The Road-Based Vehicle Rescheduling Problem: Methods and Extensions

Rolf van Lieshout
Student ID: 400376

June 28, 2016

Supervisor: prof. dr. Dennis Huisman
Co-reader: dr. Judith Mulder

Bachelor Thesis Econometrics and Operations Research
Erasmus School of Economics
Erasmus University Rotterdam

Abstract

In contrast to the well-researched aircraft and railway recovery problems, the road-based vehicle rescheduling problem (VRSP) is a relatively unexplored problem. This thesis builds further on Li, Mirchandani, and Borenstein (2009), who proposed solving the VRSP using a Lagrangian heuristic. This results in fewer cancelled trips than an ad hoc rescheduling procedure, emphasizing the benefits of optimized rescheduling. Besides replicating the results of Li et al. (2009), we compare their Lagrangian heuristic with a column generation based heuristic. This heuristic leads to better, very often optimal solutions, but computation times are longer. The last contribution of this thesis is the introduction of the vehicle rescheduling problem with retiming (VRSPRT), where trips can not only be cancelled, but also delayed. This increases scheduling flexibility such that less disruptive solutions can be obtained. To incorporate the delays in the problem formulation, we propose a new method that expands the underlying network to include all relevant delay possibilities. An advantage of this method is that both heuristics for the VRSP can directly be applied. For large instances, we propose a dynamic neighborhood exploration heuristic that allows retiming for a subset of all trips. The results indicate that retiming reduces the number of cancellations with 40% compared to the original VRSP.

Contents

1 Introduction

Despite increased attention for dynamic vehicle routing and scheduling problems and disruption management in particular, the *vehicle rescheduling problem* (VRSP) has largely remained untouched (Bunte & Kliewer, 2009; Visentini, Borenstein, Li, & Mirchandani, 2014). This problem, introduced in Li, Mirchandani, and Borenstein (2004), concerns the situation where a vehicle (bus) in a public transit system breaks down during operation. The other vehicles need to be rescheduled to serve the remainder of the disrupted trip and other trips originally scheduled for the disabled vehicle, while considering operation costs, schedule disruption costs and trip cancellation costs. As the satisfaction of passengers is greatly impacted by delays or even cancellation of trips and it has proven to be a scientific challenge to quickly find high quality solutions, the relevance of research on the VRSP is self-evident. The aim of this thesis is to contribute to the literature on the vehicle rescheduling problem by (i) implementing the methods of Li, Mirchandani, and Borenstein (2009) and replicating their results, (ii) comparing their Lagrangian relaxation based heuristic to a column generation based heuristic and (iii) generalizing their methodology to also allow for small delays, which can help to reduce cancellations even further.

The VRSP is an example of disruption management, which refers to the study of how to continue operating a certain system after one or more disruptions occur that prevent the original schedule from being executed (Yu & Qi, 2004). In the case of public transportation systems, disruptions can for example be caused by crew sickness, union strikes, mechanical malfunctions or even weather conditions. As the status of such a system changes simultaneously as the rescheduling is performed, it is required that solutions are found quickly, providing a large scientific challenge. In the VRSP, disruptions are caused by vehicle breakdowns, a topic that is remarkably less well researched than vehicle delays (Visentini et al., 2014). Another observation by the same authors, who present a review of schedule recovery models in road-, train- as well as air-based services, is that the literature on the road-based VRSP is very scarce compared to the train and aircraft recovery literature, with only main contributions from Huisman, Freling, and Wagelmans (2004) and Li et al. (2009).

Huisman et al. (2004) consider a dynamic environment where the vehicles are not scheduled beforehand, but iteratively as the day progresses. Delays in previously assigned trips and multiple scenarios for future travel times are taken into account. In Huisman and Wagelmans (2006) they integrate the dynamic vehicle scheduling with crew scheduling, which is necessary in order to apply the approach in practice. However, the authors state that it leads to very large computation times due to the complexity of the problem.

As Huisman et al. do not consider disruptions caused by vehicle breakdowns, this thesis will mostly build further on Li et al. (2009). In Li et al. (2009), the authors reconsider some of the assumptions made in Li et al. (2004) 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. A Lagrangian heuristic is developed to solve the VRSP. Overall their results indicate that the heuristic performs quite well, strongly reducing the number of cancellations in comparison with an ad hoc rescheduling procedure.

The thesis consists of two parts. First, the results of Li et al. are replicated: in Chapter 2, the VRSP is thoroughly defined and a mathematical formulation is given; Chapter 3 discusses the Lagrangian heuristic of Li et al.; in Chapter 4, the experimental setup is discussed and the results are presented. In the second part of the thesis, we build further on Li et al.: in Chapter 5, a column generation based heuristic is proposed and its performance is compared with the Lagrangian heuristic; Chapter 6 discusses the VRSP with retiming, including related literature, methodology and results. We wrap-up the thesis with a conclusion and suggestions for further research.

2 Problem Description and Mathematical Formulation

2.1 Problem Description

The VRSP is defined for a transit system (best comparable to a bus system) where a number of *service trips* need to be performed. A service 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 service 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 possible 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 service trip to the starting location of the next service trip, are referred to as *deadhead trips*.

The rescheduling needs to take place when one of the vehicles breaks down at a certain *breakdown time* and *breakdown place*. The trip currently being performed by the disrupted vehicle is referred to as the *cut trip*. In case the cut trip is a service trip, a *back-up vehicle* needs to be send 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* and is denoted by B. In the other case, the cut trip is a deadhead trip, a back-up vehicle is not necessary. 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*. The availability time of vehicles performing a deadhead trip is equal to the breakdown time. On the other hand, the availability time of vehicles performing a service trip is equal to the ending time of their current trip, as this trip must be finished before rescheduling. Moreover, vehicles can instantly be deployed from the real depot, such that the availability time of the depot is also equal to the breakdown time.

It is often undesirable to have many changes in the schedule, as the crew of a vehicle might not be familiar with all trips and reassignments might lead to crews work overtime. Therefore the objective of the VRSP is not only to minimize operating costs and cancellation costs but also schedule disruption costs.

2.2 Mathematical Formulation

We will now define some notation used to give a mathematical formulation of the VRSP. Let $N = \{B, 1, 2, ..., n\}$ denote the set of future service trips (viewed from the breakdown time) and let st_i denote the starting time of service trip i. The back-up trip is also regarded as a service trip, with a starting time equal to the breakdown time, as it can be performed from the moment of breakdown. 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.

All possible rescheduling possibilities from a (pseudo-)depot are captured in a so-called recovery network. Traditionally, in vehicle scheduling such a network is modeled as a connection-based graph, where nodes represent either depots or trips and edges represent connections (exceptions are Kliewer, Mellouli, and Suhl (2006) and Steinzen, Gintner, Suhl, and Kliewer (2010) who employ time-space networks to solve the multiple-depot vehicle scheduling problem and the integrated vehicle and crew scheduling problem respectively). The recovery network from (pseudo-)depot d is defined as $G^d = (V^d,\ A^d)$, where $V_d = N^d \cup \{d,\ t\}$ and $A^d = E^d \cup (d \times N^d) \cup (N^d \times t)$. Here, N^d denotes the set of service trips and $E^d = \{(i,\ j) | st_i < st_j, i \text{ and } j \text{ compatible}, i \in N^d, j \in N^d\}$

denotes the set of possible deadhead trips that can be performed from (pseudo-)depot d.

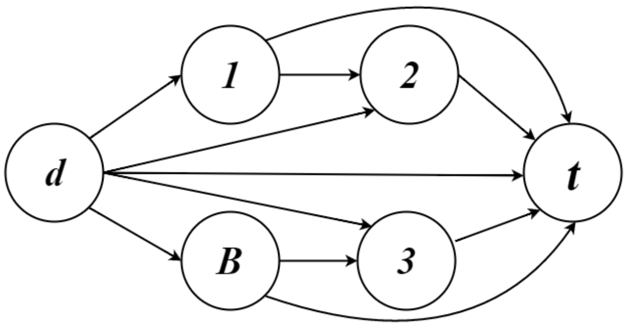

Figure 1: A recovery network. d and t denote the starting and ending depot respectively, B denotes the back-up trip and 1, 2, and 3 represent service trips.

An example of a recovery network is depicted in Figure 1. Since B is included in the recovery network, the back-up trip can be performed from d. There is an arc from every node to t because a vehicle can return to the depot after every trip. Every path from d to t represents a possible sequence of trips that can be performed from this (pseudo-)depot.

The decision variables are given by y_{ij}^d and z_i, where y_{ij}^d indicates whether arc $(i,\ j) \in A^d$ is selected and z_i indicates whether service trip i is cancelled. The VRSP can then be formulated as follows (see Li et al. (2009)):

$$\min \sum_{d\in D}\sum_{(i,j)\in A^d} c_{ij}^d y_{ij}^d + \sum_{i\in N} C_i z_i \tag{2.1}$$

$$\text{s.t.} \quad \sum_{\{j:(d,j)\in A^d\}} y_{dj}^d \le W \qquad d = s, \tag{2.2}$$

$$\sum_{\{j:(d,j)\in A^d\}} y_{dj}^d = 1 \qquad \forall d \in D - \{s\}, \tag{2.3}$$

$$\sum_{d\in D}\sum_{\{j:(i,j)\in A^d\}} y_{ij}^d + z_i = 1 \qquad \forall i \in N, \tag{2.4}$$

$$\sum_{\{j:(i,j)\in A^d\}} y_{ij}^d - \sum_{\{j:(j,i)\in A^d\}} y_{ji}^d = 0 \qquad \forall i \in N, \forall d \in D, \tag{2.5}$$

$$y_{ij}^d, z_i \in \{0,1\} \qquad \forall d \in D, \forall (i,\ j) \in A^d, \forall i \in N, \tag{2.6}$$

where c_{ij}^d is the cost of arc $(i,\ j) \in A^d$ and C_i the cost of cancelling trip i.

The objective is to minimize the sum of the operation costs and cancellation costs. Penalties for reassigning trips can be accounted for in the definition of the arc costs c_{ij}^d. Constraints (2.2) and (2.3) assure that at most W vehicles depart from the depot and exactly 1 from every pseudo-depot. Constraints (2.4) guarantee that every trip is either executed or cancelled and constraints (2.5) are flow conservation constraints.

Li et al. claim that the equality sign in constraints (2.4) can be replaced by a "$\geq$" sign if the triangle inequality holds for the costs c_{ij}^d (so $c_{ik}^d \le c_{ij}^d + c_{jk}^d$) or the penalty for each reassignment is sufficiently large. As both conditions are considered likely to hold, it is assumed that the constraints can indeed be replaced by "greater than" constraints throughout the thesis.

3 Lagrangian Relaxation Based Heuristic

As the VRSP needs to be solved quickly in order to be applicable in practice, Li et al. (2009) propose to solve the VRSP using Lagrangian relaxation. In this chapter, we give a thorough description of the methodology of Li et al. An overview of their approach 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.

3.1 Solving the Lagrangian

Li et al. relax constraints (2.4), $\sum_{d\in D}\sum_{\{j:(i,j)\in A^d\}} y_{ij}^d + z_i \geq 1$, and incorporate them in the objective function. The resulting Lagrangian problem can then be written as

$$\theta(\boldsymbol{\lambda}) = \sum_{i\in N}\lambda_i + \sum_{i\in N}\kappa_i(\lambda_i) + \sum_{d\in D-\{s\}}\mu_d(\boldsymbol{\lambda}) + \nu_s(\boldsymbol{\lambda}), \tag{3.1}$$

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

$$\kappa_i(\lambda_i) \quad = \quad \min\,(C_i - \lambda_i)z_i,\ \text{s.t. } z_i \in \{0,1\} \tag{3.2}$$

which has the trivial solution that $z_i = 1$ if $C_i - \lambda_i < 0$ and 0 otherwise. The second sub-problem is given by

$$\begin{aligned}
\mu_d(\boldsymbol{\lambda}) \quad &= \quad \min \sum_{(i,j)\in A^d} (c_{ij}^d - \lambda_i)y_{ij}^d && (3.3)\\
\text{s.t.} \quad & \sum_{\{j:(d,j)\in A^d\}} y_{dj}^d = 1 \quad , && (3.4)\\
& \sum_{\{j:(i,j)\in A^d\}} y_{ij}^d - \sum_{\{j:(j,i)\in A^d\}} y_{ji}^d = 0 \quad \forall i \in N, && (3.5)\\
& y_{ij}^d \in \{0,1\} \quad \forall k \in K, \forall (i,\ j) \in A^d, && (3.6)
\end{aligned}$$

and can be observed to be a shortest path problem. The sub-problem for the real depot is given by

$$\begin{aligned}
\nu_s(\boldsymbol{\lambda}) \quad &= \quad \min \sum_{(i,j)\in A^s} (c_{ij}^s - \lambda_i)y_{ij}^s && (3.7)\\
\text{s.t.} \quad & \sum_{\{j:(s,j)\in A^s\}} y_{sj}^s \leq W \quad , && (3.8)\\
& \sum_{\{j:(i,j)\in A^s\}} y_{ij}^s - \sum_{\{j:(j,i)\in A^s\}} y_{ji}^s = 0 \quad \forall i \in N, && (3.9)\\
& y_{ij}^s \in \{0,1\} \quad \forall k \in K, \forall (i,\ j) \in A^s, && (3.10)
\end{aligned}$$

and can be observed to be problem of finding at most W arc-disjoint paths from s to t with the lowest total costs (only paths with negative costs will be selected as the problem does not require there to be exactly W paths).

To solve the shortest path problem, Li et al. use the GOR1 algorithm of Cherkassky, Goldberg, and Radzik (1996), which runs in $O(|V| + |A|)$ time. This algorithm first computes a topological ordering of the vertices, after which all vertices are 'scanned' according to this order. However, Li et al. do not seem to realize that the first phase of this algorithm can be skipped, since for our case a topological ordering is readily available by construction (recall that for every arc (i, j) it holds that $st_i < st_j$), so sorting according to starting time gives a topological ordering. Therefore, we skip the first phase of GOR1 and immediately start scanning the vertices in the topological order. Pseudo-code for the used algorithm can be found in Appendix A.

To solve the third sub-problem, Li et al. employ an algorithm by Jiménez and Marzal (1999). However, by doing this, they implicitly relax constraints (3.9) and allow for non-negative integer-valued variables, as this algorithm computes the K shortest paths, which are not necessarily disjoint. A possible reason why the authors do this is that, despite polynomial algorithms for the K-shortest disjoint paths problem are available, the implementation is more laborious and the asymptotic running time is longer in comparison with finding non-disjoint paths (Cheng, Kumar, & Garcia-Luna-Aceves, 1989; Suurballe, 1974). Consequently, we follow Li et al. and use the algorithm by Jiménez and Marzal (1999) to solve $\nu_s(\boldsymbol{\lambda})$. Pseudo-code can be found in Appendix A.

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. An overview of the approach is the following procedure:[1]

Step 1. For each trip i, solve $\kappa_i(\lambda_i)$ by inspection. Compute the corresponding costs.

Step 2. For each pseudo-depot d, find the shortest path from d to t. Compute the corresponding costs.

Step 3. For the depot s, find shortest paths from s to t until W paths are found or the cost of the next path is positive. Compute the corresponding costs.

Step 4. Calculate $\theta(\boldsymbol{\lambda})$, a lower bound the the VRSP, by summing all Lagriangian multipliers and the costs in steps 1 to 3.

3.2 Primal Heuristic

To generate upper bounds, Li et al. (2009) propose a primal heuristic that computes a feasible solution from the Lagrangian solution. Because restrictions (2.4) are relaxed, in the Lagrangian solution a trip might be covered multiple times or might not be covered nor cancelled. 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.

[1] Li et al. (2009) refer to this procedure as the "Column generation procedure for solving the Lagrangian dual problem". However, no column generation is applied. Furthermore, this procedure does not solve the Lagrangian dual problem of finding the highest lower bound, but rather finds a lower bound.

Step 2. For each uncovered trip, compute 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 in any of the paths exists, 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 trips is either covered or cancelled.

The cost decrease of an insertion is given by the difference between the objective function of the VRSP before and after the insertion. It is not clear whether Li et al. (2009) allow for additional paths to be created from the depot in the second step of the heuristic. In this thesis, we allow it as long as permitted by the capacity of the depot.

3.3 Solving the Lagrangian Dual

The Lagrangian dual problem, the problem to find the best (highest) lower bound, is given by

$$\max\{\theta(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \geq \mathbf{0}\}. \tag{3.11}$$

Note that only non-negative values of the Lagrangian multipliers need to be considered since the relaxed constraints are 'greater than' constraints. The Lagrangian dual is solved by updating the Lagrangian multipliers using the commonly used update formula given in Held and Karp (1971), which is based on subgradient search:

$$\lambda_i^{k+1} = \max\{\lambda_i^k + \rho_k \frac{UB - \theta(\boldsymbol{\lambda}^k)}{\|g(\boldsymbol{\lambda}^k)\|^2} g_i(\boldsymbol{\lambda}^k), 0\},\ i \in N. \tag{3.12}$$

Here ρ_k denotes a step-size parameter, UB denotes the best known upper bound and $g(\boldsymbol{\lambda})$ denotes the vector of subgradients:

$$g_i(\boldsymbol{\lambda}) = 1 - z_i - \sum_{d \in D} \sum_{\{j:(i,j) \in A^d\}} y_{ij}^d,\ i \in N. \tag{3.13}$$

The intuition behind the update formula is that when a trip is not covered or cancelled in the Lagrangian solution, the penalty (Lagrangian multiplier) for violating the corresponding constraint is increased. Furthermore, when the optimum is approached, the gap between lower and upper bound gets smaller and the step-size decreases. On the other hand, if the size of the gradient is smaller, the step-size increases to still be able to obtain a sufficient decrease of the objective value.

Li et al. seem to be using a constant step-size where $\rho_k = \rho = 1$. However, to improve the likelihood that the optimization converges, in this thesis the step-size is halved every time the best lower bound has not increased in a certain number of iterations, a rule that has performed well empirically (Fisher, 2004). Also, multiple initial values are tested, to see whether there are notable differences in performance.

4 Computational Experiments

The goals of the computational experiments are to investigate the benefits of an optimized rescheduling approach compared to an ad hoc rescheduling approach and to analyse the effectiveness of the Lagrangian heuristic by Li et al. First, the experimental setup is discussed. Next, we explain the ad hoc rescheduling procedure. Afterwards, the results are presented and analysed.

4.1 Experimental Setup

The experiments are configured to be as similar to the ones by Li et al. as possible. In case of ambiguities, our interpretation is clearly stated.

Before we can reschedule, an initial schedule is needed. To this end, vehicle scheduling instances are 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 service trips (referred to as trips henceforth) are executed from 5 a.m. till midnight. Two classes of instances are examined. In class S all trips are short trips and in class M the ratio between short and long trips is 40:60. The following steps generate a vehicle scheduling instance with n trips.

Step 1. Generate v relief points, where v is an integer drawn randomly from $[n/3, n/2]$, and one depot on a 60 by 60 grids according to a uniform distribution. The travel time between two points is given by the Euclidean distance.

Step 2. Generate a number of short trips. Short trips start in peak hours (7-8 a.m. and 5-6 p.m.) with a probability of 30%. Exact starting times are determined using uniform distributions (either inside or outside the peak hours). The starting and ending relief points are drawn randomly from the set of relief points. The duration of a short trip is the travel time plus a random component between 5 and 40 minutes.

Step 3. Generate a number of long trips. Long trips have starting times that are uniformly distributed over the day. Long trips are round trips, such that only one relief point is drawn per trip. The duration of a long trip is between 3 and 5 hours.

Step 4. Compute the arc costs c_{ij}. If i and j both correspond to the depot, the cost is a combination of travel and idle time. If i corresponds to the depot, the cost is the travel time plus a fixed vehicle cost. If j corresponds to the depot, the cost only includes travel time.

The generated vehicle scheduling problem is formulated as a network flow problem (see e.g. Freling, Wagelmans, and Paixão (2001)) and solved using CPLEX, a general purpose MIP solver. Afterwards, an early short trip is selected as the cut trip. To obtain the breakdown location, it is assumed that during a trip vehicles travel in a straight line with constant speed from the starting to ending relief point. As Li et al., 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. The recovery networks for all (pseudo-)depots are constructed correspondingly. The arc costs for the VRSP are defined as follows:

$$c_{ij}^d = \begin{cases} c_{ij}, & \text{if trip } j \text{ is originally covered from (pseudo-)depot } d \\ c_{ij} + P, & \text{otherwise,} \end{cases}$$

where P is a penalty costs incurred for a reassignment. It must be noted that the fixed vehicle costs are subtracted from c_{ij}^d if $i = d$, as they can be considered sunk costs. Furthermore, the cancellation costs are defined as $C_i = 5 \times (\text{duration of } i) + C$, where C is a fixed component for every cancelled trip. Li et al. (2009) do not state how they make sure that the back-up trip is executed. In our experiments, we do so by imposing a very large cancellation cost on this trip.

The algorithms are 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.

4.2 Ad Hoc Rescheduling Procedure

According to Li et al., human schedulers often employ the following ad hoc rescheduling procedure in case of vehicle breakdowns. If no back-up vehicle is available at the depot, all scheduled trips for the disrupted vehicle are cancelled. The passengers are supposed to be picked up by a vehicle performing the same trip as the cut trip but at a later time. If a back-up vehicle is available, the human schedulers first checks when the first scheduled vehicle will pass by the breakdown location. If the back-up vehicle cannot arrive sooner, it is send out to perform the original schedule of the disrupted vehicle from the first compatible trip. On the other hand, if the back-up vehicle can arrive sooner, it is sent out to first pick up the stranded passengers and finish the remainder of the cut trip. Afterwards, it will continue to perform the schedule of the disrupted vehicle from the first compatible trip.

It must be noted that even though the ad hoc rescheduling solution can serve as a benchmark for the optimized approach, it is not always a feasible solution for the VRSP since the back-up trip is not always executed and the delay time of the stranded passengers can exceed 25 minutes. Consequently, the cost of the ad hoc solution are often lower than those of the optimized solution. Therefore, we only compare the solutions on the delay of the stranded passengers and the number of cancelled trips.

4.3 Results

Before performing the experiments, 10 test instances of class S were used to determine after how many iterations without improvement in the lower bound the step-size parameter ρ_k in equation 3.12 should be halved and what initial step-size should be used. The resulting gaps after running the algorithm for 60 seconds are presented in Table 1. As can be seen, an initial step-size of 3 and halving the step-size after 6 iterations without improvement seems to perform the best. These values are used in the remainder of this thesis.

Table 1: Optimality gaps for different parameter combinations of the subgradient search.

Initial	Allowed iterations without improvement					
step-size	2	4	6	8	10	12
1	0.68	0.54	0.57	0.66	0.66	0.66
2	0.59	0.49	0.68	0.79	0.68	0.63
3	0.40	0.15	0.12	0.58	0.58	0.85
4	0.26	0.37	0.32	1.06	0.67	1.17

To evaluate the performance of the heuristic, we test it on large instances of 700 originally scheduled trips. 10 instances of both classes are generated and solved for every configuration; three breakdown times given as a percentage into the cut trip (BT), two values for the fixed component of the cancellation cost (C), two values of the reassignment penalty (P), the case with a backup-vehicle available at the depot ($W = 1$) and the case without ($W = 0$). For these experiments, the iteration limit is set at 100 iterations and the time limit is set at 2 minutes, as solutions need to be provided quickly in real-life situations. The results are presented in Table 2. NRT is the average number of remaining trips, D the average delay for the cut trip, CT the average number of cancelled trips, RT the average number of reassigned trips, AHD and AHCT are the average delay and number of cancelled trips for the ad hoc rescheduling approach, CC the average cancellation costs, OC the average operating costs, G the average optimality gap and CPU the average computation time.

Table 2: Results of the Lagrangian heuristic for solving the VRSP.

Class S: only short trips. On average 109.8 vehicles.

				$W = 0$								
BT (%)	NRT	C	P	D	CT	RT	AHD	AHCT	CC	OC	G (%)	CPU (s)
20	604.4	1000	300	16.6	1.1	6.7	25.0	7.1	1,376	106,503	0.8	62.7
			500	13.3	1.2	6.2	25.0	7.1	1,521	107,188	0.3	48.9
		2000	300	15.5	1.1	6.8	25.0	7.1	2,481	106,273	1.4	79.1
			500	16.4	1.1	6.5	25.0	7.1	2,494	107,364	1.3	64.2
50	584.0	1000	300	17.8	0.8	6.9	25.0	7.1	1,044	102,307	0.5	57.3
			500	17.5	0.9	6.3	25.0	7.1	1,165	102,645	0.3	47.5
		2000	300	17.7	0.8	6.9	25.0	7.1	1,875	102,347	1.2	72.3
			500	16.4	0.8	6.5	25.0	7.1	1,884	103,355	1.1	56.5
80	569.0	1000	300	16.4	0.9	6.8	25.0	7.1	1,151	99,945	0.6	55.5
			500	17.6	1.1	6.1	25.0	7.1	1,393	99,829	0.3	42.2
		2000	300	18.5	0.9	6.8	25.0	7.1	2,061	99,032	1.4	72.7
			500	17.7	0.9	6.3	25.0	7.1	2,066	99,986	1.1	57.4
				$W = 1$								
BT (%)	NRT	C	P	D	CT	RT	AHD	AHCT	CC	OC	G(%)	CPU (s)
20	604.4	1000	300	14.2	0.3	2.0	32.4	1.3	345	107,272	0.9	64.2
			500	12.9	0.4	1.1	32.4	1.3	458	107,978	0.2	47.2
		2000	300	16.1	0.3	1.7	32.4	1.3	610	107,050	1.0	64.7
			500	15.9	0.2	1.5	32.4	1.3	409	108,376	0.5	49.2
50	584.0	1000	300	16.4	0.0	1.6	29.4	1.2	-	102,921	0.4	63.9
			500	15.3	0.2	1.0	29.4	1.2	232	103,846	0.2	42.3
		2000	300	14.7	0.1	1.6	29.4	1.2	201	103,061	0.7	56.7
			500	16.7	0.0	1.4	29.4	1.2	-	104,146	0.3	39.4
80	569.0	1000	300	15.9	0.3	1.3	28.7	1.1	336	99,619	0.6	56.5
			500	18.2	0.2	0.9	28.7	1.1	205	100,556	0.0	34.5
		2000	300	17.2	0.1	1.6	28.7	1.1	207	99,746	0.6	52.9
			500	17.8	0.1	1.1	28.7	1.1	197	100,728	0.1	35.6

Class M: 40% short trips, 60% long trips. On average 169.6 scheduled vehicles.

				$W = 0$								
BT (%)	NRT	C	P	D	CT	RT	AHD	AHCT	CC	OC	G (%)	CPU (s)
20	599.7	1000	300	15.2	1.5	4.5	25.0	5.3	2,304	118,590	1.5	74.6
			500	16.1	1.5	3.9	25.0	5.3	2,569	118,641	0.8	56.3
		2000	300	16.1	1.5	4.2	25.0	5.3	4,059	118,328	2.5	99.1
			500	14.9	1.5	4.1	25.0	5.3	3,773	118,881	1.7	69.3
50	583.9	1000	300	14.2	1.5	4.4	25.0	5.3	2,619	114,528	1.4	76.2
			500	13.2	1.7	3.8	25.0	5.3	2,807	114,772	0.9	56.4
		2000	300	13.7	1.5	4.3	25.0	5.3	4,393	114,352	2.5	94.5
			500	12.8	1.5	4.1	25.0	5.3	4,167	114,983	1.8	76.3
80	569.8	1000	300	14.1	1.7	3.6	25.0	5.3	3,125	110,664	1.4	69.6
			500	14.1	1.7	3.9	25.0	5.3	2,775	111,381	0.7	54.1
		2000	300	14.1	1.5	5.1	25.0	5.3	3,822	111,533	2.1	84.3
			500	14.1	1.7	3.8	25.0	5.3	4,591	111,356	1.8	75.4

				$W = 1$								
BT (%)	NRT	C	P	D	CT	RT	AHD	AHCT	CC	OC	G(%)	CPU (s)
20	599.7	1000	300	17.7	0.5	1.8	27.7	1.4	586	119,115	1.3	73.2
			500	16.1	0.5	1.1	27.7	1.4	590	119,256	0.1	36.0
		2000	300	15.4	0.5	1.8	27.7	1.4	1,072	119,128	1.5	86.3
			500	16.2	0.5	1.1	27.7	1.4	1,051	119,249	0.2	42.6
50	583.9	1000	300	14.8	0.5	1.8	24.6	1.5	716	115,040	1.0	60.7
			500	14.4	0.5	1.2	24.6	1.5	823	115,607	0.3	36.6
		2000	300	14.5	0.5	2.2	24.6	1.5	1,178	115,367	1.5	72.4
			500	14.1	0.5	1.4	24.6	1.5	1,328	115,670	0.6	40.8
80	569.8	1000	300	14.1	0.5	1.5	25.3	1.5	944	111,469	0.6	57.4
			500	14.5	0.6	0.8	25.3	1.5	1,135	111,780	0.1	25.8
		2000	300	13.7	0.5	1.5	25.3	1.5	1,424	111,547	0.9	62.5
			500	14.1	0.5	1.2	25.3	1.5	1,486	112,142	0.3	32.2

In general, the obtained results are in line with the results of Li et al. With relatively few reassignments, the number of cancellations and the delay for the cut trip are substantially reduced in comparison with the ad hoc approach. In class S, the number of cancellations is one average around 1 if $W = 0$ and close to 0 if $W = 1$. In class M more trips need to be cancelled compared to class S, on average 1.5 if $W = 0$ and 0.5 if $W = 1$. This can be explained by the fact that in class M most vehicles are performing long trips at the time of breakdown and can only be rescheduled when this trip is finished. Therefore, even though there are more scheduled vehicles in class M, there is less scheduling flexibility. The number of reassignments is smaller if $W = 1$, because trips originally scheduled for the disrupted vehicle can be executed by the vehicle from the depot (which is not considered reassigning), and is smaller in class M, because fewer trips are originally scheduled for the disrupted vehicle. Higher cancellation costs reduce the number of cancellations, but the difference is fairly small (on average only 0.1 over all configurations). Perhaps it is simply not possible to reduce cancellations further. On the other hand, higher reassignment penalties seem effective in reducing the number of reassignments. Interestingly enough, the number of cancellations barely increases in spite of the fewer reassignments. The fewest trips need to be cancelled when $C = 2000$ and $P = 500$.

In Table 3, a comparison between the results in Table 2 and in Li et al. is provided. We cancel fewer trips in class M, but more in class S. Our computation times for class S are significantly shorter, so perhaps we could find a better solution if we allowed for more iterations of the Lagrangian heuristic. However, the difference in computation times must be compared with care as we used a different computer than Li et al. A second possibility is that more trips had to be cancelled because more trips were originally scheduled for the disrupted vehicle, which can be derived from the higher number of cancelled trips for the ad hoc approach. A final explanation is the small number of solved instances per configuration. Overall, we consider the differences to be small and therefore we adopt the conclusion from Li et al. that their Lagrangian heuristic performs very well for solving the VRSP.

Table 3: Comparison of the average results in Table 2 with the results in Li et al. (2009).

		CT		AHCT		RT		Gap (%)		CPU (s)	
		Lieshout	Li	Lieshout	Li	Lieshout	Li	Lieshout	Li	Lieshout	Li
Class S	$W = 0$	1.0	0.6	7.1	6.1	6.6	6.1	0.8	0.6	59.7	93.1
	$W = 1$	0.2	0.1	0.2	1.0	1.4	1.4	0.5	1.0	50.6	93.1
Class M	$W = 0$	1.6	2.9	5.3	4.8	4.1	3.5	1.6	1.4	73.8	68.6
	$W = 1$	0.5	1.1	1.4	1.4	1.5	0.8	0.7	1.1	52.2	68.4

5 Column Generation Based Heuristic

For over three decades, column generation is one of the most successful and widely used techniques in (dynamic) vehicle routing and scheduling (Desaulniers, Desrosiers, & Solomon, 2002; Irnich, Desaulniers, et al., 2005). In this chapter, we develop a column generation based heuristic to solve the VRSP and compare its performance with the Lagrangian heuristic of Li et al.

5.1 Column Generation

Rather than solving the *compact* formulation of a problem (such as the multi-commodity flow formulation) column generation concerns solving the *extensive* formulation of a problem, which typically contains a huge number of variables.[2] The most important principle of column generation is that not all of these variables need to be considered explicitly. Instead, a sequence of reduced problems with only a subset of the variables (columns) are solved. Dual information of the solutions of the reduced problems are then used to find a new set of columns. If the optimal value of the reduced problem cannot increase if new columns are added, the obtained solution for the reduced problem is optimal for the original problem.

We will now explain column generation and its application to the VRSP in more detail. For the VRSP, the extensive formulation is as follows. Let Ω^d be the set of all feasible sequences of trips that can be performed from (pseudo-)depot $d \in D$. For each sequence $p \in \Omega^d$, let c_p denote the costs of the sequence and let the parameter a_{ip} be equal to 1 if the sequence includes trip $i \in N$ and 0 otherwise. As before, let C_i denote the cancellation cost of trip i. Furthermore, define for each sequence p the decision variable θ_p that indicates whether p is selected and define for each trip i the decision variable z_i that takes value 1 if the trip is cancelled. The VRSP can then be formulated as follows:

$$
\begin{aligned}
\min \quad & \sum_{d \in D} \sum_{p \in \Omega^d} c_p \theta_p + \sum_{i \in N} C_i z_i && && (5.1)\\
\text{s.t.} \quad & \sum_{d \in D} \sum_{p \in \Omega^d} a_{ip} \theta_p + z_i && = 1 && \forall i \in N, && (5.2)\\
& \sum_{p \in \Omega^s} a_{ip} \theta_p && \leq W && \forall d = s, && (5.3)\\
& \sum_{p \in \Omega^d} a_{ip} \theta_p && = 1 && \forall d \in D - \{s\}, && (5.4)\\
& \theta_p, z_i && \in \{0,1\} && \forall p \in \Omega^d, \forall d \in D, \forall i \in N. && (5.5)
\end{aligned}
$$

As can be seen, the extensive formulation of the VRSP is a set partitioning formulation with a huge number of decision variables, as there is a huge number of feasible sequences of trips. The objective is to minimize the sum of operation costs and cancellation costs. Constraints (5.2) ensure that every trip is either performed or cancelled. Often, these set partitioning constraints are replaced by set covering constraints (with a " $\geq$ " sign) as the linear relaxation of the set covering problem is more stable and integer solutions are more easily constructed for the set covering problem (Barnhart, Johnson, Nemhauser, Savelsbergh, & Vance, 1998). Constraints (5.3) and (5.4) assure that at most W vehicles depart from the depot and exactly 1 from every pseudo-depot.

Column generation is used to solve the linear programming relaxation of the above stated problem. This relaxed problem is referred to as the *master problem.* The master problem is

[2]Formally, the extensive formulation can be obtained by applying Dantzig-Wolfe decomposition to the compact formulation (Dantzig & Wolfe, 1960).

decomposed into two parts, the *restricted master problem* (RMP) and *pricing problems* for every (pseudo-)depot. The RMP at iteration n is the same as the master problem but restricted to a subset of all possible sequences $\Omega_n^d \subseteq \Omega^d$ for all d. The RMP is solved, resulting in a primal and dual solution. The role of the pricing problems is to verify whether the obtained solution is optimal and if not, to propose new columns that should be added to the RMP. To this end, the pricing problem for a (pseudo-)depot is to compute the minimum *reduced cost* of a column $\theta_p \in (\Omega^d \setminus \Omega_n^d)$. Columns with negative reduced costs have the potential to improve the objective value. Therefore, if the minimum reduced cost is positive for every d, the current optimal solution for the RMP is also optimal for the overall master problem. If not, columns with negative reduced cost are added to the RMP and the RMP is re-optimized.

If we denote the values of the dual variables associated with constraints (5.2) and (5.3, 5.4) by π_i and u_d respectively, the reduced costs of a column $p \in \Omega^d$ is given by $c_p - u_d - \sum_{i \in N} a_{ij}\pi_i$. As follows, the pricing problem for (pseudo-)depot d is equivalent to finding the shortest path in G^d from d to t where the arc costs are given by $c_{ij} - u_d$ if $i = d$ and $c_{ij} - \pi_i$ if $i \in N$. It is this result that makes column generation such an attractive method, as the pricing problems can often be solved using (pseudo-)polynomial algorithms.

5.2 Finding Integer Solutions

As column generation solves the linear relaxation of the extensive formulation, it does not produce integral solutions (at least in general). Therefore, to solve integer programs, more steps need to be taken. The traditional method to compute integer solutions is integrating column generation in a branch-and-bound scheme (Barnhart et al., 1998), referred to as *branch-and-price*. This allows solving the integer program to optimality. A different technique is to exploit the similarities between the pricing problems and the Lagrangian relaxation of the compact formulation (in our case both are shortest path problems). That is, to combine column generation with Lagrangian relaxation to compute high quality integer solutions (Huisman, Jans, Peeters, & Wagelmans, 2005). As the implementation of either method requires a large amount of fine-tuning before performing properly (e.g. specialized branching techniques and proper termination criteria), in this thesis a relatively simple method is used, namely the truncated column generation heuristic that Pepin, Desaulniers, Hertz, and Huisman (2009) propose for the multiple-depot vehicle scheduling problem.

Truncated column generation is an iterative heuristic consisting of two phases, a column generation phase and a rounding-up phase. The algorithm starts by solving the master problem using column generation. If the obtained solution is integer, a feasible solution has been found and the heuristic terminates. Otherwise, the rounding-up phase is invoked. All fractional variables larger than a parameter $\theta_{\min}$ are rounded up. If no such variable exists, the largest fractional variable is rounded up. After the rounding-up phase, the algorithm is repeated, but now with a part of the columns 'fixed'. This eventually produces a feasible solution for the set covering formulation of the VRSP, which can, if necessary, easily be transformed to a feasible solution of the set partitioning formulation by removing redundant covering.

Multiple acceleration strategies were tested to speed up the column generation heuristic. First, we tried generating more than 1 column with negative reduced cost in each pricing problem using the K-shortest paths algorithm discussed in Chapter 3. Second, we tried accepting the first found path with a negative reduced cost instead of only accepting the path with the minimum reduced cost. However, neither strategy yielded a improvement of the performance of the algorithm. On the other hand, a surprising observation during these tests was that more often than not, the heuristic never enters the rounding phase, implying that actually the exact optimum is found. For the vehicle routing problem with time windows, a problem with a similar formulation as the VRSP, it has been proven that the gap between integer and fractional optimal solutions decreases if the number of customers (comparable to trips in the VRSP) increases (Bramel & Simchi-Levi, 1997). Perhaps a similar result holds for the VRSP. As not only the final solution often is integer valued, but also intermediate solutions, we can keep track of the best found feasible solution.

5.3 Comparison of the Two Heuristics

To compare the performances of the Lagrangian based and the column generation based approach, we generate a number of instances and solve them using both heuristics. Furthermore, as the relative performance of the heuristics may differ with the instance size, instances of 300, 500 and 700 trips are considered. Both instances of class S and class M are tested, with and without an available back-up vehicle ($W = 0$ and $W = 1$). The breakdown time is 50% into the cut trip, C and P are set at 2000 and 500 respectively. We generate 10 instances for each configuration. As for the column generation heuristic, the original vehicle schedules serve as the initial set of trip sequences. $\theta_{\min}$ is set at 0.95.

The average results of the experiments are presented in Table 4. The column generation heuristic finds slightly higher costs for instances of 300 trips and slightly lower costs for instances of 500 and 700 trips. The computation times are more than twice as long, but are still reasonable. Of the 120 instances, the column generation heuristic finds optimal solutions in 112 cases. However, this only yields improvements over the Lagrangian heuristic in 19 cases. Apparently, the Lagrangian heuristic also finds optimal solutions very often. In 7 instances (out of 8 where it does not find the optimal solution) the column generation heuristic finds worse solutions.

Table 4: Average results of the column generation heuristic (CG) and Lagrangian heuristic (LG).

Trips	Total costs CG	CPU (s) CG	Total costs LG	CPU (s) LG	CG optimal	CG lower costs	CG higher costs
300	62,071	10.6	62,027	4.7	88%	18%	10%
500	86,313	59.6	86,332	23.2	100%	10%	0%
700	113,229	202.6	113,337	86.1	93%	20%	7%

In Figure 2 the the average objective value (of feasible solutions) is plotted against the computation time for instances with 700 trips in class M with $W = 0$. As can be seen, while the column generation heuristic finds better solutions eventually, the Lagrangian heuristic is better in finding reasonable feasible solutions quickly. On average, only after 160 seconds does the column generation heuristic find less costly solutions. This shows that the column generation heuristic needs a large set of columns before it can find high quality solutions. The same pattern is displayed for other instance sizes and configurations, and when considering individual instances instead of the average.

Summarizing, the comparison showed that both heuristics very often find optimal solutions. The column generation heuristic on average finds the best solutions, but its computation times are more than twice as long as the Lagrangian heuristic. Perhaps delicate acceleration strategies can speed up the column generation algorithm.

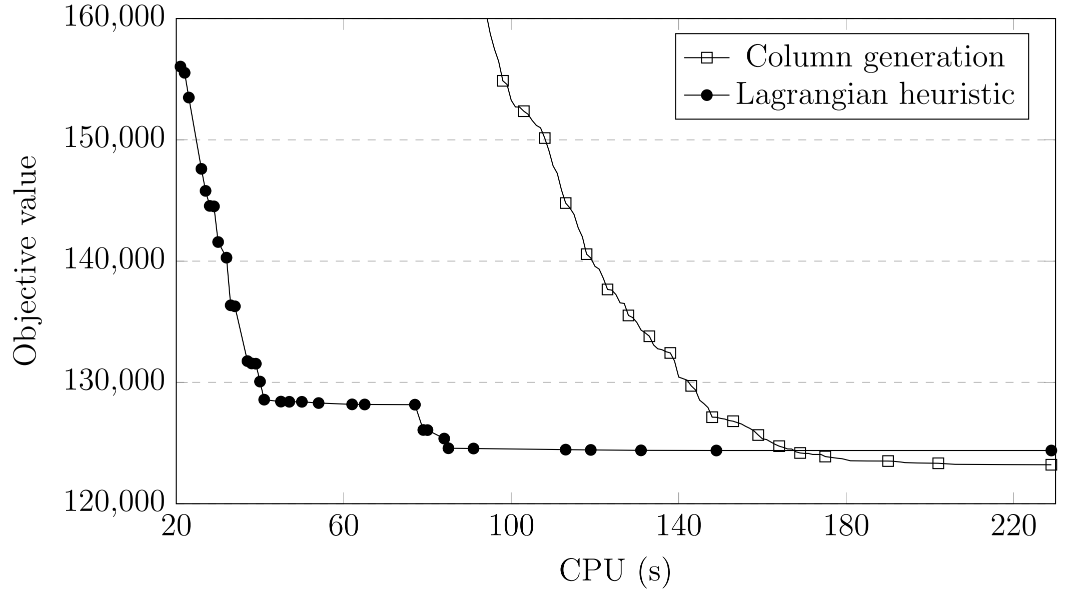

Figure 2: Plot of the average objective value (of best found feasible solutions) versus the computation time for instances of 700 trips in class M with no back-up vehicle at the depot.

6 Vehicle Rescheduling with Retiming

The major drawback of the VRSP as formulated in Li et al. is that the rescheduling options are very limited. It is only possible to either perform a trip or cancel it altogether, whereas a small delay is likely to be preferred over a cancellation. In this chapter, we introduce the vehicle rescheduling problem with *retiming* (VRSPRT). Retiming a trip means that it is delayed with a couple of minutes, with the aim to avoid cancellations. We first discuss previous research on considering delays in rescheduling problems. Next, we propose a new method for incorporating delays that results in a formulation that is very similar to the original VRSP, such that the same heuristics can be applied. Subsequently, we discuss a dynamic approach that can be used to solve large instances. Finally, the results of two experiments are presented, illustrating the added value of retiming.

6.1 Literature Review

In the context of road-based vehicle rescheduling, Huisman et al. (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 arcs (i, j). Li, Borenstein, and Mirchandani (2008) circumvent this problem by modeling delays explicitly in a multi-commodity network flow problem with time windows. 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 takes 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 and 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, Huisman, and Desaulniers (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, Potthoff, Huisman, and Kroon (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.

As is clear, many different methods for including delays already exist in the literature. However, to the best of our knowledge, no method yet exists that efficiently models all delay options and uses a flow formulation. The reason a flow formulation is preferred is that in that case both the Lagrangian heuristic from Chapter 3 and the column generation heuristic from Chapter 5 can be applied. Consequently, we will now propose a new method for incorporating delays in rescheduling problems exhibiting the desired features.

6.2 Mathematical Formulation and Methodology

The main idea for incorporating the possibility of delays into a mathematical formulation is to not model delays *explicitly*, by adding variables for the starting times and constraints these must adhere to, but *implicitly*, by adding vertices and arcs to the recovery network. To model the delays, we associate with every trip i served from (pseudo-)depot d a set T_i^d of possible starting times. The possible starting times of trip j are determined by the possible starting times of all possible predecessors i and the corresponding travel times. To limit the size of T_i^d, time can be discretized and a maximum delay D_i can be specified for each trip. Recall that the original recovery for pseudo-depot d is given by $G^d = (V^d,\ A^d)$, where $V_d = N^d \cup \{d,\ t\}$ and $A^d = E^d \cup (d \times N^d) \cup (N^d \times t)$. This recovery network can account for delays by splitting vertex i into a different vertex for each starting time in T_i^d (similar to the copies of tasks in Veelenturf et al. (2012)). Arcs that result in these delayed starting times and arcs from the new vertices to compatible trips are added to A^d.[3] Note that theoretically it is possible that the recovery network now contains cycles. However, this can only occur if the maximum allowed delay is larger than the minimum trip duration (in that case a vehicle could perform say trip 5 after trip 6), a situation that is disregarded in this thesis.

An example of an extended recovery network is presented in Figure 3. The solid part is the original network 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 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 2* after trip 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.

If retiming is accounted for by expanding the recovery network, the VRSPRT can be formulated

[3] As the starting time of a trip can be regarded as a resource, an interesting application of this methodology would be to solve the resource-constrained shortest path problem by solving the shortest path problem on the corresponding expanded graph.

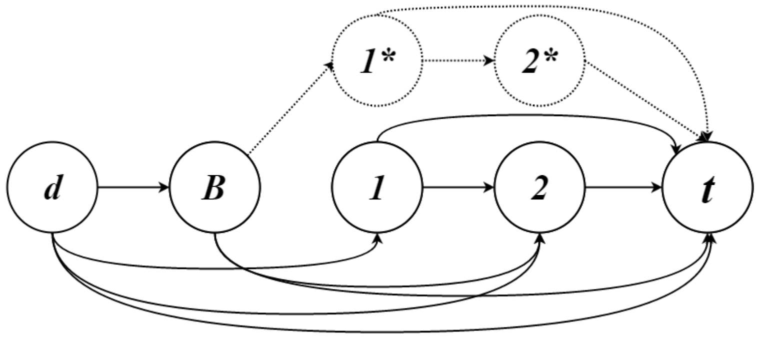

Figure 3: A recovery network extended with delay options. The dotted part of the network represents the options that become possible by introducing delays. d and t denote the starting and ending depot respectively, B denotes the back-up trip and a * indicates a trip with delay.

very similar to the VRSP. Let N^+ denote the set of all trips with all possible starting times and let $N^d(k)$ denote the set of vertices in N^+ that correspond to trip k in G^d (e.g. in Figure 3 $N^+ = \{B, 1, 1^*, 2, 2^*\}$ and $N^d(1) = \{1, 1^*\}$). The formulation of the VRSPRT is then given by:

$$\begin{aligned}
\min \quad & \sum_{d\in D}\sum_{(i,j)\in A^d} c_{ij}^d y_{ij}^d + \sum_{i\in N} C_i z_i & & & (6.1)\\
\text{s.t.} \quad & \sum_{\{j:(d,j)\in A^d\}} y_{dj}^d & \leq W \quad & d = s, & (6.2)\\
& \sum_{\{j:(d,j)\in A^d\}} y_{dj}^d & = 1 \quad & \forall d \in D - \{s\}, & (6.3)\\
& \sum_{d\in D}\sum_{i\in N^d(k)}\sum_{\{j:(i,j)\in A^d\}} y_{ij}^d + z_k & = 1 \quad & \forall k \in N, & (6.4)\\
& \sum_{\{j:(i,j)\in A^d\}} y_{ij}^d - \sum_{\{j:(j,i)\in A^d\}} y_{ji}^d & = 0 \quad & \forall i \in N^+, \forall d \in D, & (6.5)\\
& y_{ij}^d, z_i & \in \{0,1\} \quad & \forall d \in D, \forall (i,\ j) \in A^d, \forall i \in N. & (6.6)
\end{aligned}$$

As before, the objective is to minimize the sum of the operation costs and cancellation costs. Additional to reassignment costs, delay costs can now also be included in the arc costs c_{ij}^d. Constraints (6.2) and (6.3) assure that at most W vehicles depart from the depot and exactly 1 from every pseudo-depot. Constraints (6.4) and (6.5) differ compared to the VRSP, but fulfill the same function. Constraints (6.4) guarantee that every trip is either executed or cancelled and now include an extra summation, as there is no longer a one-to-one correspondence between trips and vertices. Constraints (6.5) are flow conservation constraints and now hold for all $i \in N^+$, since flow must be conserved in every vertex.

As can be seen, the structure of the given formulation for the VRSPRT is the same as the formulation of the VRSP, with the only difference being that a trip is now represented by a group of nodes, of which at most one is covered by a vehicle. Moreover, if constraints (6.4) are relaxed and incorporated into the objective function, the resulting Lagrangian problem can again be decomposed into a trivial sub-problem for every trip, a shortest path problem for every pseudo-depot (now in the expanded graph) and the K-shortest paths problem for the depot (also in the expanded graph). Consequently, both heuristics designed for the VRSP can be used to solve the VRSPRT.

Since the column generation approach has longer computation times and the difference between solution quality is not large, only the Lagrangian heuristic is used in the remainder of this thesis.

The primal heuristic that transforms the Lagrangian solution into a feasible solution needs small adjustments. In the first step of the heuristic, redundant coverings are removed. As there are multiple options for performing a trip, the trips after a removed trip can possibly be performed with less delay. Consider the recovery network from Figure 3 and assume the back-up trip needs to be removed from the path $\{d, B, 1^*, 2^*, t\}$ As it is no longer necessary to delay trips 1 and 2, the result is the path $\{d, 1, 2, t\}$. In the second step of the heuristic, uncovered trips are inserted in the existing paths. Before, to check whether an insertion is feasible it sufficed to check whether the trip is compatible with respect to its predecessor and its successor. However, now it is possible that an insertion only is feasible if trips after the inserted trip are delayed. For example, assume that trip 1 needs to be inserted in the path $\{d, B, 2, t\}$. This insertion is feasible only if trip 2 is delayed, hence the result is $\{d, B, 1^*, 2^*, t\}$.

6.3 Dynamic Neighborhood Exploration

Even with discretized time and a maximum delay, the number of vertices and arcs in the recovery network with delays can be much larger than in the corresponding recovery network without delays. For instances of 700 trips in class M, when we permit a maximum delay of only 5 minutes and discretize time to minutes, it is not rare for the number of arcs and vertices to more than triple. As a result, the pre-processing phase of the algorithm, where the recovery network is constructed, can take up to 15 minutes with delays in comparison with a mere 60 to 90 seconds without. Moreover, due to the increased problem size the Lagrangian heuristic needs more time to find good solutions. Out of 5 test instances, where the Lagrangian heuristic was terminated after 5 minutes, only in one case the number of cancellations was reduced compared to the VRSP. In two of the cases the number of cancellations even increased. On average, the heuristic could only perform 9 iterations before the time limit exceeded. As is clear, the size of the network needs to be reduced in order to realize an applicable rescheduling method.

To this end, we apply the ideas of Potthoff et al. (2010) to the VRSPRT. Potthoff et al. (2010) 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, trips in the neighborhood of *difficult* trips (trips that are difficult to perform) can be added to the subset.

More specifically, we maintain a set R of trips that are allowed to be retimed. Initially, only the cut trip is in this set. In other words, we start by solving the VRSP. If the solution of the Lagrangian heuristic does not include any cancellations, the algorithm is terminated. Otherwise, trips that lie 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 as difficult trips, but all trips i for which it holds that (i) $z_i = 1$ in the Lagrangian solution (implying that this trips has a high Lagrangian multiplier, so it is difficult to perform) or (ii) is not performed in the Lagrangian solution (before the insertion heuristic). Furthermore, we consider the iteration that lead to the best feasible solution as well as the final iteration of the Lagrangian heuristic to find the difficult trips. This broader definition enables us to extract more information from the Lagrangian optimization.

As for the neighborhood of a trip, let st_i and et_i denote the original starting and ending time and sp_i and ep_i the starting and ending relief point of trip i, respectively. Moreover, let $\tau_{u,v}$ denote the travel time between relief point u and v. Inspired by Potthoff et al. (2010), we define that trip j lies in the neighborhood of trip i if it satisfies one of the following three conditions:

1. $j = i$,
2. $s_j \in [s_i - m, s_i + m]$ and $\tau_{sp_i, sp_j} < r$,
3. $s_j \in [e_i - m, e_i + m]$ and $\tau_{ep_i, sp_j} < r$.

The first condition is trivial; if a trip is hard to cover we want to be able to delay it in the next iteration. For the second and third condition, the intuition is that it is more likely that a trip is performed if scheduling flexibility around its starting time and location and its ending time and location is increased. In this thesis, m is set to 30 minutes and r to 10 minutes, as these values lead to neighborhoods of decent sizes (up to 20 trips).

In case all trips in the neighborhood of difficult trips are already in R, there would be no trips to add to R, causing the algorithm to get stuck. To prevent this, in such a case trips that lie in the neighborhood of a trip in R, are added to R, with a maximum of 10 trips. Lastly, to speed up the algorithm, the final Lagrangian multipliers of the previous iteration are used as initial values in the next iteration. The algorithm can be terminated if all trips are covered, if no improvement is found in a certain number of iterations or if a time limit is exceeded.

6.4 Added Value of Retiming

In this section, we investigate whether the introduction of retiming can lead to less disruptive solutions, especially regarding the number of cancellations. For this purpose, we perform two experiments for the VRSPRT, where the original VRSP serves as a benchmark. In the first experiment instances of 300 trips are solved using the regular Lagrangian heuristic and in the second experiment instances of 700 trips are solved using the dynamic neighborhood exploration.

As we observed that the number of cancellations after solving the VRSP is the highest in instances of class M with no back-up vehicle available at the depot and where the breakdown occurs 80% into the cut trip, this configuration is used in the experiments. Furthermore, C is taken to be 2000 and P is taken to be 500, the combination that lead to the fewest cancellations in the experiments for the VRSP. Starting times of trips are rounded to minutes to limit the number of retiming options (so if the maximum allowed delay is 5 minutes, the trip can be delayed with 0, 1,... or 5 minutes).

In the first experiment, we solve 10 instances of 300 trips with different allowed delays and delay costs to analyse to what extent delaying trips is an effective measure for reducing the number of cancelled trips. The iteration limit is set at 350 and the time limit is set at 60 seconds.

The results of the first experiment are presented in Table 5 and in Figure 4. If the allowed delay is 0, the VRSPRT reduces to the original VRSP, so the added value of retiming can easily be deduced. As can be seen, when the allowed delay increases, the number of cancelled trips decreases. With an allowed delay of 10 minutes or more, the average number of cancelled trips is halved compared to the situation without retiming. If the delay costs are 60 per minute, this reduction can be achieved by, on average, delaying only one trip with less than 7 minutes. If the delay costs are 20 or 40 per minute the number of cancellations is slightly less, whereas the number of delayed trips is 0.5 more. It seems that with low delay costs, delays are not only introduced to avoid cancellations, but also to create cheaper connections. All in all, it can be concluded that retiming is an effective method that leads to solutions preferred to those of the VRSP.

Table 5: Results of the VRSPRT with different delay costs and allowed delays. An allowed delay of 0 minutes corresponds to the original VRSP.

Delay costs per min.	Max. allowed delay (min.)	Cancelled trips	Delayed trips	Average delay	Maximum delay	Total costs	Gap (%)	CPU (s)
20	0	1.6	-	-	-	62,301	1.9	29.6
	5	1.0	0.9	2.8	4.0	61,524	1.4	39.1
	10	0.8	1.3	5.1	7.0	60,818	0.7	43.9
	15	0.7	1.5	6.3	11.0	60,233	1.9	48.3
40	0	1.6	-	-	-	62,301	1.9	29.3
	5	1.1	0.6	2.4	4.0	61,581	1.4	40.5
	10	0.8	1.1	5.5	9.0	61,001	0.9	46.8
	15	0.7	1.4	5.7	9.0	61,269	2.0	47.5
60	0	1.6	-	-	-	62,301	1.9	29.6
	5	1.1	0.4	2.5	4.0	61,574	1.3	38.7
	10	0.8	0.8	6.2	9.0	61,324	1.2	49.1
	15	0.8	1.0	6.5	11.0	61,313	1.8	51.6

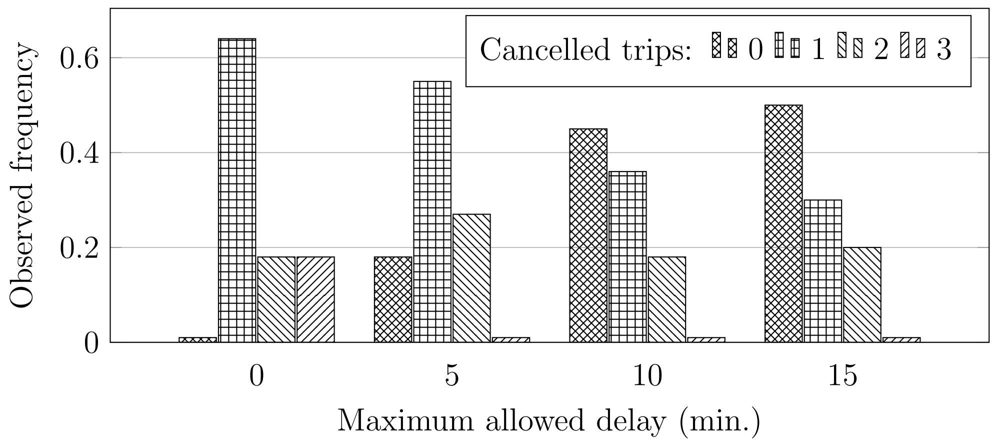

Figure 4: Histograms of the number of cancellations for the different maximum allowed delays, with delay costs of 40 per minute. An allowed delay of 0 minutes corresponds to the original VRSP.

In the second experiment, we solve 50 instances of 700 trips with the dynamic approach explained in the previous section. As a maximum allowed delay of 10 minutes and delay costs of 60 per minute performed the best in the first experiment, these values are used in the second experiment. 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 terminating the Lagrangian heuristic after 60 seconds if the optimality gap is smaller than 5 and after 120 seconds otherwise. The resulting number of cancellations serves as a benchmark for the dynamic approach. Every next iteration of the algorithm has a time limit of 30 seconds. The whole process is terminated if the number of cancellations is reduced compared to the benchmark or if 10 iterations are performed.

Of the 50 solved instances, the number of cancellations was reduced without retiming in 8 instances, despite the long running time for the initial solution. Because the algorithm terminates if the number of cancellations is reduced, these instances cannot provide us with any information regarding the performance of the dynamic neighborhood exploration. Moreover, in one instance no trips were cancelled in the initial solution. Of the other 41 instances, cancellations were reduced using retiming in 23 cases. The average results over the 41 instances are presented in Table 6.

Additional results of the 23 cases where cancellations were reduced are presented in Table 7.

If the benchmark is 1 cancelled trip (so the solution of the VRSP contains 1 cancelled trip), the dynamic neighborhood exploration reduces this to 0 by retiming in one third of the cases. To achieve this, on average 1 trip needs to be delayed with 3.5 minutes. If the initial number of cancellations is larger than 1, reductions are achieved in 80 percent of the cases. On the other hand, this requires on average 2 trips to be delayed with 4 to 5 minutes. As expected, the algorithm has a much harder time reducing cancellations if the benchmark is 1, taking on average 5 iterations and 3 extra minutes, compared to 2 or 1 iterations and around 1.5 extra minute if the benchmark is 2 or more. Surprisingly, this does not lead to a much larger set R (the set of trips for which retiming is allowed). Perhaps larger values of m and r would be more suitable if the benchmark is 1, such that the size of R increases more rapidly. Still, it can be concluded that the dynamic neighborhood exploration achieves the desirable results. The additional computation times are moderate and the number of cancellations decreases on average from 1.8 to 1.1.

Table 6: Results of the dynamic neighborhood exploration for solving large instances of the VRSPRT. Benchmark refers to the number of cancellations in the solution of the VRSP and CT is the average number of cancelled trips.

Benchmark (proportion)	CT no retiming	CT with retiming	Reduced CT (%)	Total costs no retiming	Total costs with retiming
1 (51%)	1.0	0.7	33%	118,156	117,688
2 (27%)	2.0	1.2	82%	121,884	119,872
3 or more (22%)	3.7	2.1	78%	133,993	130,793

Table 7: Results of the dynamic neighborhood exploration on instances where cancellations were reduced using retiming. Benchmark refers to the number of cancellations in the solution of the VRSP, extra CPU is the average additional computation time required to reduce cancellations and $|R|$ is the average number of trips for which retiming is allowed when the neighborhood exploration terminates.

Benchmark (proportion)	Delayed trips	Average delay (min.)	Iterations	Extra CPU (s)	$\|R\|$
1 (30%)	1.1	3.4	4.9	186.8	85.6
2 (39%)	2.0	4.2	1.8	92.6	62.0
3 or more (30%)	2.1	4.9	1.0	77.3	94.7

7 Conclusion

This thesis examined the road-based vehicle rescheduling problem (VRSP); given a series of trips, a current schedule and a vehicle breakdown, determine a new feasible schedule which minimizes operation, trip cancellation and schedule disruption costs. Despite of increased attention for disruption management, the VRSP is relatively unexplored in two ways. First, it concerns road-based rescheduling (aircraft and railway rescheduling problems are better researched) and second, it concerns disruptions caused by breakdowns (most papers consider disruptions caused by delays).

In the first part of the thesis, the results of Li et al. (2009) were replicated. Li et al. introduced the VRSP and proposed solving it using a Lagrangian heuristic. The results of Li et al. are confirmed: even for instances of 700 trips the heuristic finds high quality solutions within 90 seconds. Compared to an ad hoc rescheduling approach, the number of cancellations is reduced.

In the second part of the thesis, we built further on Li et al. (2009). First, the Lagrangian heuristic of Li et al. was compared to a column generation based heuristic. The results show that column generation leads to better (often even optimal solutions), but the computation times are larger compared to the Lagrangian heuristic. Additional research might be able to speed up the column generation heuristic.

Second, we introduced the vehicle rescheduling problem with retiming (VRSPRT). By retiming trips (delaying trips with a couple of minutes) scheduling flexibility is increased, which helps avoiding cancellations. To incorporate the delays, the underlying recovery network is expanded to include all relevant retiming possibilities. The advantage of this method, that was not found in the literature, is that it leads to a formulation very similar to the original VRSP, such that the same heuristics can be applied. The results indicate retiming can half the number of cancellations compared to the VRSP. At the same time, the incurred delays are very moderate (on average one trip was delayed with 6.5 minutes). This implies that the VRSPRT leads to less disrupted schedules compared to the VRSP, increasing the satisfaction of passengers.

As the size of the recovery network can become very large by introducing retiming, we proposed a dynamic neighborhood exploration heuristic for large instances. In this heuristic, only a subset of all trips is allowed to be delayed and relevant trips are added to this subset iteratively if trips are still cancelled. The results indicate that the heuristic performs well, especially when the number of cancellations without retiming is larger than 1. On average, the number of cancellations is reduced from 1.8 to 1.1 with 1 or 2 trips being cancelled with less than 5 minutes. Computation times do increase, but not excessively. Additional refinements to the neighborhood definition can possibly further improve the heuristic.

Wrapping up, considering we (i) replicated the results of Li et al., (ii) compared the performance of their Lagrangian heuristic to a column generation based approach and (iii) introduced the VRSP with retiming, including a dynamic heuristic for large instances, it can be concluded that this thesis succeeded in its aim of contributing to the VRSP literature. It is our hope that this thesis revitalizes interests in the VRSP. Apart from refining the methods discussed in this thesis, directions for further research include the application of time-space networks and the integration of vehicle rescheduling with crew rescheduling.

8 References

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., & Vance, P. H. (1998). Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, *46*(3), 316–329.

Bramel, J., & Simchi-Levi, D. (1997). On the Effectiveness of Set Covering Formulations for the Vehicle Routing Problem with Time Windows. *Operations Research*, *45*(2), 295–301.

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.

Cheng, C., Kumar, S. P., & Garcia-Luna-Aceves, J. (1989). A Distributed Algorithm for Finding K Disjoint Paths of Minimum Total Length.

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

Dantzig, G. B., & Wolfe, P. (1960). Decomposition Principle for Linear Programs. *Operations Research*, *8*(1), 101–111.

Desaulniers, G., Desrosiers, J., & Solomon, M. M. (2002). *Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems*. Springer.

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

Fisher, M. L. (2004). The Lagrangian Relaxation Method for solving Integer Programming Problems. *Management Science*, *50*(12), 1861–1871.

Freling, R., Wagelmans, A. P., & Paixão, J. M. P. (2001). Models and Algorithms for Single-Depot Vehicle Scheduling. *Transportation Science*, *35*(2), 165–180.

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., Jans, R., Peeters, M., & Wagelmans, A. P. (2005). *Combining Column Generation and Lagrangian Relaxation*. Springer.

Huisman, D., & Wagelmans, A. P. (2006). A Solution Approach for Dynamic Vehicle and Crew Scheduling. *European Journal of Operational Research*, *172*(2), 453–471.

Irnich, S., Desaulniers, G., et al. (2005). Shortest Path Problems with Resource Constraints. *Column Generation*, *6730*, 33–65.

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., Mellouli, T., & Suhl, L. (2006). A Time–Space Network Based Exact Optimization Model for Multi-Depot Bus Scheduling. *European Journal of Operational Research*, *175*(3), 1616–1627.

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.

Pepin, A.-S., Desaulniers, G., Hertz, A., & Huisman, D. (2009). A Comparison of Five Heuristics for the Multiple Depot Vehicle Scheduling Problem. *Journal of Scheduling*, *12*(1), 17–30.

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

Steinzen, I., Gintner, V., Suhl, L., & Kliewer, N. (2010). A Time-Space Network Approach for the Integrated Vehicle-and Crew-Scheduling Problem with Multiple Depots. *Transportation Science*, *44*(3), 367–382.

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

Suurballe, J. (1974). Disjoint Paths in a Network. *Networks*, *4*(2), 125–145.

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.

Yu, G., & Qi, X. (2004). *Disruption Management: Framework, Models and Applications.* World Scientific.

A Shortest path algorithms

Let $G = (V, A)$ be a directed graph, where V is the set of vertices and $A \subseteq V \times V$ the set of arcs. Let $l : A \to \mathbb{R}$ be a weighting function on the arcs. $l(u, v)$ is referred to as the length of arc (u, v). A path π is a sequence of vertices $\pi_1 \cdot \pi_2 \cdot \ldots \cdot \pi_n$ such that all $\pi_i \in V$ and all $(\pi_i, \pi_{i+1}) \in A$. The length of a path π is $L(\pi) = \sum_{i=1}^{n-1} l(\pi_i, \pi_{i+1})$.

The shortest path algorithm used in this thesis requires that the vertices are sorted in a topological order, meaning that for every arc (u, v), u must comes before v in the order. In this algorithm, $\pi(v)$ denotes the shortest path from the source s to vertex v.

Algorithm 1 Calculating the shortest paths from a vertex s to all other vertices in a topologically sorted graph

```
Input: A graph $G = (V, A)$ and a function $l : A \to \mathbb{R}$. $V$ must be topologically sorted and
  $s$ is the first element in this order.
  Initialization. $\pi(s) \leftarrow s$, $L(\pi(v)) \leftarrow 0$ and all other $L(\pi(v)) \leftarrow \infty$.
  for all $v \in V$ do
      for all $(v, w) \in A$ do
          if $L(\pi(v)) + l(v, w) < L(\pi(w))$ then
              $\pi(w) \leftarrow \pi(v) \cdot w$
```

For the K-shortest paths algorithm, some additional notation is used. Let $\pi^k(v)$ denote the kth shortest path from s to v and let $C(v)$ denote a set of candidate paths for the next shortest path from s to v. The algorithm for computing the K shortest paths is as follows.

Note that also other stopping conditions can be used. For instance, in this thesis the next shortest path is calculated until the length of the next shortest path is positive.

Algorithm 2 Calculating the K shortest paths from a source s to a sink t

```
Input: A graph $G = (V, A)$ and a function $l : A \to \mathbb{R}$.
  Initialization. Calculate $\pi^1(v)$ for all $v$ with a suitable one-to-all shortest path algorithm.
  $k \leftarrow 1$
  repeat
      $k \leftarrow k + 1$
      $\pi^k(s) \leftarrow \textsc{NextPath}(t, k)$
  until $k = K$

  function $\textsc{NextPath}(v, k)$
      if $k = 2$ then
          Initialize $C(v)$
          $C(v) \leftarrow \{\pi^1(u) \cdot v : (u, v) \in A \text{ and } \pi^1(u) \cdot v \neq \pi^1(v)\}$
      if $k \neq 2$ or $v \neq s$ then
          Let $u$ and $k'$ be the vertex and index such that $\pi^{k-1}(v) = \pi^{k'}(u) \cdot v$
          if $\pi^{k'+1}(u)$ not already computed then $\pi^{k'+1}(u) \leftarrow \textsc{NextPath}(u, k' + 1)$
          if $\pi^{k'+1}(u)$ exists then $C(v) \leftarrow C(v) \cup \pi^{k'+1}(u) \cdot v$
      if $C(v) \neq \emptyset$ then
          $\pi^k(v) \leftarrow \underset{\pi(v) \in C(v)}{\text{argmin}}\, L(\pi(v))$
          $C(v) \leftarrow C(v) \setminus \pi^k(v)$
      else
          $\pi^k(v)$ does not exist
      return $\pi^k(v)$
```