On the complexity of train assignment problems*

Thomas Erlebach† Martin Gantenbein‖ Daniel Hürlimann‡ Gabriele Neyer§
Aris Pagourtzis¶ Paolo Penna‖ Konrad Schlude‖ Kathleen Steinhöfel**
David Scot Taylor‖ Peter Widmayer‖

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 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.

*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).

†Computer Engineering and Networks Laboratory, ETH Zentrum, CH-8092 Zürich, Switzerland. E-mail: erlebach@tik.ee.ethz.ch

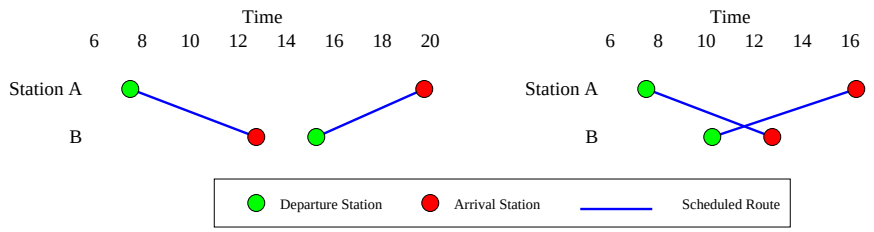
‡Institute of Transportation, Traffic, Highway- and Railway-Engineering (IVT), ETH Hönggerberg, CH-8093 Zürich, Switzerland. E-mail: huerlimann@ivt.baug.ethz.ch

§SMA and Partners Ltd., Transportation Engineers, Planners and Economists, Gubelstrasse 28, CH-8050 Zürich, Switzerland. E-mail: GabrieleNeyer@gmx.ch

¶Department of Computer Science, University of Liverpool, United Kingdom. E-mail: aris@csc.liv.ac.uk

‖Institute for Theoretical Computer Science, ETH Zentrum, CH-8092 Zürich, Switzerland, E-mail: {lastname}@inf.ethz.ch.

**GMD - FIRST, Berlin, Germany. E-mail: Kathleen.Steinhoefel@gmd.de



(a) An instance requiring only one train. (b) An instance requiring two trains.

Figure 1: Two simple train schedules

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 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

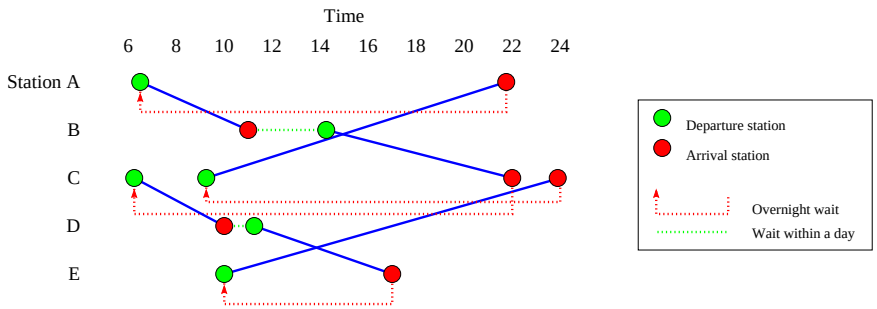


Figure 2: A cycle of four days.

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.

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.

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

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 [BCG87]. Dantzig and Fulkerson [DF54] propose the first solution that models the problem as a minimum cost circulation problem. A number of survey articles [DDSS95, BWZ97] 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 [Sch93, BHR99, CDFT89, RS94, FPW99, Löb98]. 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 [DDSS95, BCG87].

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

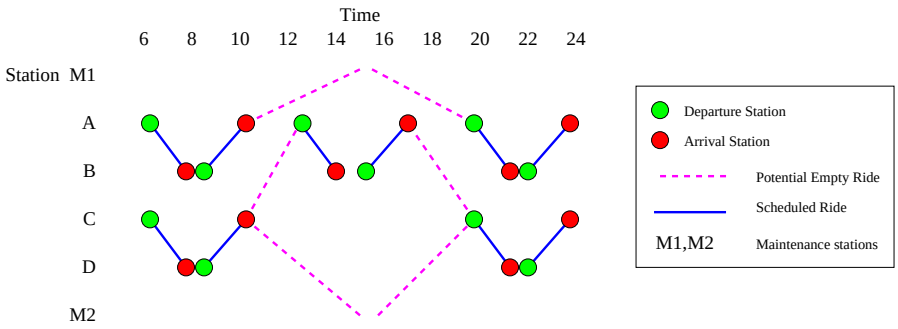


Figure 3: Output restrictions may change solutions.

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.

For the sake of readability, some of the proofs are provided in the Appendix.

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.

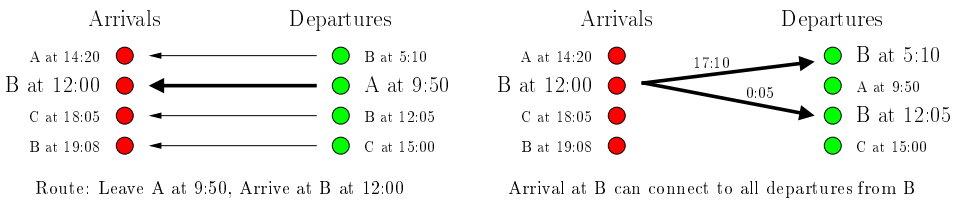


Figure 4: The bipartite matching formulation.

We clarify the relationship between one day assignments and more general forms of solutions (multiple day assignments) by proving several non-trivial results about the periodicity of general solutions and the cycle structure of one day assignments.

Lemma 1 *For* RSR *and* RSR-M*, if the number of trains at each station at the start of one day is specified, then the number of trains at every station at all times is uniquely determined, and will be the same between all solutions with the same start-of-day numbers.*

Lemma 2 *For* RSR *and* RSR-M*, consider the best one day assignment for a problem instance. Let Δt be a time interval in the assignment in which station s has at least one train for the entire interval, and let $R_{\Delta t}$ be the set of all routes arriving at or departing from that station during the interval. The lowest cost one day assignment can be modified, without change of cost, to place all routes in $R_{\Delta t}$ on the same cycle.*

Lemma 3 *For* RSR, RSR-E, *and* RSR-M*, there is an optimal periodic solution, for some period of p days.*

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 [BCG87], 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 (see Figure 4). 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.*

Empty train movements (RSR-E). When empty train movements are allowed, any arbitrary set of input routes has a feasible solution and each station is no longer required to have the same number of arrivals as departures. This gives much more flexibility for rostering, because we do not know a priori which empty routes to schedule.

We can solve the problem by using the bipartite matching formulation: in addition to the edges (and costs) above, we add edges representing empty train movements, until the graph is complete bipartite. The cost of an edge from arrival station s to departure station s', $s \neq s'$, is equal to the sum of the empty train traveling time from s to s', plus the waiting time that a train would have after arriving at s'. This model can also be modified to account for additional costs (see Section 1.2).

This relationship has been observed earlier [BCG87, DF54] and leads to a polynomial-time solution.

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 [PY91, AK97], 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.

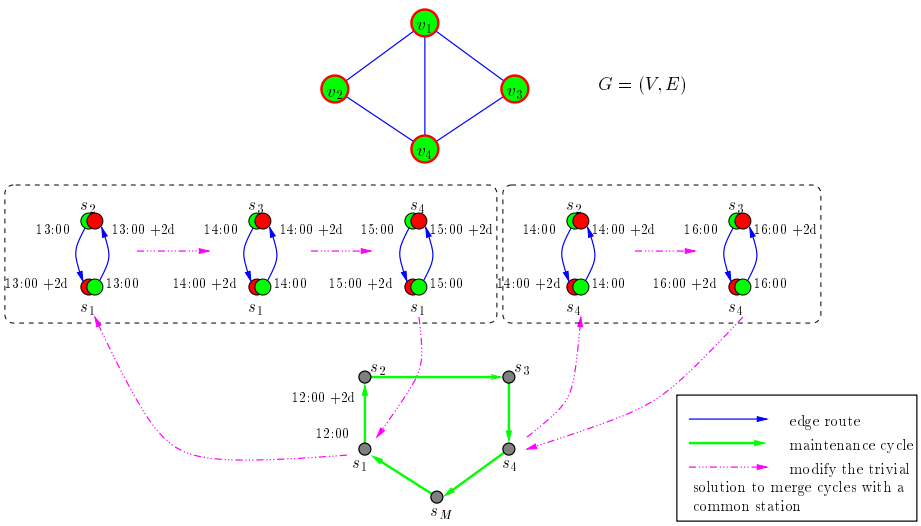


Figure 5: An instance $G = (V, E)$ of minimum vertex cover (top) and the corresponding instance of RSR-M (bottom) along with a modified feasible solution corresponding to the vertex cover $K = \{v_1, v_4\}$

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 5: 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 5 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

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

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 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 5 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 5 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

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

Lemma 4 *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.*

Proof. First, assume that we have a vertex cover $K = \{v_{i_1}, \ldots, v_{i_k}\}$ containing k vertices. We transform the infeasible trivial solution at the stations s_{i_1}, s_{i_2}, ..., s_{i_k}, one by one. At every step, we consider a station s_{i_j} such that $l \geq 1$ edge cycles containing s_{i_j} are not yet merged with the cycle containing s_M. First, we order such edge cycles according to the departure time of their routes. Let $C_1, C_2, \ldots, C_l$ be such an ordering. Then, we modify the current cycle containing s_M by inserting $C_1, C_2, \ldots, C_l$ in this order: C_1 occurs after (respectively, C_l occurs before) the route in the maintenance cycle with arrival s_{i_j} (respectively, the route with departure station s_{i_j}). Property **P2** implies that (i) the overall length of $C_1, C_2, \ldots, C_l$ is $4l$,

and (ii) the arrival time of C_l is bigger than 12:00, thus implying that when we come back at s_{i_j} we have to add one extra day. As at least one endpoint of every $e \in E$ is contained in K, all the edge cycles for the $e \in E$ are combined with the maintenance cycle into one big cycle in the end. Since, during the above transformation, every edge cycle is taken into account only once, we get a feasible solution to the instance of RSR-M with cost $C_{triv} + k$.

For the other direction, assume that we have a feasible solution F to the instance of RSR-M with total cost at most $C_{triv} + k$. Since F is a feasible solution and there is only one route into and out of the maintenance station s_M, F must consist of a single cycle. Let Q be the set of stations in $\{s_1, \ldots, s_t\}$ where F does not use the trivial solution, i.e., it does not match each arrival with the departure that happens at the same time. Let $\ell = |Q|$. At each station in Q, the total waiting-time is larger than in the trivial solution by at least one day. In every other station, the total waiting-time is at least the total waiting-time of the trivial solution at that station. Therefore, the total cost of F is at least $C_{triv} + \ell$. As F has total cost at most $C_{triv} + k$ by our assumption, we must have $\ell \leq k$. Each edge cycle must pass through at least one of the stations in Q (otherwise, all stations on the cycle would follow the trivial solution, implying that this input cycle is part of the solution, contradicting the assumption that F is feasible and consists of a single cycle). Therefore, the vertices v_j corresponding to the stations in Q are a vertex cover of G of size $\ell \leq k$. □

Now we show that our construction is a PTAS-reduction [CT94] from minimum vertex cover on cubic graphs to RSR-M. Since minimum vertex cover is known to be APX-complete in graphs with degree bounded by Δ, for any $\Delta \geq 3$ [PY91, AK97], we get the following result:

Theorem 3 *The* RSR-M *problem is* APX*-hard.*

Proof. Let an instance of minimum vertex cover in cubic graphs be given by an undirected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$. We can assume without loss of generality that G is connected and, therefore, $m \geq n - 1$. Since G has degree at most 3, we have

$$K_{opt} \geq \frac{m}{3}, \tag{2}$$

where K_{opt} denotes the size of an optimal vertex cover for G.

Consider the instance of RSR-M constructed from G as above. We will show that any $c(r)$-approximate solution for this instance of RSR-M, with $c(r) = \frac{1}{31}(r-1) + 1$, allows the efficient construction of an r-approximate solution for the instance of vertex cover, for any $r > 1$. Since the other properties of a PTAS-reduction are clearly satisfied, this will be sufficient to prove that our reduction is a PTAS-reduction in this case.

Let $A = C_{triv} + K_A$ denote the cost of some feasible solution to the constructed instance of RSR-M, and let OPT denote the cost of an optimal solution. By Lemma 4, we have $OPT = C_{triv} + K_{opt}$. Hence,

$$\frac{A}{OPT} = \frac{A - OPT}{OPT} + 1 = \frac{(K_A - K_{opt})}{C_{triv} + K_{opt}} + 1 \tag{3}$$

In order to bound the latter quantity from below, we need to bound C_{triv} with respect to K_{opt}. From (1) we get

$$C_{triv} = 2n + 4m + 2 \leq 10m. \tag{4}$$

By combining (2) and (4), we obtain $C_{triv} \leq 10m \leq 30K_{opt}$. Hence, we can bound the quantity in (3) as follows:

$$\frac{A}{OPT} \geq \frac{(K_A - K_{opt})}{31K_{opt}} + 1 = \frac{1}{31}\left(\frac{K_A}{K_{opt}} - 1\right) + 1.$$

If $A/OPT \leq c(r) = \frac{1}{31}(r-1)+1$, this implies $K_A/K_{opt} \leq r$. By Lemma 4, we can efficiently construct a vertex cover of size K_A from the solution to RSR-M with cost A. Thus, the reduction is indeed a PTAS-reduction as defined in [CT94]. □

We remark that 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:

Theorem 4 *The* RSR-ME *problem is* APX*-hard.*

Proof. Consider the reduction from the minimum vertex cover problem in cubic graphs as described above. View the constructed instance of RSR-M as an instance of RSR-ME where the duration of an empty train movement from any station to any other station is exactly two days. We claim that any feasible solution S to this instance of RSR-ME that uses empty train movements can be converted into a feasible solution S' that does not use empty train movements without increasing the cost. This conversion is achieved by removing each empty train movement from S and, for each removed empty movement from some station s to some station s', putting an extra train at stations s and s'. The cost of the two extra trains is equal to the cost of the removed empty movement. In the resulting schedule S', at least one station on each edge cycle must have an extra train so that all routes arriving at or departing from this station can be merged into a single cycle. Thus, all edge cycles can be merged with the maintenance cycle at no extra cost, and S' is feasible.

Again, we have that the original graph has a vertex cover of size at most k if and only if the constructed instance of RSR-ME has a feasible solution of cost at most $C_{triv}+k$. The remaining part of the APX-hardness proof for RSR-M remains valid for RSR-ME as well. □

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 5 *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 5 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 6 *The* RSR-ME *problem, restricted to empty movements with symmetric costs, admits a polynomial-time* 5*-approximation algorithm.*

By combining our approximation results with Theorems 3 and 4, we obtain the following result.

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

5 Conclusions

The factor 2 in Theorem 5 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 7 *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.

Proof. We have already shown the $(t + c - 1)/t < 2$ approximation. Further, we do not need to add more than one empty train per station, and we do not need one at the maintenance station, as all trains in this station are automatically on a cycle containing maintenance. □

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

[AK97] 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.

[BCG87] A. Bertossi, P. Carraresi, and G. Gallo. On some matching problems arising in vehicle scheduling models. *Networks*, 17:271–281, 1987.

[BHR99] 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.

[BWZ97] M.R. Bussieck, T. Winter, and U.T. Zimmermann. Discrete optimization in public rail transport. *Mathematical Programming*, 79:415–444, 1997.

[CDFT89] 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.

[CT94] 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.

[DDSS95] J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. *Time Constrained Routing and Scheduling.* Elsevier, 1995.

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

[FPW99] 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.

[Löb98] A. Löbel. Optimal vehicle scheduling in public transit. *PhD thesis, TU Berlin*, 1998.

[PY91] C. Papadimitriou and M. Yannakakis. Optimization, approximization and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[RS94] C. Ribeiro and F. Soumis. A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research*, 42(1):41–52, 1994.

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

A Postponed Proofs

A.1 Lemma 1

Proof. All train movements are defined by the scheduled routes. If at time t there are x trains at a station, and in the next Δt time units, a trains arrive and d trains depart, then there are $x + a - d$ trains at the station at time $t + \Delta t$. □

A.2 Lemma 2

Proof. Consider some lowest cost one day assignment, which does not have the above property on some interval $R_{\Delta t}$. There must be at least 2 different output cycles which have a train in station s during some time within Δt. By taking two such trains, and locally changing the train assignments so that they switch routes to be performed next, the two cycles have been merged into one. This can be continued until all routes in $R_{\Delta t}$ are on one cycle.

In the case of RSR-M, each of the initial cycles contained a maintenance station, and combining two cycles into one will not violate this constraint. □

A.3 Lemma 3

Proof. Notice that we can assume a finite number of possible states for the railway system at the start of each day: a finite number of trains in a finite number of locations. Eventually, the start of day configuration (which train is in which station) must repeat. In RSR-M, where all trains eventually are maintained, we can pick two such configurations far enough apart, say p days, so that all trains have been maintained in between them. For RSR-M and RSR, all costs stem from the number of trains in the system. This number is constant, so repeating these p days over will have the same cost per day as whatever assignments the optimal solution originally had, and the maintenance constraints are satisfied.

For RSR-E, in addition to the cost of train ownership, empty train movements have associated costs. Nevertheless, if an infinite day solution has cost c per day, then for some p, there must be a configuration which repeats after p days and has average cost c per day. Repeating these p days would give an optimal p day periodic solution. (We should be careful here, because we are dealing with average daily costs in an infinite time assignment, but the fact that there are only a finite number of discrete different costs available – one per potential empty movement – eliminates any convergence problems.) □

A.4 Theorem 1

Proof. We begin with the simplest case, RSR.

RSR. Consider an optimal solution S^*, and at each station s, let $start_s$ be the number of trains at station s at the start of the day in S^*. Let S' be *any* solution with $start_s$ trains at each station s at the start of the day. Lemma 1 implies that, in each station and at any time, S' has the same number of trains as S^*. We merely need to show that we can construct a 1 day feasible assignment.

Within station s, arbitrarily label these trains from 1 to $start_s$. Next, consider the first route that each physical train in the optimal solution makes within the first day: it is clear that there should be $start_s$ trains which make their first movement of the day from s. Next, consider all routes that each physical train uses within that day, in the optimal assignment. We can make each of the $start_s$ trains at station s make those same movements. Doing this from all stations will cover all routes for the day, using

the same number of trains as the optimal solution. At the end of the day, there will again be $start_s$ trains at station s, which made their last run of the day to arrive there. (Each station must have the same number of arrivals and departures per day to have any feasible solution.) Match these trains to the next day trains arbitrarily. This gives a one day assignment which will cover all routes, using the same number of trains as the optimal solution. Notice that, in the above proof, we are implicitly assuming that no overnight scheduled ride occurs. However, the proof can be easily adapted to deal with this case as well.

RSR-M. For an optimal solution S^* to RSR-M, we can begin in the same way, of course using $start_s$ numbers sufficient for RSR-M rather than RSR. By Lemma 2, we can assume that the one day assignment has only one distinct cycle overlapping any interval during which a station has at least one train. Next, consider the optimal solution: if at any time, on any day in the optimal p day assignment, routes r and r' are serviced consecutively by one train, then there must be at least one train (namely, that one) in the station during the time interval between atr and dtr'. By Lemma 1, our solution will have the same number of trains in s during that time interval, and by Lemma 2, r and r' can be assumed to be on the same cycle within our one day assignment. By Lemma 3, the optimal solution is periodic and is made of cycles, and by transitivity, all routes from one cycle in the optimal solution will be contained within a cycle of the one day assignment. Every route is on some cycle of the optimal solution, and all cycles are maintained, so our maintenance constraints are also covered in our one day assignment, and we are done with our proof for RSR-M.

RSR-E. For RSR-E, the proof is a bit more complex: due to empty movements, knowing how many trains begin the p day cycle at each station does not give sufficient information to calculate how many trains will be in each station at any point in time. Even for a specific time, this number may change from one day to the next, because the optimal cost p day assignment may have different empty movements on different days. Also, the cost measure can be more complex than just the number of trains: empty train movements are most likely more expensive than waiting within a station, and this must be weighed against the cost of additional trains. These additional costs must be added to the costs of potential empty movements.

We return to our output format definition, to clarify what we mean by a solution. Remember that the output can be represented as a set of edges from the departure to arrival events (for scheduled routes) and from arrival to departure events (for unscheduled movements or waiting within a station). Call these latter edges *empty train edges*, because there are no passengers on the train during these times. Without loss of generality, we can assume that each cycle within a solution alternates between scheduled route edges and empty train edges: first, two consecutive empty movements can be replaced by one, by summing their costs. Next, any two consecutive scheduled edges represent a scheduled route ending at a station, and then that train instantaneously beginning another scheduled route. In this case, we can insert a zero time, zero weight "empty train edge" between these two routes.

Consider the optimal p day assignment (which is optimal in general by Lemma 3), and the best one day assignment. If the schedule has r scheduled routes, then the optimal p day assignment will have $2rp$ total edges, while the optimal one day assignment will have $2r$. Let multigraph $G_{1 \cdot p}$ have one vertex for each scheduled arrival and departure event for each station. If the optimal one day solution has an edge from event a to b, $G_{1 \cdot p}$ has p such edges. Let multigraph G_p have the same vertices, but an edge from event a to b if that edge occurs somewhere in the p day schedule. The edge multiplicity is the number

of times it occurs in the p day assignment. Notice that the two graphs have the same number of edges, and that each vertex has the same in (out) degree in both graphs. This reflects the fact that in both solutions, each scheduled route happens every day, and is preceded and followed by one empty train edge.

Let $G_{1 \cdot p-p}$ be another multigraph, again with the same vertices. Consider an edge which occurs in $G_{1 \cdot p}$ x times, and in G_p y times. If $x = y$, then the edge is not in $G_{1 \cdot p-p}$. If $x > y$, then the edge is in $G_{1 \cdot p-p}$ with multiplicity $x - y$. If $x < y$ then the *reversed* edge, *with negated cost,* is in $G_{1 \cdot p-p}$ with multiplicity $y - x$. Notice that there will be no scheduled edges in $G_{1 \cdot p-p}$, as they will occur in both $G_{1 \cdot p}$ and G_p exactly p times. There will be the same number of edges with positive and negative weight, and each vertex will have the same in degree as out degree (e.g., for a departure event, the in degree from the one day assignment edges will be matched by the out degree of the reversed p day assignment edges). Finally, if the one day solution is more expensive than the p day solution, the total weight of the edges must be positive.

This multigraph can be partitioned into simple cycles, and one of them must have positive weight. All positive weight edges on this cycle occur in the one day assignment. None of the negative weight edges are in the one day assignment, instead they are in the p day assignment (reversed, with positive weight). If, within our one day assignment solution, we replace all of these positive weight edges with the negative weight ones (reversed, with positive weight), we will get a new solution. By choosing edges which form a cycle in $G_{1 \cdot p-p}$, we ensure that the new one day solution is also cyclic, that is, it is still a valid assignment. This one day solution will have a lower cost than the best one day solution, a contradiction, so it must be that the best one day solution did not originally have a higher cost than the optimal p day assignment, or the optimal assignment in general. This completes the RSR-E case, and the proof of the theorem.

□

A.5 Theorem 6

Proof. The cost C_{triv} of the initial infeasible solution is a lower bound on the cost OPT of an optimal feasible solution. Assume that the initial solution consists of k cycles. Then C_{triv} is at least k. When we combine two cycles, we add at most two extra trains. Therefore, the additional cost due to the extra trains is at most $2(k-1) \leq 2OPT$.

Furthermore, we claim that the total cost of empty movements added by the algorithm is at most $2OPT$. To see this, note that the method by which the algorithm adds empty movements can be viewed as the computation of a minimum spanning tree in an auxiliary graph H. Consider the instant when the algorithm was no longer able to merge cycles by putting extra trains at stations. The graph H contains one vertex for every cycle that existed at this instant and that did not pass through a maintenance station, and a single "maintenance vertex" that represents all cycles that existed at this instant and did pass through a maintenance station. H is a complete graph and the weight of an edge between two vertices is equal to $c(s, s')/2$, where s and s' are those stations in the cycles corresponding to the endpoints of the edge that minimize $c(s, s')$. Let MST denote the weight of a minimum spanning tree in H. The algorithm essentially grows trees by adding minimum-cost edges leaving one of the current subtrees and, therefore, computes a minimum spanning tree. Since it adds empty movements in pairs, the sum of the costs of all empty movements added in this phase of the algorithm is $2MST$. On the other hand, we can map all empty movements used in the optimal solution to the RSR-ME problem to edges of the graph H (ignoring the direction of the empty movements). The set of these edges must contain a spanning tree, because otherwise there would be a set S

of stations such that S does not contain a maintenance station and there is no empty movement or scheduled route between a station in S and a station outside S. Therefore, the cost of the empty movements in the optimal solution is at least MST.

Putting things together, we have that the total cost of the solution computed by the algorithm is at most $5OPT$: the initial infeasible solution has cost at most OPT, the added extra trains cost at most $2OPT$, and the additional pairs of empty movements cost at most $2OPT$. □