On the Complexity of Train Assignment Problems*

Thomas Erlebach[1], Martin Gantenbein[2], Daniel Hürlimann[3], Gabriele Neyer[4], Aris Pagourtzis[5], Paolo Penna[2], Konrad Schlude[2], Kathleen Steinhöfel[6], David Scot Taylor[2], and Peter Widmayer[2]

[1] Computer Engineering and Networks Laboratory, ETH Zentrum, CH-8092 Zürich, Switzerland
erlebach@tik.ee.ethz.ch

[2] Institute for Theoretical Computer Science, ETH Zentrum, CH-8092 Zürich, Switzerland
{lastname}@inf.ethz.ch

[3] Institute of Transportation, Traffic, Highway- and Railway-Engineering (IVT), ETH Hönggerberg, CH-8093 Zürich, Switzerland
huerlimann@ivt.baug.ethz.ch

[4] SMA and Partners Ltd., Transportation Engineers, Planners and Economists, Gubelstrasse 28, CH-8050 Zürich, Switzerland
GabrieleNeyer@gmx.ch

[5] Department of Computer Science, University of Liverpool, United Kingdom
aris@csc.liv.ac.uk

[6] GMD - FIRST, Berlin, Germany
Kathleen.Steinhoefel@gmd.de

Abstract. We consider a problem faced by train companies: How can trains be assigned to satisfy scheduled routes in a cost efficient way? Currently, many railway companies create solutions by hand, a time-consuming task which is too slow for interaction with the schedule creators. Further, it is difficult to measure how efficient the manual solutions are. We consider several variants of the problem. For some, we give efficient methods to solve them optimally, while for others, we prove hardness results and propose approximation algorithms.

1 Introduction

We consider the problem of assigning trains to the routes of a railway network so as to implement a given schedule and to minimize the associated cost, subject to various constraints. This problem is sometimes called *train assignment*, *train rostering*, *vehicle scheduling* or *rolling stock rostering*, and currently, it is commonly done by hand. For instance, to modify train schedules from one year to

* 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).

the next within Switzerland, the Swiss Federal Railways SBB uses several man years of labor.

With today's powerful computers, the train assignment problem should lend itself nicely to automatic solutions. It has the additional benefit that it can take effect immediately: no customer acceptance of a new schedule is needed. Furthermore, a useful system need not be perfect: any tool that proposes an initial assignment and gives an interactive indication of how easy or difficult it is to make modifications will be useful. The final schedules and train assignments may still require human expertise.

We explore how different constraints change the problem from versions with efficient, optimal solutions, to versions which are APX-hard. Among the constraints we consider, we focus on scheduling the maintenance of trains and on allowing or disallowing movements of empty, non-scheduled trains (deadheading). For the APX-hard problem versions, we propose approximation algorithms.

1.1 The Basic Model

As input, we are given a set of *train routes*: each train route is specified by a *departure time/station* and an *arrival time/station*. The routes are *periodic*, and for the purpose of this paper, we will assume a daily period: *each route in the input runs every day*. Naturally, our results do not depend on the interpretation of the periods, and hence they also apply to other time frames, such as weekly schedules. The goal is to assign trains to perform the routes in a cost effective way, subject to constraints.

In Figure 1.a, we show a graphical representation of a two-routes two-stations

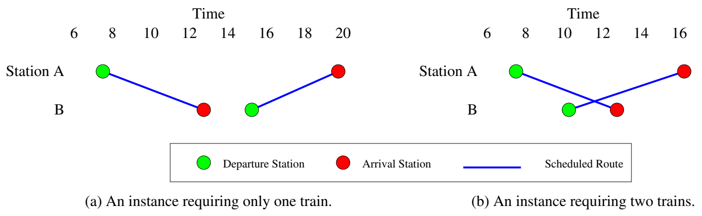

Fig. 1. Two simple train schedules

instance: the x-axis represents the time and the y-axis represents the stations; an edge between two points represents a route. For this example, one train is sufficient to cover both routes. A train t first begins at station A and travels from A to B to cover the first route. Once t arrives in B, it can wait and then cover the second route. At this point, the train is back at station A and is ready for covering the route from A to B of the next day. So, the train repeats the same cycle every day.

The reason one train can cover both the routes is that the arrival time of the first route precedes the departure time of the second one. Something different happens in the example of Figure 1.b: train t arrives at B too late to perform the second route. So, we need another train t' at station B at the beginning of the day. On the other hand, at the end of the day t is at station B and it can be used the next day for the second route. Similarly, t' is now at A and it can be used for the first route on the second day. Hence, both trains come back to their original position (i.e. at the same station at the same time) after *two days*.

In both examples, we can represent the train assignment as a cycle followed by the train(s): connect every arrival of one route with the departure of the other one (these edges represent waits within a station). If the arrival endpoint precedes the departure we have a wait within the same day; otherwise the edge represents an *overnight wait*. Then, the *length* of a cycle, measured in days, is defined as the sum of the waiting times between consecutive routes and the traveling times of all routes on the cycle (notice that every cycle takes at least one day). In general, it is possible to have cycles of several days and several routes (see the example in Figure 2).

There is a precise relationship between the cycle length and the number of trains: if a cycle takes k days, then k different trains are needed to serve the routes in that cycle within the same day. We can therefore define the following optimization problem:

Basic Rolling Stock Rostering (rsr)
Instance: A set of stations $S = \{s_1, s_2, ..., s_m\}$, and daily train routes, $R = \{r_1, r_2, ..., r_n\}$. Each route r_i consists of a departure event (dsr_i, dtr_i) and an arrival event (asr_i, atr_i), where dsr_i and asr_i represent departure and arrival stations of route r_i, and dtr_i and atr_i represent departure and arrival times.
Solution: A collection of ordered sets of routes. Each ordered set represents a cycle to be followed by at least one train: a route r_i precedes a route r_j if r_i and r_j are serviced consecutively by the same train; this is possible only if $asr_i = dsr_j$. Each route must occur in exactly[1] one of the ordered sets. We also call these sets cycles.
Cost: The number of trains needed, that is, the sum over all the cycles of the length (in number of days) of each cycle.

Note that an instance of rsr has a solution if and only if the number of arrival events equals the number of departure events at each station.

1.2 Model Variations and Assumptions

In addition to the basic model, we consider variants in which *empty movements* (deadheading) and/or *maintenance* are allowed or needed:

Empty movements allowed. For every pair of stations, we are given in the input the time for an empty, unscheduled train movement, from one station to the other one. The cycles may contain some of these empty movements.

[1] This imposes some restrictions on which kind of solutions we allow. In Section 2 we discuss this issue in detail.

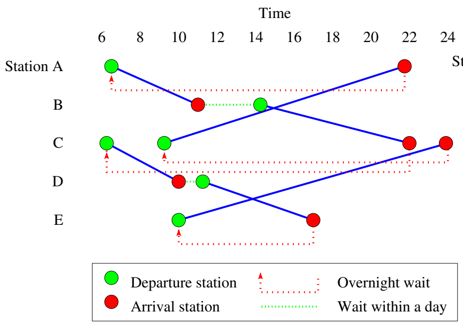

Fig. 2. A cycle of four days

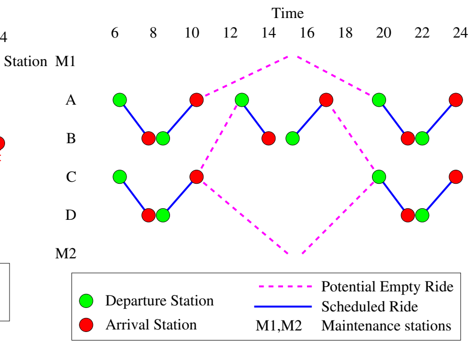

Fig. 3. An instance with empty movements and maintenance stations

Maintenance required. In the input, a (nonempty) subset of the stations is designated as *maintenance stations*. In order to be maintained, every train must eventually (periodically) pass through some maintenance station. So, every output cycle must contain a maintenance station.

Variants of RSR in which we allow empty movements, or require maintenance, or both, are denoted by RSR-E, RSR-M, and RSR-ME, respectively. For all those problems, we can further consider different possible costs to minimize. Further, we expect that an empty train movement is more expensive than simply waiting within a station, even if they take the same amount of time, due to added depreciation for track and train wear and repairs, fuel and labor costs, etc.

In defining the optimization problems we are implicitly making some assumptions. We discuss them in more detail in the sequel.

(Implicit) Assumptions. In this work, we only consider problems where all trains are identical, that is, any train can be used for any route. Two routes with the same departure and arrival stations may need different amounts of time, due to different paths taken. The latter is of no concern to us, since we do not take into account the intermediate stops between the departure and the arrival station; they need not even be specified in the input. We also address the case of routes which take more than one day: for these, there will sometimes be two (or more) trains on the same route at the same time, having started on different days.

We assume that maintenance is performed *instantaneously* as trains pass through the maintenance stations. This assumption is more realistic than it may seem: for instance, SBB keeps an inventory of about 10% extra trains to replace others in need of repair. By rotating these trains in and out of active duty, we can simplify maintenance scheduling, replacing an unmaintained train by a maintained one, once they are at the same station.

Trains do not need to pause between routes: we assume that if one route arrives at a station by the time another route leaves from that station, then one train can service both of those routes. Commonly, several minutes are needed between routes to prepare the train for the second route. For the problems we consider, we can 'pad' all departure times by several minutes, and then ignore the need for preparation time, which will make the assumption true.

Finally, we are imposing a particular structure to the solutions, due to the fact that every route occurs in exactly one cycle. As we will see (Section 2) this, in almost all cases, does not affect optimality (w.r.t. more general solutions) and it allows for train assignments that are simpler to understand.

1.3 Related Previous Work

The simplest version of rolling stock rostering, where we only want to minimize the number of trains needed to run a given schedule, is known as the minimum fleet size problem [2]. Dantzig and Fulkerson [7] propose the first solution that models the problem as a minimum cost circulation problem. A number of survey articles [8,4] discuss this simple problem and more complex variations. Because the realistic problem variants have quite a few different objectives and a lot of constraints, the only resort is to engineer a heuristic solution. To this end, a wealth of heuristic approaches have been tried, from branch and bound, branch and cut, linear programming and relaxation, to simulated annealing, to name but a few [14,3,5,13,10,11]. Experiments show that in many of these cases, the obtained solutions for random data or even for real inputs come close to the optimum and sometimes even reach it. In an effort to come up with a guarantee for the quality of a solution, we are trying to understand the inherent approximation complexity of the problem; no such study has been reported in the literature thus far. In the process of our study, we also improve the runtime for the simplest problem version, RSR with no extra constraints [8,2].

1.4 Our Contribution

In Section 2, we show that our definition of the problem(s) imposes added structure on the solutions: the way two routes are combined within a cycle is the same every day. Although this is what has been done in practice so far, up to our knowledge, the optimality of such solutions (w.r.t. more general ones) has never been investigated. We prove that, for all but one (RSR-ME) of our problems, this optimality holds, and show why this is not the case for RSR-ME.

In Section 3, we present an $O(n \log n)$-time algorithm for the basic rostering problem without maintenance, thus improving the running time of existing solutions for this version.

We consider maintenance in Section 4. First, we show that (even with our simplifications) both RSR-M and RSR-ME are APX-hard, that is, there exists a constant $r > 1$ for which even approximating the problem within a factor r is NP-hard. Then, in Section 4.2 we look at approximation algorithms and we show that RSR-M and RSR-ME have a polynomial-time 2- and 5-approximation algorithm, respectively. Finally, we show that the algorithms perform provably better if some additional hypotheses on the input hold.

Due to space limitations, most of the proofs are omitted and can be found in the full version of this paper [9].

2 Periodicity in the Solutions

Here we discuss the structure of our solution. We study solutions which look the same each day: if on day one, one train consecutively services routes r and r', then whichever train services route r on any day will next service r'. We will call these *one day assignments*. While it may seem obvious that a periodic daily schedule can have an optimum train assignment (w.r.t. number of trains) which looks the same each day, this is not necessarily the case. We prove that one day assignments give best possible solutions for RSR, RSR-M, and RSR-E. For RSR-ME, however, we now give an example where any one day assignment uses more trains than a solution without this restriction.

Consider the RSR-ME example in Figure 3. In any one day assignment, at least two trains are needed, because two train routes are simultaneously scheduled. Clearly, just after the first 4 or last 4 routes, there will be a train at A and C. With only two trains, the only way to maintain the train at A without missing a scheduled route is to make an empty movement from A to M1 at the same time as an empty movement from C to A, and then move both trains back after the two mid-day routes. (We can assume that all empty movements not shown in Figure 3 are too lengthy to help.) Similarly, to service the train at C, it can make a mid-day unscheduled movement to M2, while the train at A services the two mid-day routes. By this argument, in a one day assignment, only one of the two trains can be maintained, and so 3 trains are needed. A two day assignment does not have this problem: we can alternate between maintaining the two trains every other day, as mentioned above, and satisfy the routes with just two trains.

The following result clarifies the relationship between one day assignments and more general forms of solutions (multiple day assignments). Its proof is based on several non-trivial results about the periodicity of general solutions and the cycle structure of one day assignments (see [9]).

Theorem 1. *For* RSR, RSR-E, *and* RSR-M, *considering only the cost of train ownership (and extra costs for empty movements in* RSR-E*), the best one day assignment is optimal for any solution.*

Theorem 1 tells us that our one day assignment output restriction will not increase our optimal solution costs for RSR, RSR-E, and RSR-M.

By employing aperiodic solutions that decrease the frequency of maintenance over time, the average daily cost of an RSR-ME solution can be made arbitrarily close to that of a solution without any maintenance. Therefore, it is meaningful to consider RSR-ME solutions restricted to one-day assignments, as we do.

3 Fast Basic Rostering

We return to the simplest problem version, RSR. This problem is sometimes called the minimum fleet size problem [2], and polynomial-time solutions are known based on minimum cost bipartite matching, or flow problems. Here, each route is modeled by 2 vertices, for the arrival and departure events of the route.

An arrival vertex is connected to a departure vertex if they represent the same station, and the cost of the edge is set to the time between the arrival and the departure. A minimum perfect matching then minimizes the total waiting time of trains, because the time spent by trains performing scheduled routes is fixed. This minimizes the total number of trains used by the system.

First we notice that this basic problem can be solved more efficiently, without using the machinery of the minimum perfect bipartite matching algorithm. Our main task is to calculate, for each station, the number of trains at the start-of-day. We can then just make an arbitrary train assignment for one day which covers all scheduled routes, and this will be an optimal solution by Theorem 1.

The calculation starts by creating a list of all routes into or out of each station s. Then, we order all arrivals and departures within each station s. For each station s, we linearly (by time) search through all arrivals and departures from that station, and calculate the minimum number of trains such that the station begins each day with enough trains so that it will never have a negative number due to departures throughout the day. Finally, for any route out of a station, we pick any train that is currently in the station and assign it to that route. All steps, except for sorting, take linear time; altogether we get:

Theorem 2. *The* RSR *problem can be solved in $O(n \log n)$ time.*

The minimum cost perfect bipartite matching approach can be generalized to RSR-E by adding edges corresponding to all possible empty train movements to the bipartite graph, leading to a polynomial-time solution [2,7].

4 Rostering with Maintenance

We show that whether or not empty train movements are allowed, trying to minimize costs is hard once maintenance is needed. First, we prove that RSR-M and RSR-ME are APX-hard, thus implying that even approximating the problem within some constant factor $r > 1$ is NP-hard. Then, we present a 2-approximation algorithm for RSR-M and a 5-approximation algorithm for RSR-ME.

4.1 Hardness

We present an approximation preserving reduction from the minimum vertex cover problem on cubic graphs (i.e., graphs with maximum degree 3) to RSR-M. Since this restriction of minimum vertex cover is APX-hard [12,1], our reduction implies the same hardness result for RSR-M.

For an undirected graph $G = (V, E)$, a set $K \subseteq V$ is called a vertex cover if it contains at least one endpoint of every edge in E. Let thus $G = (V, E)$ be an undirected graph with maximum degree 3. We set $n = |V|$ and $m = |E|$. The reduction works as follows:

- We create a single maintenance station s_M and a station s_j for each vertex $v_j \in V$.
- For every edge $e \in E$, with $e = \{v_i, v_j\}$, we create an *edge cycle* composed of two routes: one going from station s_i to station s_j and the other one going back from s_j to s_i (we will specify their arrival and departure time in the sequel).
- We add a *maintenance cycle* consisting of routes from s_M to s_1, from s_1 to s_2, from s_2 to s_3, ..., from s_{n-1} to s_n and from s_n to s_M.

Then, we want to assign departure and arrival time to the routes on each of these cycles so that the following properties are fulfilled:

- The optimal solution *without maintenance* simply consists of the union of the edge cycles and of the maintenance cycle. We call such a solution *trivial solution* and we denote its cost by C_{triv}.
- If we instead require maintenance, then there exists a solution of cost $C_{triv}+k$ if and only if G has a vertex cover of size k. Moreover, we force any feasible solution to consist of a single cycle by having only one route arriving at s_M and one route departing from s_M.

The main idea underlying the construction is the following. In order to transform the solution without maintenance into a solution with maintenance, we have to change the solution at some stations so that we can "merge" the edge cycles and the maintenance cycle into a single one. Moreover, by using a solution different from the trivial one at a station s_j, *all* the cycles passing through s_j can be combined into one cycle at an extra cost of one day (viewed differently, this corresponds to employing an extra train). Notice that the edge cycles passing through s_j correspond to those edges of G that have v_j as one endpoint. Intuitively, the station s_j "covers" all those edge cycles, that is, it allows to merge them with the maintenance cycle.

An example of the reduction. In order to illustrate our reduction from minimum vertex cover to the RSR-M problem, an example is given in Figure 4: the top part shows the graph G, while the lower part shows the edge and the maintenance cycles along with the departure/arrival times. In particular, all the routes take two days[2] and all the routes in the same cycle have the same arrival and departure time (for the sake of legibility, Figure 4 shows the departure and the arrival time – 12:00 and 12:00+2d, respectively – of only one route in the maintenance cycle). Hence, in every cycle, the arrival time of one route matches the departure time of the next route (on a different day due to the 2 days traveling time). Moreover, all the routes in the maintenance cycle have the same departure and arrival time.

Let us observe that the trivial solution consists of the maintenance cycle (yielding $2(n + 1) = 10$ days) and of $m = 5$ edge cycles (i.e. $4m = 20$ days). So, we have $C_{triv} = 30$. However, the trivial solution is not feasible since none of

[2] We denote this by "+2d" in the arrival time.

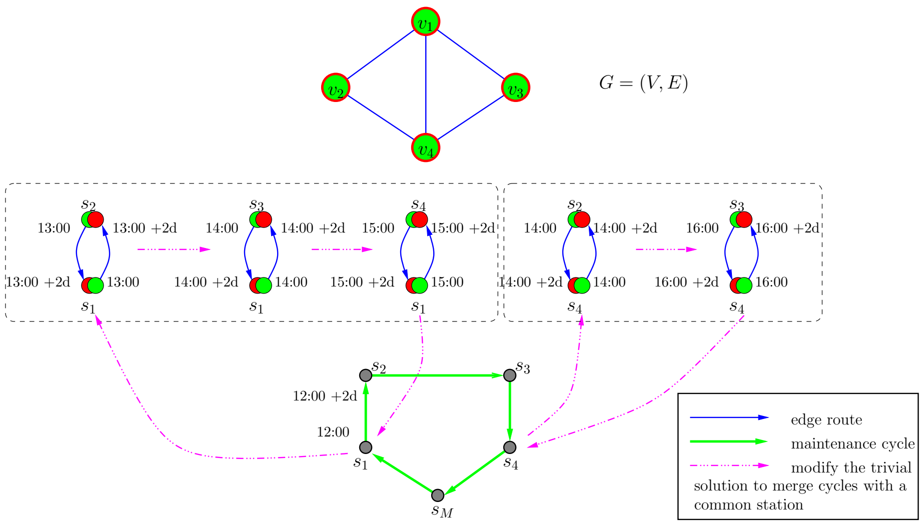

Fig. 4. An instance $G = (V, E)$ of minimum vertex cover (top) and the corresponding instance of RSR-M (bottom) along with a modified feasible solution corresponding to the vertex cover $K = \{v_1, v_4\}$

the edge cycles passes through s_M. To get a feasible solution, we must modify it in at least two stations, for example at stations s_1 and s_4 (corresponding to the vertex cover consisting of v_1 and v_4). We represent such changes in Figure 4 with dashed arrows: an arrow from a station in the maintenance cycle to an edge cycle means that we "leave" the maintenance cycle at this station and we "enter" in the corresponding edge cycle at the same station; an arrow between two edge cycles means that we follow them in the order given by the arrow. (Notice that there is always a station common to both cycles.) It is easy to verify that the cycle represented in Figure 4 has total length equal to $C_{triv} + 2 = 32$. (Whenever we leave the maintenance cycle and then we come back to the same station, we add one day.) □

We now formally state the properties that the arrival and departure time of each route must satisfy in the reduction. In particular, the length of each route is 2 days and the following properties hold:

P1. Every route in the maintenance cycle has departure time equal to 12:00 (and thus arrival time 12:00 + 2d).

P2. Each edge cycle is formed by two routes having the same arrival and departure time. Thus, every edge cycle has length 4. Moreover, each departure time is greater than 12:00 and any two routes with a common station but in different cycles have different departure/arrival times.

It is worth to observe that property **P2** can be guaranteed using departure times of the form 't:00', where t is some integer in $[13, 24]$. Indeed, since G has maximum degree 3, for every edge cycle, there are at most 4 other edge cycles with a common station. So, we are guaranteed that we can assign a departure time t to every edge cycle one by one (at each step there are only 4 values in $[13, 24]$ that we cannot use). Hence, the construction can be performed in polynomial time.

We first observe that, if we ignore the maintenance constraint, then the optimal solution is given by the union of the edge cycles and the maintenance cycle. This is due to the fact that in every cycle the arrival time of one route matches the departure time of the next route (of course two days later). Since every edge cycle has length 4 days and the maintenance cycle has a duration of $2(n+1)$ days, we have that the cost of this solution is

$$C_{triv} = 2(n+1) + \sum_{e \in E} 4 = 2n + 4m + 2. \tag{1}$$

Lemma 1. *A feasible solution of cost at most $C_{triv} + k$ to the constructed instance of* RSR-M *can be converted into a vertex cover of size at most k in the original graph in polynomial time, and vice versa.*

By using the above lemma, it is possible to prove that our construction is a PTAS-reduction [6] from minimum vertex cover on cubic graphs to RSR-M. Moreover, the above lemma (and thus the reduction) also applies to RSR-ME. Since minimum vertex cover is known to be APX-complete in graphs with degree bounded by Δ, for any $\Delta \geq 3$ [12,1], the following result holds:

Theorem 3. *The* RSR-M *and the* RSR-ME *problems are* APX*-hard.*

We remark that, for RSR-M, the reduction above can be modified so that the routes have more realistic travel times, e.g. so that each route takes approximately one hour. However, we presented the reduction using two-day routes in order to easily generalize the result to the case of empty movements.

4.2 Approximation Algorithms

We present a simple 2-approximation algorithm for the RSR-M problem. First, we ignore the maintenance constraint and compute a minimum-cost partition of the given routes into cycles using the algorithm of Section 3. The solution we get may contain cycles that do not pass through a maintenance station. As long as there exists a cycle in our solution that does not go through a maintenance station, we merge this cycle with some other cycle. Each of these steps increases the cost of the current solution by at most one day: one overnight wait is sufficient to combine two cycles with a common station.

If at some time step there is a cycle that does not pass through a maintenance station, but no combination with another cycle is possible, then the given instance does not have a feasible solution (because the stations on this cycle do not appear on any route outside the cycle). Otherwise, every cycle goes through a maintenance station in the end, and we obtain a feasible solution.

Let k be the number of cycles in the initial solution (the minimum-cost solution ignoring the maintenance constraint). The cost C_{triv} of this initial solution is a lower bound on the cost OPT of an optimal feasible solution. Besides, the cost of the initial solution is at least k, since each cycle has cost at least 1. Each application of the transformation combines at least 2 cycles, so there can be at most $k-1$ such transformations. Since each of them yields an extra cost of 1, the total cost of the final feasible solution is at most $C_{triv} + (k-1) \leq 2 \cdot OPT$.

Theorem 4. *The* RSR-M *problem admits a polynomial-time* 2*-approximation algorithm.*

Now we present an approximation algorithm for the problem RSR-ME. We make the (reasonable) assumption that the costs for empty train movements are symmetric, i.e., the cost for an empty movement from s_1 to s_2 is the same as the cost for an empty movement from s_2 to s_1.

First, we apply the algorithm of Theorem 4 and combine cycles containing a common station as long as possible (also combining two cycles not containing a maintenance station). Then, if a cycle does not pass through a maintenance station, it passes only through stations that do not occur on any other cycle. Therefore, we must use empty movements to combine such a cycle with another cycle.

We add empty movements by repeating the following step until the solution is feasible. Let σ be a cycle that does not pass through a maintenance station. For a station s on σ and a station s' not on σ, define $c(s, s')$ to be the sum of the cost of an empty movement from s to s' and an empty movement from s' to s. Select s and s' such that $c(s, s')$ is minimized. Add an empty movement from s to s' and one from s' to s and put extra trains at s and s'. Now we can assign the trains arriving at s and s' to outgoing routes from s and s' such that all cycles passing through s and s' are combined into one cycle.

Theorem 5. *The* RSR-ME *problem, restricted to empty movements with symmetric costs, admits a polynomial-time* 5*-approximation algorithm.*

By combining our approximation results with Theorem 3, we obtain the following result.

Corollary 1. *The* RSR-M *and the* RSR-ME *problem are* APX*-complete.*

The factor 2 in Theorem 4 comes from the fact that combining 2 cycles requires at most 1 extra train, which will only double the total solution cost if every train is in a one day cycle to begin with, and no two trains are ever in any station at once. In general, we expect our approximation algorithms to give better performance when applied to real data. The following theorem, providing a better analysis of the 2-approximation algorithm for RSR-M, gives a strong indication of this.

Theorem 6. *Consider an* RSR *instance, and let an optimal solution have a total of t trains. Also, let s be the number of stations and c be the minimum* *number of cycles possible for an optimal one day assignment to the* RSR *problem. For the same instance, but with maintenance stations specified, we can give a* $\min\{t+c-1, t+s-1\}/t$ *approximation for* RSR-M*.*

In the SBB data we look at, we see that we need over 100 trains to cover all routes, but we only have about 40 terminal stations. Further, in seeing that many trains are often within a station at once, this is also an indication that the number of cycles, c can be much less than the number of trains. Thus, we can prove that on these instances, our approximation factor will be significantly less than the worst case bound. In fact, for the SBB data, we find that we can combine all train movements into one cycle, without increasing the solution cost, which takes more than 100 days to complete. (This does include some unscheduled movements which they perform.) This is a very good indication that in real problem instances, we can hope to find maintenance solutions within a small percentage of optimal.

Acknowledgments

We are grateful to Jean–Claude Strüby and Peter Hostettler from SBB for several useful discussions and for providing us with SBB data.

References

1. P. Alimonti and V. Kann. Hardness of approximating problems on cubic graphs. In *Proc. 3rd Italian Conference on Algorithms and Complexity*, LNCS 1203, pages 288–298, Berlin, 1997. Springer-Verlag. 396, 399
2. A. Bertossi, P. Carraresi, and G. Gallo. On some matching problems arising in vehicle scheduling models. *Networks*, 17:271–281, 1987. 394, 395, 396
3. P. Brucker, J.L. Hurink, and T. Rolfes. Routing of railway carriages: A case study. *Memorandum No. 1498, University of Twente, Fac. of Mathematical Sciences*, 1999. 394
4. M.R. Bussieck, T. Winter, and U.T. Zimmermann. Discrete optimization in public rail transport. *Mathematical Programming*, 79:415–444, 1997. 394
5. G. Carpaneto, M. Dell'Amico, M. Fischetti, and P. Toth. A branch and bound algorithm for the multiple depot vehicle scheduling problem. *Networks*, 19:531–548, 1989. 394
6. P. Crescenzi and L. Trevisan. On approximation scheme preserving reducibility and its applications. In *Proc. 14th Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, LNCS 880, pages 330–341, Berlin, 1994. Springer-Verlag. 399
7. G. Dantzig and D. Fulkerson. Minimizing the number of tankers to meet a fixed schedule. *Nav. Res. Logistics Q.*, 1:217–222, 1954. 394, 396
8. J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. *Time Constrained Routing and Scheduling*. Elsevier, 1995. 394
9. T. Erlebach, M. Gantenbein, D. Hürlimann, G. Neyer, A. Pagourtzis, P. Penna, K. Schlude, K. Steinhöfel, D.S. Taylor, and P. Widmayer. On the complexity of train assignment problems. Technical report, Swiss Federal Institute of Technology Zürich (ETH), 2001. Available at `http://www.inf.ethz.ch/`. 395

10. R. Freling, J.M.P. Paixão, and A.P.M. Wagelmans. Models and algorithms for vehicle scheduling. *Report 9562/A, Econometric Institute, Erasmus University Rotterdam*, 1999. 394
11. A. Löbel. Optimal vehicle scheduling in public transit. *PhD thesis, TU Berlin*, 1998. 394
12. C. Papadimitriou and M. Yannakakis. Optimization, approximization and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991. 396, 399
13. C. Ribeiro and F. Soumis. A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research*, 42(1):41–52, 1994. 394
14. A. Schrijver. Minimum circulation of railway stock. *CWI Quarterly*, 6(3):205–217, 1993. 394