The Vehicle Rescheduling Problem with Retiming

Rolf van Lieshout[1], Judith Mulder[1,2], Dennis Huisman[1,3]

Econometric Institute Report EI2016-37

Abstract

When a vehicle breaks down during operation in a public transportation system, the remaining vehicles can be rescheduled to minimize the impact of the breakdown. In this paper, we discuss the vehicle rescheduling problem with retiming (VRSPRT). The idea of retiming is that scheduling flexibility is increased, such that previously inevitable cancellations can be avoided. To incorporate delays, we expand the underlying recovery network with retiming possibilities. This leads to a problem formulation that can be solved using Lagrangian relaxation. As the network gets too large, we propose an iterative neighborhood exploration heuristic to solve the VRSPRT. This heuristic allows retiming for a subset of trips, and adds promising trips to this subset as the algorithm continues. Computational results indicate that the heuristic performs well. While requiring acceptable additional computation time, the iterative heuristic finds improvement over solutions that do not allow retiming in one third of the tested instances. By delaying only one or two trips with on average 4 minutes, the average number of cancelled trips is reduced with over 30 percent.

Keywords. Vehicle rescheduling, retiming, Lagrangian heuristic.

[1]Econometric Institute, Erasmus University Rotterdam, 3000 DR Rotterdam, The Netherlands
[2]Corresponding author. Tel: +31 10-4081257. Email address: mulder@ese.eur.nl
[3]Erasmus Center for Optimization in Public Transport (ECOPT), Rotterdam, The Netherlands

1 Introduction

Unforeseen events in public transportation systems can lead to severely disrupted schedules, substantially harming passenger satisfaction. Many real time rescheduling methods have been proposed to reduce the impact of disruptions, often tailored to the mode of transport (e.g. road, air, railway) and the nature of the disruption (e.g. weather conditions, mechanical malfunctions or union strikes). In this paper we consider an extension of the *vehicle rescheduling problem* (VRSP), the problem that originates when a vehicle (bus) in a transportation system breaks down and cannot continue operation. Both the mode of transport of this problem (aircraft and railway rescheduling problems are better researched than road-based problems) and the nature of its disruption (most papers consider disruptions that only lead to delays) have received considerably less attention in the literature (Bunte & Kliewer, 2009; Visentini, Borenstein, Li, & Mirchandani, 2014).

Li, Mirchandani, and Borenstein (2004) introduced the VRSP. In Li, Mirchandani, and Borenstein (2009), the authors reconsider some of their assumptions and propose a new mathematical formulation for the VRSP. More specifically, they no longer exclude the possibility that trips other than the disrupted trip are cancelled and impose penalty costs on rescheduled trips, as it is often desirable that schedule changes are kept to a minimum. Li et al. (2009) propose a Lagrangian heuristic to solve the VRSP. Overall their results indicate that the heuristic performs well, strongly reducing the number of cancellations in comparison with an ad hoc rescheduling procedure.

The contribution of this paper is threefold. Firstly, we introduce the vehicle rescheduling problem with retiming (VRSPRT). As retiming, the possibility to delay trips, increases scheduling flexibility, formerly inevitable cancellations can be avoided, an idea that has been successfully demonstrated by Veelenturf, Potthoff, Huisman, and Kroon (2012) in the crew rescheduling problem. Secondly, by expanding the underlying recovery network with all relevant delay options, we can give a formulation of the VRSPRT that is very similar to the one of the VRSP, such that we can use the heuristic that Li et al. (2009) developed for the VRSP to solve the VRSPRT. Thirdly, by combining retiming with the idea of neighborhood exploration developed by Potthoff, Huisman, and Desaulniers (2010), we are able to find high quality solutions for large problem instances.

The paper is organized as follows. In Section 2, the VRSPRT is thoroughly defined. In Section 3, relevant literature on considering delays in recovery and scheduling problems is discussed. Section 4 presents a mathematical formulation for the VRSPRT. Section 5 presents the Lagrangian heuristic to solve the VRSPRT. Section 6 discusses how the concept of neighborhood exploration can be used to solve large problem instances. In Section 7 the results are presented, illustrating the added value of retiming. We wrap-up the paper with a conclusion and suggestions for further research.

2 Problem Description

The VRSP(RT) is defined for a bus transit system where a number of *trips* need to be performed. A trip consists of multiple stops where passengers are picked up and dropped off and has a specified starting and ending location and a starting and ending time. Every vehicle starts at the depot and executes a sequence of trips before returning to the depot. A pair of trips in a sequence needs to be *compatible*, meaning that a vehicle can perform the second trip after the first trip. Not all pairs of trips are compatible, as the times that they are carried out might overlap or it takes too long to get from the ending location of the first trip to the starting location of the second trip. Movements of a vehicle without passengers, from or to the depot or from the ending location of one trip to the starting location of the next trip, are referred to as *deadheads*.

The rescheduling needs to take place when one of the vehicles breaks down at a certain *breakdown time* and *breakdown location*. The trip currently being performed by the disrupted vehicle is referred to as the *cut trip* and the remainder of the trips scheduled for the disrupted vehicle is referred to as the *cut path*. In case the cut trip is a trip, a *back-up vehicle* needs to be sent to the breakdown location to pick up the stranded passengers and finish the remainder of the trip. This task is referred to as the *back-up trip*. In the other case, the cut trip is a deadhead, a back-up vehicle is not necessary. As the latter case gives an easier and less interesting problem, we only consider the case where the cut trip is an actual trip.

After the breakdown of one vehicle, the remaining operating vehicles can be seen as *pseudo-depots* from which one vehicle can be deployed at a certain *availability time* from a certain *availability location*. The availability time of vehicles performing a deadhead is equal to the breakdown time and the availability location is their current location. On the other hand, as ongoing trips must be finished before rescheduling, the availability time of vehicles performing a trip is equal to the ending time of their current trip, and the availability location is the ending location of their current trip. Moreover, vehicles can instantly be deployed from the real depot (if there are vehicles available), such that the availability time of the depot is also equal to the breakdown time.

The VRSP can be defined as follows. Given a current schedule, a breakdown time and a breakdown location, find a feasible schedule that minimizes the sum of cancellation and reassignment costs. Li et al. (2009) also include operation costs in this definition, but as the costs of operating the transit system can be regarded as sunk costs, we decided to focus only on cancellation and reassignments costs. Reassignment costs are taken into account as it is often undesirable to have many changes in the schedule (the crew of a vehicle might not be familiar with all trips and reassignments might lead to crews work overtime). The VRSPRT is simply the VRSP with the possibility to delay trips and with the inclusion of delay costs in the objective.

3 Literature Review

In the context of road-based vehicle rescheduling, Huisman, Freling, and Wagelmans (2004) are the first to take delays into account. They consider a dynamic environment where they reschedule when one or more vehicles have incurred delays. By imposing delay costs on arcs in a multi-commodity flow formulation, delays for future trips can be reduced or avoided. A disadvantage of modeling delays in this manner, is that not all delays on arcs can be computed beforehand, because delays might be propagated to later trips. For instance, if trip i can be executed by two vehicles, one of which is delayed, it is unknown whether delay costs should be imposed on arc (i, j). Li, Borenstein, and Mirchandani (2008) circumvent this problem by modeling delays explicitly in a multi-commodity network flow problem with time windows for the truck schedule recovery problem. However, this requires a large number of additional variables and constraints (one variable and one additional constraint for each trip and one variable and four constraints for each compatible pair of trips), making it impractical for solving medium or large sized instances without a specialized algorithm. The authors solve instances of only 23 vehicles and 31 trips using CPLEX. Trip cancellations are not considered in their research, but could easily be included in the problem formulation.

In the airline industry, flight delays are very costly, explaining why more papers take delays into account in the air-based rescheduling literature. Thengvall, Yu, and Bard (2001) incorporate delays in a multi-commodity flow formulation by adding a series of flight options for each flight that needs to be covered. The delay times are predetermined, so e.g. one flight option represents a 20-minute delay and a second option a 60-minute delay. All arc costs can be computed beforehand because the formulation is based on a time-space network rather than the connection-based network used by Huisman et al. (2004). On the other hand, their method potentially leads to redundant arcs and inefficiencies, since perhaps the first option is impossible to be reached or a plane that could depart with a 21-minute delay must wait for another 39 minutes. Other papers employ a set partitioning formulation, where every set represents a certain sequence of flights (trips). The advantage of the set partitioning formulation is that delays can be incorporated without difficulties, as the delay cost of a sequence of flights can be evaluated after the whole sequence is constructed. Løve, Sørensen, Larsen, and Clausen (2002) develop two neighborhood search heuristics to solve the resulting problem. Stojković and Soumis (2001) and Eggenberg, Salani, and Bierlaire (2010) take a different approach and solve the set partitioning problem using column generation. A difference between the two is that the former represents the recovery network as a connection based network, while the latter uses a time-space network. An advantage of the time-space network is that the time windows of every flight are satisfied in every path in the network. Consequently, the time windows need not be considered in the pricing problem, which is therefore easier to solve. On the other hand, Eggenberg et al. (2010) do not embed the algorithm in a branch-and-bound scheme,

so they cannot guarantee optimality.

Railway rescheduling problems are less similar to the VRSP as aircraft rescheduling problems, because the disruption of a train immediately influences surrounding trains since tracks and stations are shared across trains. However, some of the proposed solution approaches are still useful. Potthoff et al. (2010) solve the railway crew rescheduling problem using an iterative approach. As the number of possible duties is extremely large, initially only a subset of all duties is considered. The corresponding problem is referred to as the core problem and solved using column generation. In case some tasks are cancelled in the solution of the core problem, duties that lie in a neighborhood of the uncovered tasks are added to the core problem, after which the procedure is repeated. Veelenturf et al. (2012) extend the railway crew rescheduling problem by introducing retiming. Delay possibilities are incorporated into the problem by introducing copies of tasks that need to be executed, each with a different starting time. The problem is solved by column generation in combination with Lagrangian heuristics.

Retiming has not only been considered in rescheduling problems, but also in scheduling problems. In the (multiple-depot) vehicle scheduling problem with time windows (MDVSPTW), the starting time of every trip is not fixed, but restricted within a predefined time interval. Desaulniers, Lavigne, and Soumis (1998) formulate the MDVSPTW as a multi-commodity flow model with continuous time windows. They solve the model using branch-and-price. Due to the time windows, the column generation subproblem is a shortest path problem with resource constraints. Kliewer, Amberg, and Amberg (2012) use discrete instead of continuous time windows, meaning that the starting time of a trip can only be shifted by full minutes. The underlying network, represented using a time-space network, is extended with time window arcs representing the shifted options. Furthermore, they remove time window arcs that do not create new connections. Using this technique, all time windows are satisfied in all paths in the underlying network, such that it is not needed to include specific time window constraints in the mathematical formulation. Kliewer et al. (2012) use this modeling technique to solve the MDVSPTW and the crew scheduling problem simultaneously. Their solution approach consists of a combination of column generation and Lagrangian relaxation and is based on Huisman, Freling, and Wagelmans (2005).

As is clear, many different methods for including delays exist in the literature. However, to the best of our knowledge, no method exists that efficiently models all delay options and can give high quality solutions in real time, which is demanded for the VRSPRT. Consequently, we will now propose a new method for incorporating delays in rescheduling problems exhibiting the desired features.

4 Mathematical Formulation

Let $N = \{0, 1, 2, ..., n\}$ denote the set of future trips (viewed from the breakdown time). The back-up trip is denoted by 0 and has a starting time equal to the breakdown time, as it can be performed from the moment of breakdown, and a duration equal to the remaining duration of the cut trip plus a certain service time. Define D as the the set of all (pseudo-)depots, including the real depot, denoted by s, and all scheduled vehicles apart from the disrupted vehicle. Furthermore, t is used to denote the depot as an endpoint.

Define N^d as the set of trips that can be performed from (pseudo-)depot d, and $E^d = \{(i, j)|i < j, i \text{ and } j \text{ compatible}, i \in N^d, j \in N^d\}$ as the set of deadheads that can be performed from (pseudo-)depot d. The recovery network from (pseudo-)depot d without retiming is then given by $G^d = (V^d,\ A^d)$, where $V^d = N^d \cup \{d,\ t\}$ and $A^d = E^d \cup (d \times N^d) \cup (N^d \times t) \cup (d, t)$. To incorporate retiming, we associate with every trip $j \in N^d$ a set S_j^d of *relevant* starting times. A starting time of a trip is relevant if it is the earliest starting time creating a new connection. The relevant starting times of trip j are determined by the the availability time of the depot, the relevant starting times of all compatible predecessors i and the corresponding deadhead times. Furthermore, note that the size of S_j^d can be reduced by rounding the starting times to full minutes.

To expand the recovery network G^d with retiming options we first split each vertex i into a different vertex for each starting time in S_i^d (similar to the copies of tasks in Veelenturf et al. (2012)). We denote the set of vertices corresponding to trip i by $N^d(i) = \{n_i^1, n_i^2, ..., n_i^{|N^d(i)|}\}$. Next, we add arcs from the new vertices to the depot t. Further, we check for all trips i that cannot be performed from d without retiming whether they can be performed from this (pseudo-)depot if the trip is delayed. If this is the case, we add an arc from d to n_i^1 (there is only one relevant starting time for these trips as d is the only possible predecessor). Finally, we enumerate all pairs of vertices that correspond to trips and add an arc from vertex n_i^k to vertex n_j^l if n_j^l has the earliest starting time of the vertices in $N^d(j)$ that are compatible with n_i^k. This ensures that only arcs that create new connections are included in the network. We denote the resulting expanded recovery network from (pseudo-)depot d by $G_e^d = (V_e^d,\ A_e^d)$ and the set of all vertices corresponding to trips by N_e^d. Lastly, we define $A_e^d(k) = \{(i, j) \mid i \in N^d(k) \text{ and } (i, j) \in A_e^d\}$, the set of arcs leaving vertices associated to trip k.

An example of an extended recovery network is presented in Figure 1. The solid part is the network without retiming and the dotted part is added when delays are introduced. It can be seen that originally, trip 1 could not be performed after the back-up trip. However, if trip 1 is delayed (see vertex n_1^2) it can be performed after the back-up trip, with the side-effect that trip 2 needs to be delayed as well if one wants to execute it after trip 1. Note that even though it is possible that a vehicle performs for example trip n_2^2 after trip n_1^1, this arc is not added to the network as it is simply not necessary to delay trip 2 if it is performed after trip 1.

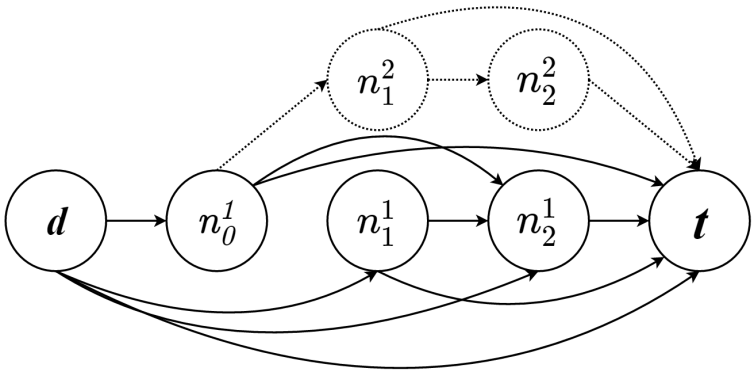


Figure 1: A recovery network extended with delay options. The dotted part of the network represents the options that become possible by introducing delays. d and t denote the starting and ending depot respectively.

Let c_{ij}^d denote the cost of arc $(i,j) \in A_e^d$, C_k the cost of cancelling trip k and W the number of back-up vehicles available at the depot. If we introduce decision variables y_{ij}^d, indicating whether arc $(i,j) \in A_e^d$ is selected, and z_k, indicating whether trip k is cancelled, the VRSPRT can be formulated as follows:

$$\min \quad \sum_{d\in D} \sum_{(i,j)\in A_e^d} c_{ij}^d y_{ij}^d + \sum_{k\in N} C_k z_k \tag{1}$$

$$\text{s.t.} \quad \sum_{\{j:(d,j)\in A_e^d\}} y_{dj}^d \leq W \qquad d = s, \tag{2}$$

$$\sum_{\{j:(d,j)\in A_e^d\}} y_{dj}^d = 1 \qquad \forall d \in D \setminus \{s\}, \tag{3}$$

$$\sum_{d\in D} \sum_{(i,j)\in A_e^d(k)} y_{ij}^d + z_k = 1 \qquad \forall k \in N, \tag{4}$$

$$\sum_{\{j:(i,j)\in A_e^d\}} y_{ij}^d - \sum_{\{j:(j,i)\in A_e^d\}} y_{ji}^d = 0 \qquad \forall d \in D, \forall i \in N_e^d, \tag{5}$$

$$y_{ij}^d, z_k \in \{0,1\} \quad \forall d \in D, \forall (i,\ j) \in A_e^d, \forall k \in N. \tag{6}$$

The objective (1) is to minimize the sum of reassignment costs and delay costs (which are both included in the arc costs c_{ij}^d) and cancellation costs. Constraints (2) and (3) assure that at most W vehicles depart from the depot and exactly 1 from every pseudo-depot. Constraints (4) guarantee that every trip is either executed or cancelled. In case the triangle equality holds for the arc costs or the penalty for each reassignment is sufficiently large, the equality sign in these constraint can be replaced by a "$\geq$" sign (Li et al., 2009). As both conditions are likely to hold, we assume this can be done in the remainder of the paper. Constraints (5) are flow conservation constraints.

Note that by expanding the recovery network with retiming options instead of modeling explicitly, we can give a formulation for the VRSPRT that has the same structure as the VRSP. In the next section, we will exploit this similarity to develop a heuristic for the VRSPRT.

5 Lagrangian Heuristic

Because of the similar structures of the VRSP and the VRSPRT we can modify the Lagrangian heuristic that Li et al. developed for the VRSP to solve the VRSPRT. In this section we give a description of this heuristic and discuss the adaptations that are needed to apply it to the VRSPRT.

An overview of the Lagrangian heuristic is as follows:

Step 1. Compute a lower bound by solving the Lagrangian problem.

Step 2. Compute an upper bound by applying an insertion-based primal heuristic that transforms the Lagrangian solution into a feasible solution.

Step 3. Update the Lagrangian multipliers based on subgradient search.

Step 4. Repeat steps 1 to 3 until the gap between the lower and upper bound is sufficiently small or a given time limit is exceeded.

5.1 Solving the Lagrangian

We relax constraints (4), $\sum_{d\in D}\sum_{(i,j)\in A_e^d(k)} y_{ij}^d + z_k \geq 1 \quad \forall k \in N$, and incorporate them in the objective function. The resulting Lagrangian problem can be written as

$$\theta(\boldsymbol{\lambda}) = \sum_{k\in N}\lambda_k + \sum_{k\in N}\kappa_k(\boldsymbol{\lambda}) + \sum_{d\in D\setminus\{s\}}\mu_d(\boldsymbol{\lambda}) + \nu_s(\boldsymbol{\lambda}), \tag{7}$$

where $\kappa_k(\boldsymbol{\lambda}), \mu_d(\boldsymbol{\lambda})$ and $\nu_s(\boldsymbol{\lambda})$ are sub-problems for trip k, pseudo-depot d and the real depot s respectively. The first sub-problem is given by

$$\kappa_k(\boldsymbol{\lambda}) \quad = \quad \min\,(C_k - \lambda_k)z_k, \text{ s.t. } z_k \in \{0,1\} \tag{8}$$

which has the trivial solution that $z_k = 1$ if $C_k - \lambda_k < 0$ and 0 otherwise. For the other sub-problems, let $T(i)$ return the trip $k \in N$ associated with vertex $i \in N_e^d$ and define

$$o_{ij}^d(\boldsymbol{\lambda}) = \begin{cases} c_{ij}^d, \text{ if } i = d \\ c_{ij}^d - \lambda_{T(i)}, \text{ otherwise.} \end{cases}$$

Then, we can write the second sub-problem as

$$\mu_d(\boldsymbol{\lambda}) \quad = \quad \min \sum_{(i,j)\in A_e^d} o_{ij}^d(\boldsymbol{\lambda}) y_{ij}^d \tag{9}$$

$$\text{s.t.} \quad \sum_{\{j:(d,j)\in A_e^d\}} y_{dj}^d = 1 \quad , \tag{10}$$

$$\sum_{\{j:(i,j)\in A_e^d\}} y_{ij}^d - \sum_{\{j:(j,i)\in A_e^d\}} y_{ji}^d = 0 \qquad \forall i \in N_e^d, \tag{11}$$

$$y_{ij}^d \in \{0,1\} \quad \forall (i,\ j) \in A_e^d, \tag{12}$$

which can observed to be a shortest path problem with transformed arc costs $o^d_{ij}(\boldsymbol{\lambda})$. This problem can be solved very efficiently, especially if we exploit the topological ordering of the graph, which can be achieved by sorting according to scheduled starting times as trip i can never be performed after trip j if $i < j$ (Cherkassky, Goldberg, & Radzik, 1996).

The sub-problem for the depot is given by

$$\nu_s(\boldsymbol{\lambda}) = \min \sum_{(i,j)\in A^s_e} o^s_{ij}(\boldsymbol{\lambda}) y^s_{ij} \tag{13}$$

$$\text{s.t.} \sum_{\{j:(s,j)\in A^s_e\}} y^s_{sj} \leq W \quad , \tag{14}$$

$$\sum_{\{j:(i,j)\in A^s_e\}} y^s_{ij} - \sum_{\{j:(j,i)\in A^s_e\}} y^s_{ji} = 0 \qquad \forall i \in N, \tag{15}$$

$$y^s_{ij} \in \{0,1\} \quad \forall (i,\ j) \in A^s_e, \tag{16}$$

and can be observed to be the problem of finding at most W least cost arc-disjoint paths from s to t (only paths with negative costs will be selected as the problem does not require there to be exactly W paths). Note that W is usually either 0 or 1. If W is 0, the objective value is always 0. If W is 1, the objective value is the minimum of the length of the shortest path and 0. For the general case, Li et al. propose to solve the third sub-problem using the algorithm of Jiménez and Marzal (1999).

As can be seen, for a given set of Lagrangian multipliers, $\theta(\boldsymbol{\lambda})$ can be solved by decomposing the problem into 'easy' problems that can be solved by inspection or for which specialized algorithms can be employed.

The Lagrangian multipliers are updated using the update formula from Held and Karp (1971), which is based on subgradient optimization. Based on initial experiments, we use an initial step-size of 7 and half the step-size every 6 iterations.

5.2 Primal Heuristic

To generate upper bounds, Li et al. (2009) propose a primal heuristic that computes a feasible solution from the Lagrangian solution. Because constraints (4) are relaxed, a trip might be covered multiple times or might not be covered nor cancelled in the Lagrangian solution. As cancellations are quite costly, the idea of the primal heuristic is to first remove redundant covering and then insert uncovered trips into existing paths (routes). Only if an uncovered trip cannot be inserted in any of the paths, it is cancelled. A more detailed description of the heuristic is given below.

Step 1. Remove redundant covering for each trip. If a trip is covered more than once, remove the trip from paths other than the cheapest path covering the trip.

Step 2. For each uncovered trip, determine in which paths it can be inserted and request an insertion in the path that yields the maximum cost decrease. If no feasible insertion position

exists in any of the paths, cancel it.

Step 3. For every requested insertion position, insert the trip that yields the maximum cost decrease.

Step 4. Repeat steps 2 and 3 until every trip is either covered or cancelled.

The cost decrease of an insertion is given by the difference between the objective function of the VRSPRT before and after the insertion. In this step, we include a very small portion of the operation costs in the objective to break symmetries. During a removal, we need to check whether trips after the removed trip can be performed with less delay. Consider the recovery network from Figure 1 and assume the back-up trip needs to be removed from the path $\{d, n_0^1, n_1^2, n_2^2, t\}$. As it is no longer necessary to delay trips 1 and 2, the result is the path $\{d, n_1^1, n_2^1, t\}$. During an insertion, we need to check whether the inserted trip and trips after the inserted trip need to be delayed to maintain a feasible path. For example, assume that trip 1 needs to be inserted in the path $\{d, n_0^1, n_2^1, t\}$. This insertion is feasible only if trip 1 and trip 2 are delayed, hence the result is $\{d, n_0^1, n_1^2, n_2^2, t\}$.

6 Iterative Neighborhood Exploration

Even with starting times rounded to full minutes and small allowed delays, the introduction of retiming can lead to a significant increase in the size of the recovery network. For instances of 700 trips, when we permit a maximum delay of only 5 minutes, the number of arcs and vertices can be more than three times as large compared to the network without retiming. As a result, the Lagrangian heuristic has a very hard time in finding good solutions. In five test instances, the heuristic could on average perform only nine iterations in 5 minutes. Due to this very limited number of iterations, instead of resulting in fewer cancellations, the introduction of retiming lead to solutions with more cancelled trips. As is clear, the size of the network needs to be reduced in order to realize a feasible solution method.

To this end, we apply the ideas of Potthoff et al. (2010) to the VRSPRT. In this paper, the authors consider a subset of all possible duties in the railway crew rescheduling problem. If not all tasks are covered in the resulting solution, duties in the neighborhood of uncovered tasks are added to the duties under consideration, after which the process is repeated. The neighborhood is defined such that duties in the neighborhood can possibly cover the uncovered task. Likewise, in the VRSPRT we can start by only allowing a subset of all trips to be delayed. If the Lagrangian heuristic gives a solution with cancelled trips, certain trips can be added to the subset. An overview of our approach is outlined in Figure 2.

More specifically, we maintain a set R of trips that are allowed to be retimed. Initially, only the back-up trip is in this set. In other words, we start by solving the VRSP. If the solution of the

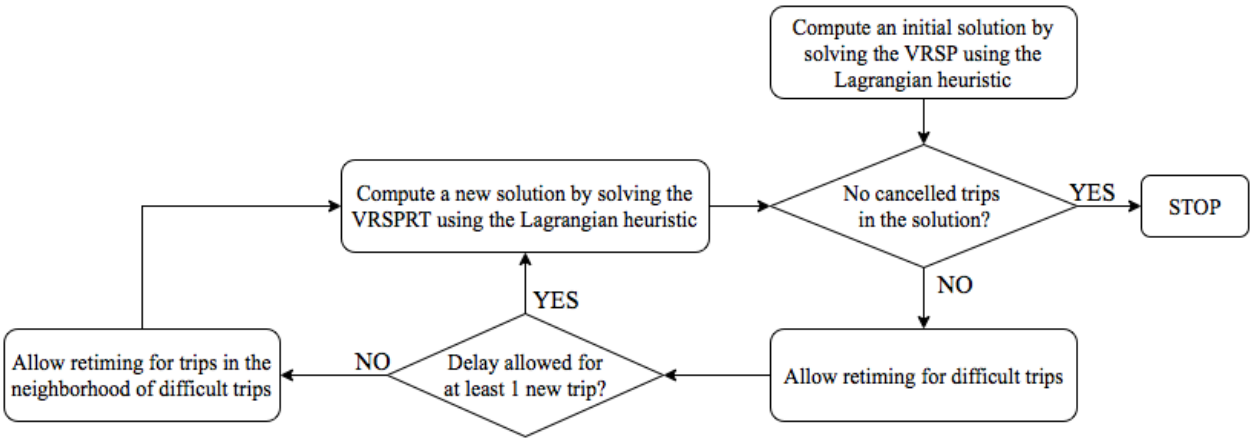


Figure 2: The iterative neighborhood exploration heuristic for the VRSPRT.

Lagrangian heuristic does not include any cancellations, the algorithm is terminated. Otherwise, trips in the neighborhood of difficult trips are added to R. To speed up the process, we do not only regard cancelled trips in the final solution of the heuristic as *difficult* trips, but all trips i for which it holds that (i) $z_i = 1$ in the Lagrangian solution (implying that this trip has a high Lagrangian multiplier, so it is difficult to perform) or (ii) is not performed in the Lagrangian solution (before the insertion heuristic). This broader definition enables us to extract more information from the Lagrangian optimization.

As for the neighborhood of a trip, let s_i and e_i denote the scheduled starting and ending time of trip i. Let τ_{ij} denote the deadhead time from the ending location of trip i to the starting location of trip j and let v_{ij} denote the deadhead time from the starting location of trip i to the starting location of trip j. We define that trip j lies in the neighborhood of trip i if it satisfies one of the following conditions:

1. $|s_j - s_i| < m - v_{ij}$,
2. $|s_j - e_i| < m - \tau_{ij}$,

where m is a given positive parameter. The intuition for this definition is as follows. If a trip from Rotterdam to Amsterdam starting at 1pm and ending at 3 p.m. is cancelled, this indicates that there is a shortage of vehicles in Rotterdam around 1 p.m. If we allow delays for trips starting near Rotterdam around 1 p.m. (the first condition), perhaps the cancellation can be avoided. Moreover, to prevent missed connections in Amsterdam if the formerly cancelled trip is delayed, delays can also be allowed for trips starting near Amsterdam around 3 p.m. (the second condition).

In our experiments, we initialize m with the value 20, as this leads to neighborhoods of appropriate sizes (on average 6 trips). In case all trips in the neighborhoods of difficult trips are already in R, we keep increasing m by 10 until we find trips in the neighborhood that are not in R. Finally,

to speed up the algorithm, the Lagrangian multipliers of the iteration of the Lagrangian heuristic that led to the best feasible solution are stored and used as initial values for the Lagrangian multipliers in the next iteration of the algorithm.

7 Computational results

We have performed a computational study to analyse the perfomance of the iterative neighborhood exploration heuristic and to investigate whether the introduction of retiming can lead to better solutions, especially regarding the number of cancellations. First, we describe the problem instances that we considered. Second, we compare the solutions of the iterative neighborhood exploration heuristic to the solutions of the VRSP. Finally, we test the sensitivity of the solutions with respect to the different cost parameters.

7.1 Experimental setup

Problem instances are generated as in Li et al. (2009). First, a vehicle scheduling instance consisting of 700 trips is generated using the method of Carpaneto, Dell'Amico, Fischetti, and Toth (1989). This method aims to simulate a real-life public transport system in which short and long trips are executed from 5 a.m. till midnight. Short trips have a trip duration around 50 minutes and are more likely to start in peak hours (7-8 a.m. and 5-6 p.m.). Long trips have a duration between 3 and 5 hours and have starting times that are distributed uniformly over the whole day. We consider two classes of instances. In class S all trips are short trips and in class M the ratio between short and long trips is 40:60. The vehicle scheduling instance is solved, after which an early short trip is selected as the cut trip. The breakdown occurs 80% into the cut trip. When a back-up vehicle arrives at the breakdown location, it serves the remainder of the cut trip after a service time of 3 minutes. Moreover, only vehicles that can arrive at the breakdown location within 25 minutes after breakdown are allowed to serve as back-up vehicles. As the number of cancellations in the VRSP with a back-up vehicle available at the depot ($W = 1$) is already only 0.3 on average, we only consider the more difficult case where there is no back-up vehicle available at the depot ($W = 0$). The recovery network is constructed accordingly.

To incorporate costs for schedule disruptions, a penalty P is added to arcs $(i, j) \in A_e^d$ if the trip corresponding to node j is not originally performed from depot d. The cancellation costs are defined as $C_i = 5t_i + C$, where t_i is the duration of trip i and C is a fixed component. We impose a very large cancellation cost on the back-up trip to make sure it is performed. The delay costs are v per minute and also incorporated in the arc costs. No delay costs are imposed for the back-up trip. Lastly, starting times of trips are rounded to full minutes to limit the number of retiming options.

We generated 50 instances for each class. Average characteristics of the problem instances are

presented in Table 1. The average number of vehicles in class M is larger than in class S, because of the longer trips that exist in class M. This also explains why the average number of trips in the cut path is larger for class S, as in class S more trips can be scheduled per vehicle. Note that even though instances in class M might seem easier to solve (there are more vehicles that can be used for rescheduling and fewer trips in the cut path), this turns out not to be the case as many vehicles are performing long trips at the moment of breakdown and can only be rescheduled when this trip is finished.

Table 1: Average characteristics of the used problem instances.

Class	Vehicles	Remaining trips	Trips in cut path
S	109.2	562.3	6.2
M	168.2	567.1	5.0

7.2 Performance of the Iterative Neighborhood Exploration Heuristic

In this first set of experiments, we set P equal to 500, C to 2000 and v to 20. The maximum delay we allow is 10 minutes. To make sure that the cancellations are reduced by retiming and not by a longer running time of the heuristic, we first find a good solution without retiming by running the Lagrangian heuristic for 2 minutes. For every next iteration of the Lagrangian heuristic we set a time limit of 1 minute. The algorithm terminates if we find a solution without cancellations or if the total running time reaches 6 minutes. The algorithm is implemented in Java on a HP EliteBook 8460p running Windows 10 with an Intel Core i5 processor at 2.5 GHz and 4 GB of RAM.

In Table 2 we present the average number of cancelled trips, reassigned trips and average costs of the solutions found by the iterative heuristic and by the heuristic without retiming. The iterative heuristic finds solutions with fewer cancellations in 26 percent of the cases in class S and in 36 percent of the cases in class M. On average, retiming reduces the number of cancelled trips from 0.94 to 0.60 and from 1.48 to 1.04, respectively. To achieve this reduction, slightly more trips need to be reassigned. The decrease in objective value when retiming is introduced is partially compensated by extra incurred costs for delays and reassignments, but the decrease is still substantial, at 523 (11 percent) for class S and 858 (14 percent) for class M. In Figure 3, the histograms for the number of cancelled trips with and without retiming are displayed. It can be seen that the entire distribution of the number of cancelled trips shifts to the left when retiming is introduced.

From Table 2 and Figure 3 it can be concluded that the iterative heuristic certainly achieves the desired results with respect to the number of cancelled trips. We will now analyse whether the introduced delays to achieve this reduction are not too long. In Table 3 we present the average number of cancelled, reassigned and delayed trips, the average delay length, the average cost

Table 2: Results of the iterative heuristic compared to the heuristic without retiming.

Class	Without retiming			With retiming			Fewer cancellations due retiming (%)
	Cancelled trips	Reassigned trips	Costs	Cancelled trips	Reassigned trips	Costs	
S	0.94	4.90	4628	0.60	5.40	4105	26
M	1.48	3.66	5996	1.04	4.20	5138	36

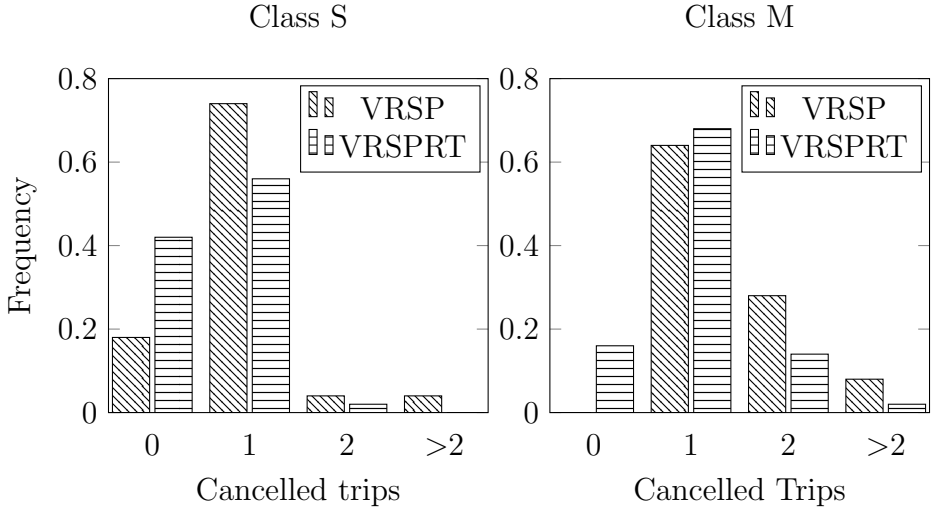


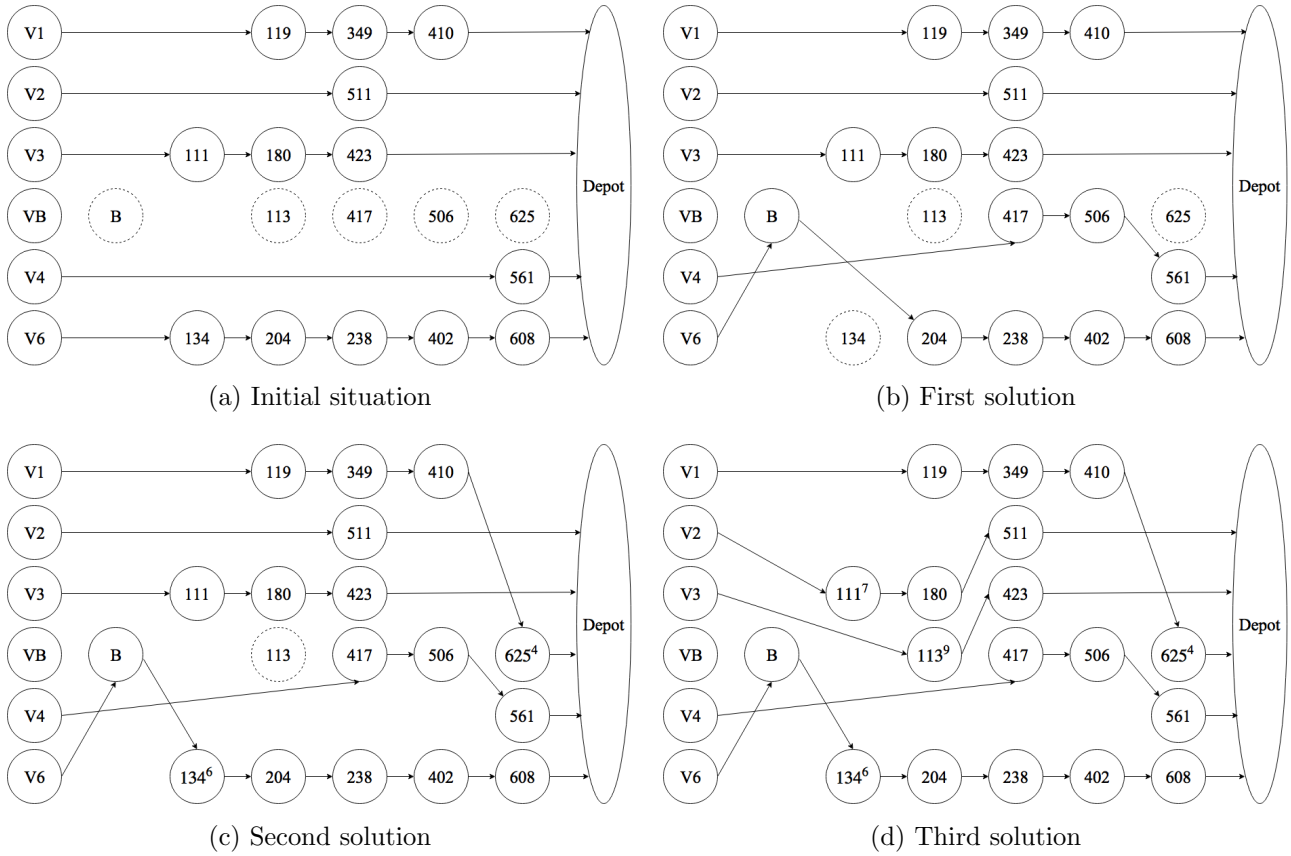
Figure 3: Histograms of the number of cancellations with and without retiming for class S and class M.

reduction and the average number of trips for which retiming was allowed ($|R|$) for the solutions where cancellations were avoided using retiming. As can be seen, the number of delayed trips is very moderate; on average only 1.2 trips in class S and 1.4 trips in class M. Furthermore, the length of the delays that are needed to avoid cancellations are more than reasonable; on average only 4 or 5 minutes. Finally, note that $|R|$ is below 70 in both classes. This shows that using a proper definition of difficult trips and the neighborhood of a trip, we are able to identify the trips for which retiming is really beneficial. All in all, it can be concluded that the iterative neighborhood exploration heuristic performs well and that retiming allows for solutions preferred to those of the VRSP.

Table 3: Characteristics of the solutions where the iterative heuristic was successful.

Class	Cancelled trips	Reassigned trips	Delayed trips	Average delay	Cost reduction (%)	$\|R\|$
S	0.23	6.62	1.23	4.31	33	68.4
M	0.61	5.33	1.39	5.04	32	62.0

Now that we have presented the average results, we will further illustrate the behavior of the iterative heuristic using one of the instances in class M as an example. The original schedule for the vehicles that turn out to be relevant is visualized in Figure 4a. Nodes with a letter V represent the pseudo-depots for the relevant vehicles and nodes with a number correspond to trips. VB denotes the vehicle that breaks down, and B denotes the trip where the breakdown takes place. Nodes with dotted perimeters indicate that the the trip is cancelled. As this figure depicts the situation before rescheduling, all five trips in the cut path are cancelled.



(a) Initial situation

(b) First solution

(c) Second solution

(d) Third solution

Figure 4: Visualization of the initial schedule and solutions found by the iterative heuristic. Only the schedule for the vehicles that are rescheduled somewhere in the process are depicted. A node with a dotted perimeter corresponds to a cancelled trip. If a trip is delayed, the superscript indicates the length of the delays.

In Figure 4b, the schedule corresponding to the initial solution of the iterative heuristic (the solution of the VRSP) is presented. As can be seen, vehicle 6 performs the back-up trip, which causes trip 134 to be cancelled. Vehicle 4 performs two other trips from the cut path. In total, three trips are still cancelled in this solution. The cost of this solution is 8576.

The iterative heuristic identifies five difficult trips in the first solution, trips 113, 134, 417, 450 and 625 (trip 450 is not in the figures). The difficult trips are allowed to be delayed in the second

iteration of the iterative heuristic. Using the Lagrangian heuristic, a better solution is found after 8 seconds, which is presented in Figure 4c. In this solution, trip 134 is delayed with 6 minutes and trip 625 with 4 minutes. Trip 113 is however still cancelled. The cost of the solution is 4776.

Trips 113 and 417 are identified as difficult trips in the second solution. As retiming was already allowed for these trips, the neighborhoods of trips 113 and 417 are explored, leading to 11 trips that are added to $|R|$. When the Lagrangian heuristic is invoked, an improvement over the previous solution is found in 13 seconds. The schedule for the relevant vehicles is visualized in Figure 4d. It can be seen that by delaying trip 113 with 9 minutes and trip 111 (one of the trips in the neighborhood of trip 113) with 7 minutes, all cancellations can be avoided. The final solution has total costs of 3521.

The process described above is typical for the behavior of the iterative heuristic. If the heuristic finds improvements over the solution of the VRSP, it finds these improvements quickly. Furthermore, if there are two or more cancelled trips in the initial solution, improvements are found in multiple steps.

7.3 Sensitivity Analysis

In the previous section we demonstrated the effectiveness of the iterative heuristic and we are able to conclude that the introduction of retiming leads to better solutions. In this section we repeat the analysis with different values for the reassignment penalty P and the delay costs per minute v. As we saw in the previous section that the number of reassignments was very limited, we test if we can reduce the number of cancellations even further if we set P at 50 instead of 500. Moreover, we also consider a delay costs v of 40 per minute instead of 20, as it is interesting to test whether the effectiveness of the heuristic deteriorates if we increase the delay costs.

In Table 4 we present the average number of cancelled trips, reassigned trips and average costs of the solutions found in class S by the iterative heuristic and by the heuristic without retiming for the different values of P and v. Table 5 shows the average number of cancelled, reassigned and delayed trips, the average delay length, the average cost reduction and the average $|R|$ of the solutions where cancellations were avoided using retiming in class S, for the different values of P and v. Table 6 and Table 7 contain the same results for class M.

We start by discussing the results for class S. In Table 4 we observe that, as expected, if P is 50 instead of 500 the number of cancelled trips is lower (0.90 versus 0.94), and the number of reassigned trips is higher (6.64 versus 4.97) in the initial solution without retiming. Still, the number of reassignments is rather small considering that theoretically more than 500 trips could be reassigned. If the option of retiming is introduced, the difference in the number of cancellations becomes even larger. This is because the iterative heuristic finds improvements more often with the lower reassignment penalty, in 36 to 42 percent of the instances instead of in 22 to 26 percent. The

effect of the higher delay costs per minute is less consistent. If P is 500, the number of cancellations is higher if v is 40 instead of 20, whereas if P is 50, the number of cancellations is lower when the delay costs are higher. This also means that when P is 50 the heuristic finds lower costs when the delay costs are higher. Overall, the fewest number of cancelled trips is 0.46 and is obtained using P equal to 50 and v to 40.

The characteristics of the solutions where retiming reduced the number of cancellations are in line with our expectations, see Table 5. If P is lower, fewer trips are cancelled and more trips are reassigned. Moreover, the effect of the higher delay costs is consistent and as expected for these solutions. If v is higher, the number of cancelled trips is higher and the average delay is shorter. The number of delayed trips is roughly the same. Apparently, when v is 40 instead of 20 the heuristic finds slightly worse solutions given that the number of cancellations is reduced, but this effect is compensated as the number of cancellations is reduced more often.

Table 4: Results of the iterative heuristic compared to the heuristic without retiming in class S, for different values of the reassignment penalty P and the delay costs per minute v.

		Without retiming			With retiming			
P	v	Cancelled trips	Reassigned trips	Costs	Cancelled trips	Reassigned trips	Costs	Fewer cancellations due retiming (%)
50	20	0.90	6.64	2381	0.48	8.22	1501	36
	40				0.46	8.04	1485	42
500	20	0.94	4.97	4628	0.60	5.40	4105	26
	40				0.68	5.28	4239	22

Table 5: Characteristics of the solutions where the iterative heuristic was successful in class S, for different values of the reassignment penalty P and the delay costs per minute v.

P	v	Cancelled trips	Reassigned trips	Delayed trips	Average delay	Cost reduction (%)	$\lvert R\rvert$
50	20	0.06	10.39	1.11	4.45	82	51.6
	40	0.19	10.14	1.14	3.46	75	54.0
500	20	0.23	6.62	1.23	4.31	33	68.4
	40	0.36	6.36	1.18	3.69	27	84.1

The results for class M, in Table 6 and Table 7, are somewhat different. Compared to when P is 500, when P is 50, the number of cancellations is slightly lower without retiming, and slightly higher with retiming, even though the number of reassignment is about twice as large in both cases. It turns out that the iterative heuristic finds improvement in fewer instances with the lower reassignment penalty. The number of cancelled trips is slightly lower if v is 20 compared to v

is 40, but the difference is not large. As for the solutions where the iterative heuristic reduced cancellations, in Table 7 we observe that if P is 50 these contain fewer cancelled trips than if P is 500. The effect of the different delay costs per minute depends on the reassignment costs.

Table 6: Results of the iterative heuristic compared to the heuristic without retiming in class M, for different values of the reassignment penalty P and the delay costs per minute v.

		Without retiming			With retiming			
P	v	Cancelled trips	Reassigned trips	Costs	Cancelled trips	Reassigned trips	Costs	Fewer cancellations due retiming (%)
50	20	1.44	7.42	3867	1.08	8.16	3174	36
	40				1.10	8.48	3227	32
500	20	1.48	3.66	5996	1.04	4.20	5138	36
	40				1.06	4.18	5073	38

Table 7: Characteristics of the solutions where the iterative heuristic was successful in class M, for different values of the reassignment penalty P and the delay costs per minute v.

P	v	Cancelled trips	Reassigned trips	Delayed trips	Average delay	Cost reduction (%)	$\lvert R \rvert$
50	20	0.50	9.89	1.50	4.48	59	84.8
	40	0.38	10.56	1.13	3.61	62	70.2
500	20	0.61	5.33	1.39	5.04	32	62.0
	40	0.63	4.79	1.58	5.03	33	84.3

From the sensitivity analysis we can conclude that the performance of the heuristic is quite robust with respect to the reassignment cost and delay costs per minute. A lower reassignment penalty leads to solutions with more reassignments, but this only results in a reduction in cancellations in class S. The effect of the delay costs is fairly small and not consistent across the tested cases.

8 Conclusion

In this paper, we discussed the vehicle rescheduling problem with retiming (VRSPRT). As the problem becomes intractable within reasonable time if we allow retiming for all trips, we proposed an iterative neighborhood exploration heuristic that allows retiming for a relevant subset of trips. Computational experiments indicate that the heuristic performs well. The iterative heuristic finds improvement over solutions that do not allow retiming in one third of the tested instances. By delaying only one or two trips with on average 4 minutes, the average number of cancelled trips

is reduced with over 30 percent. This shows that by using retiming, it is possible to find better solutions.

As we limited ourselves to vehicle rescheduling in this paper, a relevant topic for future research is to combine the vehicle rescheduling with crew rescheduling. This can range from varying the reassignment penalties to account for specific crew requirements up to complete integration.

9 References

Bunte, S., & Kliewer, N. (2009). An Overview on Vehicle Scheduling Models. *Public Transport*, *1*(4), 299–317.

Carpaneto, G., Dell'Amico, M., Fischetti, M., & Toth, P. (1989). A Branch and Bound Algorithm for the Multiple Depot Vehicle Scheduling Problem. *Networks*, *19*(5), 531–548.

Cherkassky, B. V., Goldberg, A. V., & Radzik, T. (1996). Shortest Paths Algorithms: Theory and Experimental Evaluation. *Mathematical Programming*, *73*(2), 129–174.

Desaulniers, G., Lavigne, J., & Soumis, F. (1998). Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research*, *111*(3), 479–494.

Eggenberg, N., Salani, M., & Bierlaire, M. (2010). Constraint-Specific Recovery Network for Solving Airline Recovery Problems. *Computers & Operations Research*, *37*(6), 1014–1026.

Held, M., & Karp, R. M. (1971). The Traveling-Salesman Problem and Minimum Spanning Trees: Part II. *Mathematical Programming*, *1*(1), 6–25.

Huisman, D., Freling, R., & Wagelmans, A. P. (2004). A Robust Solution Approach to the Dynamic Vehicle Scheduling Problem. *Transportation Science*, *38*(4), 447–458.

Huisman, D., Freling, R., & Wagelmans, A. P. (2005). Multiple-depot integrated vehicle and crew scheduling. *Transportation Science*, *39*(4), 491–502.

Jiménez, V. M., & Marzal, A. (1999). Computing the K Shortest Paths: A New Algorithm and an Experimental Comparison. In *Algorithm Engineering* (pp. 15–29). Springer.

Kliewer, N., Amberg, B., & Amberg, B. (2012). Multiple depot vehicle and crew scheduling with time windows for scheduled trips. *Public Transport*, *3*(3), 213–244.

Li, J.-Q., Borenstein, D., & Mirchandani, P. B. (2008). Truck Schedule Recovery for Solid Waste Collection in Porto Alegre, Brazil. *International Transactions in Operational Research*, *15*(5), 565–582.

Li, J.-Q., Mirchandani, P. B., & Borenstein, D. (2004). Parallel Auction Algorithm for Bus Rescheduling. In *Computer-Aided Systems in Public Transport* (pp. 281–299). Springer.

Li, J.-Q., Mirchandani, P. B., & Borenstein, D. (2009). A Lagrangian Heuristic for the Real-Time Vehicle Rescheduling Problem. *Transportation Research Part E*, *45*(3), 419–433.

Løve, M., Sørensen, K. R., Larsen, J., & Clausen, J. (2002). Disruption Management for an Airline - Rescheduling of Aircraft. In *Applications of Evolutionary Computing* (pp. 315–324). Springer.

Potthoff, D., Huisman, D., & Desaulniers, G. (2010). Column Generation with Dynamic Duty Selection for Railway Crew Rescheduling. *Transportation Science*, *44*(4), 493–505.

Stojković, M., & Soumis, F. (2001). An Optimization Model for the Simultaneous Operational Flight Pilot Scheduling Problem. *Management Science*, *47*(9), 1290–1305.

Thengvall, B. G., Yu, G., & Bard, J. F. (2001). Multiple Fleet Aircraft Schedule Recovery Following Hub Closures. *Transportation Research Part A*, *35*(4), 289–308.

Veelenturf, L. P., Potthoff, D., Huisman, D., & Kroon, L. G. (2012). Railway Crew Rescheduling with Retiming. *Transportation Research Part C*, *20*(1), 95–110.

Visentini, M. S., Borenstein, D., Li, J.-Q., & Mirchandani, P. B. (2014). Review of Real-Time Vehicle Schedule Recovery Methods in Transportation Services. *Journal of Scheduling*, *17*(6), 541–567.