough trains available at all times at each station to satisfy all tasks assigned to trains at that station. Here, tasks are assigned to trains in general, but not to specific trains.

In the *train assignment* problem, specific trains are assigned to the tasks for each station. Thus, given a train assignment, one can determine what the next movement is for a specific train, given its previous movement.

Ideally, these problems should be solved at the same time. Because the train length problem does not assign tasks to specific trains, train specific constraints such as maintenance cannot be considered. Thus, a solution to the train length problem may not have any legal solution, following all maintenance constraints, in the train assignment problem. Further, with arbitrary costs, we show that even the train length problem is hard to solve in general. On the positive side, we show that good solutions are attainable for both parts of the problem on realistic data.

1.2 Our contribution

In this work, we review the proposed flow-based solutions to the rolling stock rostering problem, and evaluate their strengths and weaknesses. We delve deeper into the solution structure of the problem than previous work, to better characterize flow algorithm limitations. We review our work with railway companies, and try to determine what structure among the input data will guarantee that our flow based algorithm, plus heuristics, will give good performance solutions.

In Section 2, we begin by defining the problem more completely, with regards to the input, constraints, and output. In Section 2.3 we show how the train length problem is often modeled as a graph/circulation problem, and discuss some of the problems to this approach. We will study one specific class of solutions (Section 2.4), used in past work, although this restricted solution type may come with high costs. Nevertheless, it is favored by the railway companies because it has additional structure which is simpler to follow. We finish the section discussing some past work.

In Section 3, we begin with a quick review of hardness results for the problem. Next, we spend some time considering the mathematical structure of the general solutions to the problem, and of the limited solution types introduced in Section 2.4. This section is rather theoretical, and seems quite far removed from practice, but this in depth look is needed to later show what input instances can be well solved, that is, solved almost optimally.

Given flow based solutions to the train length problem, we discuss some heuristic approaches for train assignments in Section 4. We are currently working with both SBB and DB to see how useful these solutions are. Here, we explore what input instance structure is useful for our heuristics to perform well. From our talks with SBB and DB, this structure seems to hold for real life cases.

We conclude, and discuss ongoing work in Section 5.

2 Problem

Here, we discuss the basic problem, and then list some of the variants and constraints which need to be considered. In all versions, we are given 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. Generally, intermediate stops along a route are not relevant, and therefore a route is specified only by its initial and final stations. We consider *periodic* schedules, where it would be normal to consider daily or weekly periods. With slight modifications (i.e. extra trains for special events) train schedules are very regular, repeating the same weekly schedule for several months. Even within a week, Monday through Thursday may have very similar schedules. The algorithms we consider will only produce a specific class of output types (introduced in Section 2.4), with additional structure which makes combining several short solutions into larger ones easier. We discuss this combination process in Section 4.

Clearly, additional input, beyond just the schedule, is needed: distances between stations, time to get from station to station, maintenance schedules and locations, cost per kilometer to run the train, cost to maintain a train, cost of a train over its expected lifetime, crew costs, and coupling/decoupling costs are some that spring to mind, depending on which variant is to be solved.

The most basic problem we will consider here is the rolling stock rostering problem of identical cars, with regular maintenance. Only limited types of costs are considered, namely those costs which are linear in the number of trains to run any particular route. We explain other variants, and how they relate to this basic one, below. First, we discuss the maintenance requirements.

2.1 Maintenance

Railway cars, and especially locomotives, need regular maintenance. There are several types of maintenance, and they occur at different frequencies. For the sake of our discussion, we will refer to any service that requires a special facility as "maintenance". This will include real maintenance, such as brake replacement, but also inspections, cleanings, or even refueling.

Maintenance requirements can be based on both time and mileage, and can vary greatly. Requirements may vary from short term ("Trains need to be cleaned every few days" or "Diesel trains need refueling every few hundred kilometers") to long term ("Trains need a complete overhaul every million kilometers") tasks. Likewise, the time to perform maintenance for these tasks can vary, from several hours to several months.

For all but the most frequent maintenance tasks, we model the problem by using "instantaneous" maintenance. (For the most frequent, see Section 4.2.) What this means is that when it is time for a train to be maintained, it needs to get to a maintenance station, at which point it is replaced within its cycle by a newly maintained train. (Additionally, some maintenance stations are only open during certain times or days of the week.) Thus, if the maintenance takes several days or weeks, the actual maintenance schedule can be set almost independently of the rostering decisions. Currently, the train companies have more than enough maintenance capacity, so this will not impose any unrealistic limitations. In fact, DB already uses this system within their own plans. Instead of thinking of just a plan for each physical train, including maintenance, it is useful to consider "logical trains", those in the cyclic solution performing the scheduled tasks, and "maintenance trains", those being maintained. When a train undergoes instantaneous maintenance, one logical and maintenance train are swapped. Clearly, if the trains being swapped are specifically identified, the two views contain the same information.

Realistically, it is not practical to expect a plan for a particular train to be held precisely for millions of kilometers. Unexpected changes such as extra trains or breakdowns happen frequently enough that trains occasionally get "shuffled" to cover for each other. Railway companies keep an over-stock of trains of approximately 10% to replace others, to keep these unforeseen changes from propagating problems through the entire schedule, but when these changes are made, any long-term plans need to be recalculated.

It is more pragmatic to only preschedule more regular maintenance, certainly in intervals of up to 10,000 km, and perhaps in intervals up to 100,000 km. Anything above that happens infrequently enough that even a long extra movement to a maintenance station will not greatly affect the cost of any solution.

2.2 Variants

Many other considerations can be made. We list some of the most important, related to the variants on the input, or extra constraints.

- *Station turn around times.* Within each station, a certain amount of time is needed between routes. A train can not arrive and then instantaneously depart on a new route.

This time can vary by station, and can vary from a few minutes up to over an hour. In the cases we have seen, this can be modeled simply by modifying the end times of each train route. If instead the time varies greatly depending on whether or not the train is coupled to another at the station, the flow algorithms do not model this very well, but if the station is a busy one with many trains in it, and additional coupling time is not too large, our heuristics (Section 4) will still perform well by initially ignoring this added time.

- *Station capacities.* Train stations do not have space for unlimited trains. During the day, this is usually not a problem: if the stations have the capacity to store all of the trains at night, when very few are in use, then storing them during the day is much easier. Many of the trains will be in service, and the rest are generally well dispersed by the scheduled routes (the schedules do not send more trains into a station at once than it can hold).

  Other works, using the graph model described in Section 2.3, claim to properly observe station capacities, but this is mistaken for several reasons, which we will discuss. Instead, we will use station capacities only for extreme cases. For instance, if a station has no secure facilities to store trains overnight, we will limit overnight capacity to zero. We further discuss station capacity issues in Section 2.3.1.

- *Shunting.* Even if a station's capacity is given, the topology of tracks within the station needs to be known in order to figure out if the trains at that station can pass each other. There is a difference between 3 short platforms and one platform 3 times as long. Routing trains through stations, including shunting questions, have been proven $\mathcal{NP}$-hard in [KRZ97, KZ95], and we will not consider them here. These problems are generally considered after the rostering solution is determined, with local decisions made for each station.

- *Train length.* In [EGH$^+$01a, EGH$^+$01b], many of these problems are considered for the case when all trains are of the same length. Here, we will allow trains to have different lengths, where lower bounds are set by passenger demand and upper bounds are set by physical constraints.

- *Identical vs. different trains types.* Theoretically, any route can have a set of different train types that can satisfy it, based on factors such as the speed or braking power of a locomotive. In discussions with DB and SBB, we have found that they already plan one specific train type for each route. Thus, problems can generally be solved separately. Even if a high-speed train is capable of satisfying a regional route, and has time between runs to do it, they are managed separately. Further, within Germany, DB is broken into different regions, which can be solved (nearly) independently, due to the company structure. (Some exceptions exists for routes equally split between regions.)

  The flow model we will use allows for efficient solutions for identical train units. (The related multi-commodity flow problem is $\mathcal{NP}$-hard.) Given independent solutions for different train types within the same region, the biggest savings can come from combining empty movements of different train types, or combining an empty movement of one train type with a scheduled movement of another. (Trains which are longer than needed are less expensive than a shorter train plus an empty movement.) We will handle this heuristically.

- *Self-locomotive cars vs. separate cars and locomotives.* Some trains come in units which contain a locomotive. For instance, the Zürich S-Bahn comes in units of several cars, which include first class, second class, smoking and non-smoking areas, as well as a locomotive. These cars are considered as a single unit, and are always scheduled together. This is the simpler of the two variants.

  In cases when cars and locomotives are separate, we will solve the problem in stages: first, we will solve the train length problem for passenger cars, based on passenger demand. Next, we will solve a locomotive problem, based on the "demand" created by the passenger cars scheduled. These solutions can then be given final assignments.

  So far, in our dealings with SBB and DB, this problem has not come up. For SBB, the data we are currently considering is for self-locomoting trains. For DB, they have asked us to solve a *pure locomotive* rostering problem: they have already decided which locomotive types (and how many) they need for each route, and we have been asked to give a rostering for just the locomotives. As previously mentioned, oftentimes the railway companies already have parts of the solution they want from experience, and this makes the algorithmic job easier. We have also been told that train companies are moving towards all self-locomotive trains in the future.

- *Combined routes.* Sometimes a single train departs a station, and later splits into two (or more) directions. Also, separate trains can combine. For the problems that we consider, this will not cause any problem: when such routes are planned, the timing and cost of coupling or decoupling the trains is already fixed. These routes usually occur in the case with separate cars and locomotives. For passenger cars, the first (second) type can be well modeled by considering two trains, which depart from (arrive at) the same station at the same time, but which arrive at (depart from) different stations. Because routes are specified by only their initial and destination stations, these routes will be treated just as any other route, and we can ensure that the trip will appear as one continuous route to a passenger within one of the cars.

  For the locomotives, it is better to model the above events as 3 separate routes, one for the combined parts, and one for each of the separate parts. This will help to use locomotives efficiently, for instance in case one of the locomotives for the separate parts is powerful enough to pull the combined train. Once again, this simply entails ensuring that the problem input to the locomotive rostering algorithm is specified correctly. More complex combinations of routes can also occur (trains split and recombine with others), but the same ideas carry through for these cases too.

  We discuss a very minor problem with this model (and of piggybacking empty trains to scheduled routes in general) in Section 2.3.1. For the case of these combined rides, the problem is not significant.

- *Additional safety requirements.* It is assumed that all safety requirements are followed between scheduled routes. To ensure that safety requirements are also met between empty train movements, the full track topology is needed. This should be checked after the rostering solutions are given (or after the train length problem is solved), and is not included in the rostering process. In general, our empty movements will have a window in which they can occur. If any route from one station to the other is available safely anytime in that window, the empty route can take place. These issues will be dealt with locally, as modifications to the rostering solution.

Additionally, many different costs can be considered. We will consider two flow models, one with the usual costs that are proportional to the amount of flow (we call them linear costs), and one that in addition allows for fixed costs for arcs with nonzero flow. The first version can be solved in polynomial time, while the fixed costs one is $\mathcal{NP}$-hard, normally approached with some branch and bound techniques. Here we list different costs and, for each of them, we indicate whether it can be modeled as a linear cost or not:

- *Number of cars.* This is the most obvious cost factor, and probably the most important to minimize. Trains are very expensive, and it costs tens of thousands of dollars per year *even to maintain a train which is not used.* On the other hand, the trains are already owned by the companies, so using fewer to fulfill a schedule merely allows more routes to be scheduled, or older (or maintenance intensive) trains to be retired earlier, or fewer trains to be purchased in the future. The train company can't get a refund for trains unused, but reducing future expenditures is clearly worthwhile. This is a linear cost.
- *Cost per distance.* As trains are used, they require more maintenance, they use energy (electricity or fuel), they create wear and tear on the rails themselves, and additionally they must pay to use some rails. All of these costs are linear, save the last: if a long train uses some rail, reserving the rail for that time is not twice as expensive as it is for a train half as long. Thus, this final cost needs to be modeled as a fixed cost.
- *Crew.* Every train needs a crew (deadhead movements do not need a full crew, but at least they need a driver). After the initial cost of such a skeleton crew, further crew costs may be linear, but the initial crew cost is best modeled with a fixed cost.
- *Coupling/decoupling.* This cost is linear in the number of couplings, not in the length of the trains being coupled. These costs will not be considered during a pure train length phase of the problem, because to calculate them one needs knowledge of which routes one train will consecutively service. Within the assignment phase, railway companies prefer piggybacking empty trains onto scheduled ones (which may involve coupling) to deadhead movements, so it will be safe to heuristically favor piggyback movements during this phase.

The flow model, which we discuss next, is more concerned with the train length phase of the rostering process, while we will use heuristics in Section 4 to solve the other problems of the assignment phase.

2.3 Graph model

Let an *event* be the scheduled arrival or departure of a train at a station, that is, an $\langle s, t \rangle$ pair with station s at time t. Thus, each route is associated with two events. Notice that several routes can share an event, if they arrive at the same station at the same time.

Using events as vertices, we can model the routes of one period with a graph in an intuitive way. In Figure 1, we show such a graph. In the graph are edges to represent the scheduled rides (with lower and upper bound capacities to represent passenger demand and track/station limitations), deadheading edges between two stations to represent movements of empty trains to stations where they are needed, and edges within a station to represent trains waiting within the station. (Scheduled routes, such as scheduled shunting movements for locomotives to move passenger cars, can be represented as a standard scheduled route, where the departure and arrival stations are the same.) Station capacities can be represented by bounds on these edges. The cost of the edges can take account of the cost of train ownership, of moving the trains (which causes additional maintenance and energy usage).

Even if this single period schedule is repeated indefinitely, there is no guarantee that any solution to the above problem can be repeated each time: it may finish the period with a different number of trains in each station than it began. This problem is overcome with a

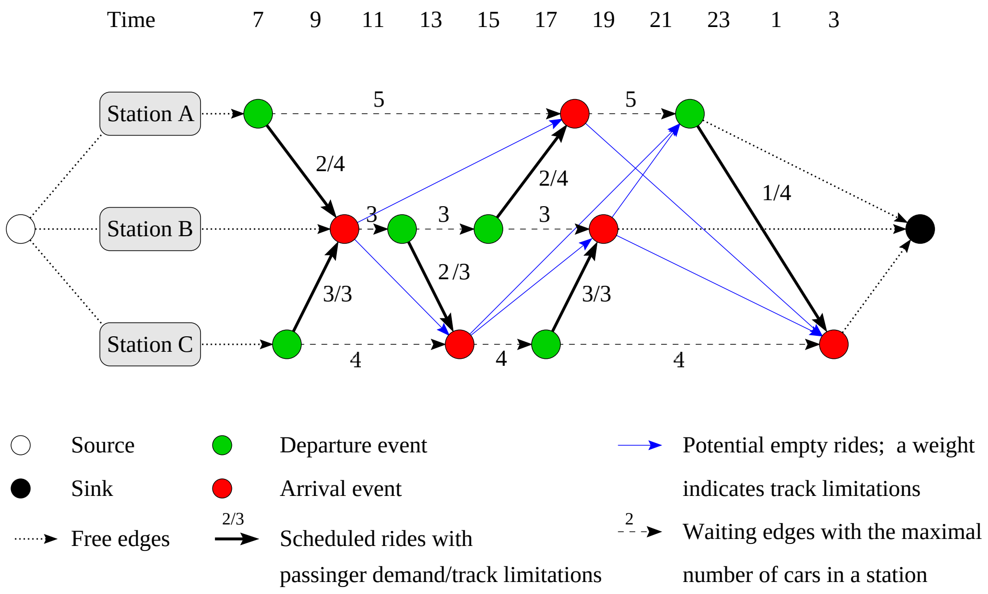

Figure 1: A route graph for one period.

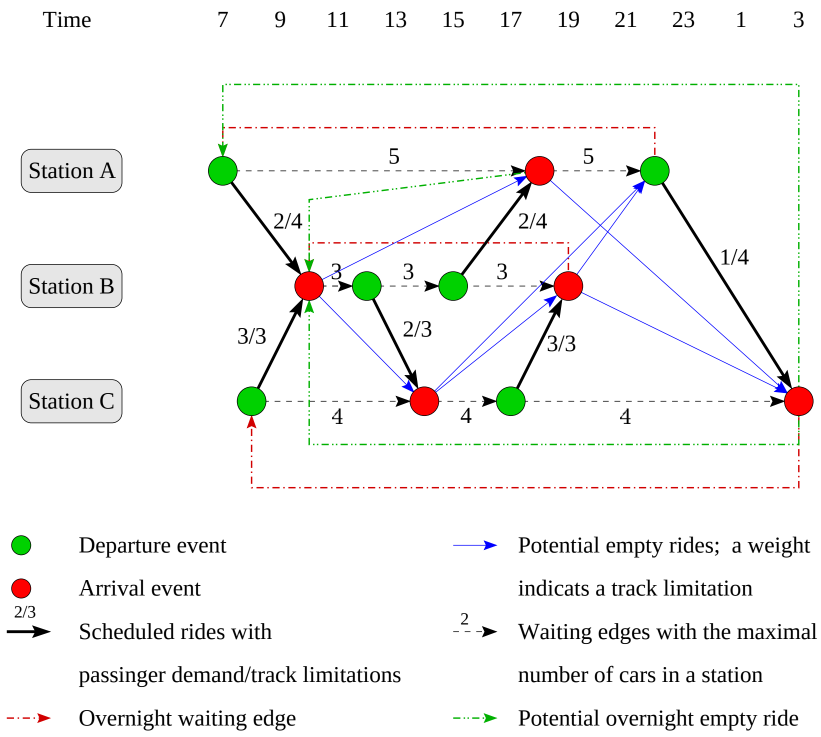

Figure 2: A route graph for many periods.

slight modification to the graph, as seen in Figure 2, which brings us to one of the standard models used for the problem. Here, we use a minimum cost circulation algorithm to get a solution which can be repeated each period: each period will begin with the same number of trains in a station as the period before. Further, if a train performs routes r and r' consecutively in one period, then in every period the train which performs route r will next perform r'. We will call this a *one period solution*. These solutions are easier to plan, and to follow, and are favored by SBB and DB, although it is possible for them to cost more than solutions that are allowed to span more than one period.

2.3.1 Model problems

There are several problems with this model, which have been overlooked in much previous work. We start with a discussion of the deadheading edges in the graph. In order to not increase the number of edges by too many, only one potential empty ride edge is drawn from each scheduled route to every other station. We draw it between the arrival of a train and the next event at the destination station which can be reached by a train departing as soon as possible, to avoid introducing extra vertices. This may allow more time for the actual movement than is needed, and the movement can take place anywhere within this window.

This causes problems if station capacity limits are being used. If the empty movement time is over-estimated, then the model abstraction places the train between stations (not at either) for the entire time. Thus, such a solution could inadvertently overflow a station's capacity. An alternate graph model would choose one specific time for each of the empty movements, and accurately depict it, adding events if needed. This type of solution would not overflow station capacities, but may fail to fully utilize capacity due to a bad placement of the potential empty movement within the possible window. By adding multiple edges, this problem can be minimized, but other problems with station capacity make any incremental improvement in accuracy not worth the increased graph size.

Another major problem with station capacity is that because routes are specified by only their extreme stations, there may be more trains at a station at any given moment, due to intermediate stops within the routes. Further, without full station topology information (and a solution to the assignment as well as the length problems), knowing a station is large enough to hold the trains scheduled to be there is not sufficient to say that cars can be routed through the station.

Considering the fact that the stations have capacity enough to house almost all of the trains during inactive periods overnight, the station capacity problem is generally considered separately, during shunting and storage problems. We will only use capacity in special cases, such as when a station must be kept empty at night, due to lack of secure storage facilities or personnel. This is an important case, and can be modeled without further topology knowledge.

The lack of intermediate station information for routes brings up one additional problem. During the assignment phase, when we attempt to piggyback empty movements onto scheduled routes rather than make deadheading movements, some scheduled routes may contain, as subroutes, the needed empty movement. Without intermediate route information, the assignment heuristics cannot find these piggyback movements. If these mid-route couplings would not delay trains too much (they may), then we can try to find this with post-processing, as it would be a relatively local switch to any rostering solution.

2.4 One period solutions

Consider a train length solution from the circulation problem in Figure 2. As with any circulation, the in and out degrees of each vertex are equal; the circulation can be decomposed into cycles. These cycles may overlap each other, or contain an edge more than once, for edges assigned weight more than one in the solution. For these "long train" edges, it is convenient to think of multiple edges, each of weight one, to distinguish among them (i.e. the first and second units of a train ride are considered as separate routes, although they travel together).

Any such cyclic decomposition can be used to assign specific trains to the routes: consider a physical train unit, which has just completed a route within the cycle. Its next action is modeled by the next action in the cycle, whether that be waiting within a station, an empty movement to another station, or immediately embarking upon its next scheduled route (which would in reality include a short wait within the station for the station turn around time).

We call such a decomposition a one period solution. This does not imply that each of the cycles needs one period to complete, they can take much longer. (They do need *at least* one period.) What is implied is that a cycle which takes x periods to complete also requires x train units. On each day, a single unit will complete the tasks for the ith day of the cycle, and will finish the day at the starting position for the $(i+1)$st day of the cycle (wrapped over the x days). As previously stated, this implies that once any train consecutively performs two routes, then in every period those two routes will both be serviced by one train.

This adds structure to the solution which may not be needed, but which is convenient to have. Multi-period assignments (or even non-periodic assignments) can be made for the same train length solution. To better conceptualize a two period solution, imagine that the graph in Figure 2 is doubled, to explicitly show two periods. Intuitively, the graph still models the same (infinite) sequence of train routes, and thus should still have the same (optimal cost) circulation, only doubled because twice as long a time period is represented. This is more formally proved in Section 3.2. However, two routes in the first half of the graph which are serviced by one train may be serviced by separate trains in the second half. In this case, train behavior during alternate periods would look the same.

This two (or more) period solution allows greater flexibility. For instance, it may allow for a change of the number of trains at a station on alternate periods, while the one period solution cannot allow such changes. Although this flexibility will not allow for better solutions for the pure train length problem (which does not include maintenance), it can allow better cost solutions for the more general rostering problem. We will more formally discuss this in Section 3.2, and explain our approaches to the problem in Section 4. For more background, see [EGH$^+$01a, EGH$^+$01b, Gan01].

2.5 Past work

The flow approach to this problem is an intuitive one, and it is one which is mentioned as early as 1954 [DF54], and is standard enough to be included in textbooks [AMO93] and surveys [DDSS95], yet it is also still being studied in more recent works [BHR99, Sch93]. The standard claim is that once the problem has been modeled by such a graph, standard circulation algorithms will provide useful solutions to the problem.

We feel that there is good reason why such solutions have not yet been completely accepted, and incorporated into real railway systems, and these reasons go beyond the standard time delays of putting theory into practice. Instead, we think that those proposing the model have oftentimes ignored necessary problem constraints, most importantly that of maintenance. Too many limitations have been overlooked, such as the station capacity problems mentioned above. Although the details above may look algorithmically "dirty", glossing over some of these aspects of the problem may lead to flow based solutions which are nearly unrelated to valid rolling stock rostering solutions, with completely different costs.

Our goal is to build upon past work, and to determine when these techniques do provide good solutions. If these conditions hold in real data, then we want to apply this standard flow based approach to the problem. In order to discover what conditions are needed, we feel it is first necessary to better understand the structure of the solutions needed for the rolling stock problem, and then to find what extra conditions hold (or which are lacking) for single period solutions. We discuss this solution structure in Section 3.2.

3 Theoretical results

Here, we discuss some of abstract properties of the problem.

3.1 Hardness

It is shown in [EGH$^+$01a, EGH$^+$01b] that there is a constant such that the rolling stock rostering problem is $\mathcal{NP}$-hard to approximate within that constant. This holds even for one period solutions with highly simplified maintenance constraints, with all train lengths set to 1, and the most basic cost measures possible (the cost is equal to the number of trains needed).

In [Gan01], a variant of a proof from [HS89] 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. The proof reduces Steiner tree problems to a periodic train schedule with fixed costs.

On the positive side, it seems that the number of empty movements needed for real life train schedules is not so large that these fixed costs should dominate. In our experiments, we are finding that only a small percentage of the train movements are empty. It is safe to assume that empty movements are less expensive than scheduled movements, because a smaller crew is needed. If fixed costs of the empty movements are ignored, and our algorithm produces an answer with only a small number of empty movements and not too many couplings or decouplings, it is safe to say that it must be nearly optimal even if the fixed costs were included.

Further, fixed costs are *not* arbitrary. We have been told that in order to move a train from one station to another, piggyback movements are preferred over deadheading. This implies that coupling costs are not too large (or an extra locomotive driver for the deadhead movement is very expensive, but we also know that empty movements are cheaper than scheduled movements).

3.2 Solution structure

Periodicity of schedules has been studied in the past, but with a definition that is different: a schedule or its implementation with vehicles is required to be periodic, with some prescribed period. For the simple case of merely 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 [Orl82] uses a periodic version of Dilworth's Theorem [Dil50], whereas Gertsbakh and Gurevich [GG77] use the concept of a "deficit function". Later, Serafini and Ukovich [SU89] propose quite a general framework for periodic scheduling problems that also implies the above result.

In [EGH$^+$01a, EGH$^+$01b], many issues of periodicity are discussed. These works concentrate on periodicity solutions for simpler versions of full rolling stock rostering problem (all train lengths are 1, and maintenance is only needed "occasionally"). Their results do not hold for the more complex rolling stock instances we consider here, but instead we will consider one period solutions to the (complex) train length problem. We prove that for the train length problem on an infinite, periodic set of routes, the best one period solution is has an optimal cost per period. The costs can be any of the linear costs from Section 2.2. (They will not include the coupling/decoupling charges, which apply to the rostering problem rather than the train length problem.)

First, we note that within the infinite graph model (which would be pictured by repeating the center portion of Figure 1 forever, connected to itself in a chain), because all edges go forward in time, and flow is not created or destroyed anywhere, the same total flow must cross over each point in time. Any integer flow problem has an optimal integer flow solution, and there are a finite number of edges crossing each point in time, so for any point in time there are only a finite possible number of edge weights which can be assigned for that point in time for any finite integer flow.

Considering this, we see that eventually in any optimal infinite integer flow solution, the flow across some boundary between periods must repeat. Further, we claim that there must be two such points with the minimum cost per period between them. We need to be a bit careful, due to the fact that an infinite solution can contain infinite portions which are suboptimal, as long as "it all averages out". Our goal is not to deeply explore measure theory and the counter-intuitive truths of infinities, but clearly this statement holds "in the limit", that is, there must be some pair with the cost per period arbitrarily close to the optimal cost.

From this, we see that there is a p period solution which is arbitrarily close to the optimal cost per period. We will assume that it has the optimal cost per period for the rest of this proof. (The proof holds without this assumption, but the language becomes more burdensome.) We want to compare this solution to one which represents repeating a single period train length (flow) solution through p days.

Consider folding edges of the p period flow solution into a multigraph representing a single period. Any vertices (events) from the p day solution which correspond to the same events, in different periods, are combined into one vertex. As we are dealing with only linear costs, all costs for a heavily weighted edges can be broken into separate edges, each with equal portions of the cost. Because the original solution can be decomposed into edge disjoint cycles (Dilworth's Theorem [Dil50]), so can the new, folded graph.

Similarly, consider p copies of the single period solution, in a one-period multigraph. The edges of the first graph must weigh less than the edges of the second, by p times the periodic savings of the p period solution to the one day solution.

Finally, let G be a multigraph representing the p period solution minus the one period solution. If an edge occurs more often in the p period solution, it will be in G that number of times. If an edge occurs more often in the one period solution, it occurs in G, *reversed and with negated cost*, that many times. Just as both of the individual multigraphs could be decomposed into edge disjoint cycles, so can G.

The total cost of G must be negative, and thus the cost of some cycle in any decomposition must also be negative. Given the one period solution, "adding" this cycle to it will remove some edges from the solution (those which have been negated), and replace them with some other edges which occur somewhere in the p day solution. This will give a one day solution which has a lower cost than the supposedly optimal one, a contradiction.

4 Heuristics

So far, we have spent most of our space with discussion of the train length problem, while devoting little to the assignment problem which is additionally needed for a complete rostering solution. The circulation algorithm above merely finds the lengths of the different routes, and determines which deadheading movements are needed, with no consideration of maintenance. A train assignment is still needed to assign specific trains to these routes, and it must take train specific constraints into account.

There are advantages to separating the rostering problem into two phases. Besides the simplicity of smaller problems, we also have a good understanding of the first phase; circulation algorithms are well studied. Further, using such a first step will allow us to find bounds on how good a particular solution to the whole problem is. Solving the problem while ignoring maintenance constraints will provide a lower bound to any solution with maintenance, and if we find solutions near this value which include maintenance we are well satisfied.

Unfortunately, as discussed in Section 3.1, the problems including maintenance are $\mathcal{NP}$-hard. In [EGH$^+$01a, EGH$^+$01b], some approximation bounds are given for the problem, with simplified maintenance constraints, when all trains have length one. In that version, all trains need "eventual" maintenance, that is, they just need to eventually pass through a maintenance station. Here, we want more strict, realistic maintenance schedules, but instead of considering theoretical worst case input, we consider real life data.

From discussions with DB and SBB, we are convinced that although maintenance theoretically hard to schedule, in practice it is not nearly so bad. Maintenance stations tend to be at, or near, commonly used stations. Within DB data, many of the station endpoints themselves are capable of performing the most frequent maintenance. Less frequent major overhauls are less important to plan far in advance, due to uncertainty in planning for distant time frames and the fact that they occur so infrequently. Occasional inefficiencies will not greatly harm the overall cost of a solution.

4.1 Train assignments

We consider several ways of creating specific train rostering assignments from the train length solution. The simplest is to just decompose the solution into as few cycles as possible. Each vertex in the graph has the same in degree as out degree (considering weighted edges as multi-edges). If the length solution gives a connected graph, all edges can be combined into a single Eulerian cycle. Thus, all trains will follow the same route in a multi-period cycle, but staggered by which of they periods within the cycle they begin. Such a solution has an added benefit: all trains in service will cover the same mileage, thus evenly spreading the workload among them. If a Eulerian cycle is not present, we use the same technique to decompose the solution into as few cycles as possible and treat each separately. (The proof that this decomposition works is due to extensions of Dilworth's Theorem [Dil50].)

Once this is done, maintenance still must be considered. First, we note which stations are capable of maintenance (or those which are close). For regular maintenance, there is a buffer of well over 1000 km between recommended and required maintenance. Thus, we can simply look within the cycle of a train, and find if any routes end in (or near) a maintenance station after the recommended distance.

We expect that this simple approach will perform very well, but experiments are still underway. If successful, this approach will provide great flexibility: the current state (distance to maintenance) of each train can be considered separately, and maintenance schedules can be made up individually. Thus, we can have a one period solution to both the train length and "logical" train assignment (without maintenance) problems, but the maintenance can take its own form, different for each train along a cycle. If a more "steady-state" solution is desired, the times for maintenance can just be added within the cyclic solution, which bring the same number of trains, period after period, to any given maintenance site.

A second approach for creating cycles is to attempt to do the same as above, but do it separately by train length. Although this approach may not result in as clean a cyclic decomposition, it will decrease the couplings and decouplings needed with stations. Thus, if possible, two long train routes can be consecutively serviced by a single train. We can find long chains of consecutive long train runs, and only have coupling and decoupling at the ends of these chains. While this may result in more cycles (cycles of long trains will never be split into short ones), saved shunting costs may easily pay for whatever difficulties this adds to keeping trains maintained.

While it may seem that we are pinning our hopes on the problem instances solving our problems for us, there are powerful steps we can take to remedy any situation when this does not occur. These go beyond algorithmic solutions (many search techniques can be used to explore the solution space of cycles from any given train length solution). Instead, we find that in real life, we can change the rules of the game.

We are already considering instantaneous maintenance, as introduced in Section 2.1, where a train must merely end a route (or an empty movement) at a maintenance station, at which time a maintained train is swapped with the logical train in the route cycle. This not only realistic, it is done in practice. Of course, there is little difference between making an empty movement to, vs. from, a service station. Thus, a maintained train can use an empty movement get to a nearby station within the cycle of a train which needs maintenance, and they can swap positions there. While this may cause some extra mileage to the trains, extra 50-100 km out of 10000 seems a small price to pay for "anytime, anywhere" maintenance.

Sometimes, railways may want to consider purely operational cycle properties. For instance, perhaps it is preferable to keep a given train on a small set of similar routes for as long as possible, rather than sending it through a cycle of all routes (if a Eulerian tour is present). So far, experiments with this look very promising, though we expect that it may make following maintenance somewhat harder.

We should note that many times, trains are grouped and planned regionally. When this happens, even if very long cycles exist, the trains will always stay within the region, so they should not stray too far from any base. Also, crews need not stay with the same train for their whole day. For crew planning (which we do not consider here), crew members usually want to finish each day at the same place they begin, a restraint we do not want to impose upon the trains. Thus, crews may switch trains during the day, but this should be minimized, as a late crew may propagate delays through the system.

4.2 Refueling and minor cleanings

The above techniques appear to be sufficient for maintenance which does not occur too frequently. For the most frequent maintenance jobs, they are not as attractive: wasting even 50 km out of 500 does not seem very efficient, and if a single logical train is swapped for a maintained train several times per day, the intuitive usefulness of the logical trains diminishes.

The most frequent maintenance type procedures are minor cleanings, and fueling for diesel locomotives. The minor cleanings cause no problems: no special facilities are needed, just some staff at a station. This might entail something as simple as collecting garbage from the train between different routes. Because any facility will do, and the time for the maintenance is so short, it can even be incorporated within the station turn around times.

Much more crucial is the fueling of diesel trains. Clearly, this is much more rigid constraint, as the train obviously stops when it runs out of fuel. Luckily, most trains are now electric, but diesel trains will still be used for years to come. We are currently running experiments to find how difficult this fueling constraint is, but work is still in a preliminary, data collection stage.

4.3 Combining solutions

The number of trains at any station at the start and end of a period will be the same for a one period rostering. Throughout the period, the events will look identical to the previous period. This is useful when considering practical matters within a station. For instance, it makes it easier to consistently put the same route within a period on the same track: if the train can be routed to a track during one period, it can be routed there every period.

It also makes "incremental" changes easier. For instance, suppose that within a week long period schedule, a few additional train routes are being considered for a Thursday. Running any program from scratch might choose a globally change, which might force huge amounts of extra work: new routings within stations and different route sequences for each train may be needed, and some of these changes may be visible to customers. For instance, train lengths may change for routes during other days of the week. In general, changes should be transparent to the customers. If another solution, of nearly the same cost, changes only some trains during Thursday, it may be preferred.

To do this, one can consider the flow algorithm, and the placement of the trains at the start and end of Thursday. Using a graph similar to that in Figure 1, with the source and sink edges replaced by these trains, if a flow solution exists, there will be a train length solution which uses the same number of trains as before, but no changes will be visible outside of Thursday. For potential added routes, or one day extra trains, this is a nice feature.

One can also get a very quick estimate of the total trains needed for the length problem. Consider that within a standard weekly cycle, Friday tends to be the busiest day. Using a graph such as the one in Figure 1, with just Friday routes, we can get a lower bound on the number of trains needed in the system, because at least this many are needed for just Friday. Perhaps even more are needed: this figure allows trains to begin and end the day wherever is easiest, without considering previous and following days. Using the graph in Figure 2, we can expect an upper bound on the number of trains needed. This need not hold, but should for real data: if Friday has the most routes, and they are nearly a superset of all other days, then having enough trains to satisfy a repeating Friday will be sufficient for an entire week. In reality, we might expect the number of trains needed for the week to fall somewhere between these two, which initial experiments confirm. By checking only one day, the system can run much faster, and still provide good estimates on the number of trains needed.

We can extend this idea to try to provide even more regularity within the week. Again, consider that Friday is the busiest train day. If the flow algorithm is run for Friday alone (we need to experiment to decide which of the two models to use), we can determine how many trains we hope to start a Friday with. Next, if Monday through Thursday are identical (this is not realistic overall, but for some small regions it may come close to being true), we can run a single day for them, fixing the number of trains at the end to be the number needed for

Friday. Finally, we can run one more flow algorithm to connect the end of Friday, through the weekend, to Monday. In this way, for the identical days, the routing and track assignments within a station can look identical to the train customers. Friday, and the weekend, will look different, but clearly they must because different numbers of routes occur on those days.

In general, this is simply an argument that it is fairly easy to "glue" separate flow solutions together, once boundary conditions (the number of trains at each station at the start and end of the times considered) match. It may be useful to even consider all distinct schedule days within a year (there should not be too many of them), and to look at them individually before proceeding to create more complex solutions.

Clearly, the heuristics for finding rosterings, with maintenance, must take place after these combinations: the maintenance constraints are global, rather than local within a sub-solution.

5 Conclusions

As mentioned above, some of the cost functions cannot be modeled with our flow as a minimum flow circulation problem. Indeed, these versions of our problem are $\mathcal{NP}$-hard (see Section 3.1). However, the flow model approach is still valid in many practical cases and it provides a "dual" approach of classical branch and bound heuristics. The latter guarantee the optimality of the solution and, on real data, often run quickly. The flow model, instead, guarantees a fast feasible solution and, on real data, gives near optimal solutions.

Interestingly, the two approaches can be combined together into a heuristic which is faster than the "pure" branch and bound. The idea is to first obtain a solution from the flow model and then use this to prune the search tree used by the branch and bound algorithm. We first observe that any solution from the flow model with linear costs remains feasible for the problem with fixed costs; it just might be the case that this solution is not the optimal one. Consider the set of empty edges (i.e. the set of empty movements) used by this solution. Because of its feasibility (also for the fixed costs version), any branch and bound algorithm will (at least) also find such a solution. However, we can allow some flexibility to this initial set of empty edges by adding back to them those empty edges that in the original graph that are not too expensive. The idea is to have a search space that contains better solution than the one given by the flow model, but still much smaller than the original entire search space with all of the empty edges. The reason why do not add back high-cost empty edges is twofold: on one hand there cannot be many of them in any optimal solution; on the other, a train that uses one of these edges is not available for quite a long time, thus making the use of such and edge not productive.

There are still many rounds of discussions needed between researchers and railway companies. Because so many of the problems studied interact with each other, no "self-contained" system can ever hope to be optimal. For instance, we are considering ways in which a rostering program might make minor schedule change suggestions, such as shifting a few trains by a few minutes, should such changes make it possible for a more efficient rostering.

In general there are more problems to be solved than there is manpower to solve them. Considering the scale of the railway company costs, even small improvements can save vast amounts of money. This is motivating them to open their goals and databases to researchers.

References

[AMO93] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows - theory, algorithms, and applications*. Prentice Hall, 1993.

[BHR99] 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.

[DDSS95] 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.

[DF54] G. Dantzig and D. Fulkerson. Minimizing the number of tankers to meet a fixed schedule. *Naval Research Logistics Quarterly*, 1:217–222, 1954.

[Dil50] R. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51:161–166, 1950.

[EGH+01a] T. Erlebach, M. Gantenbein, D. Hürlimann, G. Neyer, A. Pagourtzis, P. Penna, K. Schlude, K. Steinhfel, D. Taylor, and P. Widmayer. On the complexity of train assignment problems. In *ISAAC: International Symposium on Algorithms and Computation, LNCS*. Springer-Verlag, 2001.

[EGH+01b] T. Erlebach, M. Gantenbein, D. Hürlimann, G. Neyer, A. Pagourtzis, P. Penna, K. Schlude, K. Steinhfel, D. Taylor, and P. Widmayer. On the complexity of train assignment problems. Technical Report 363, Department of Computer Science, ETHZ Zürich, 2001. Available at `http://www.inf.ethz.ch/`.

[Gan01] M. Gantenbein. The train length problem. Diploma Thesis, Department of Computer Science, ETH Zürich, 2001.

[GG77] I. Gerthsbak and Y. Gurevich. Constructing an optimal fleet for a transportation schedule. *Transportation Science*, 11:20–36, 1977.

[HS89] S. Hochbaum and A. Segev. Analysis of a flow problem with fixed charges. *Networks*, 19:291–312, 1989.

[KRZ97] L.G. Kroon, H.E. Romeijn, and P.J. Zwaneveld. Routing trains through railway stations: complexity issues. *European Journal of Operational Research*, 98:485–498, 1997.

[KZ95] L.G. Kroon and P.J. Zwaneveld. Routing trains through railway stations including shunting decisions. In *Operations Research Proceedings*, pages 438–444. Springer Verlag, 1995.

[Orl82] 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.

[Sch93] A. Schrijver. Minimum circulation of railway stock. *CWI Quarterly*, 6(3):205–217, 1993.

[SU89] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM J. Discrete Mathematics*, 2(4):550–581, 1989.