A Resource Aggregation Based Label-Correcting Algorithm For The Resource Constrained Shortest Path Problem In A Column Generation Context

MASTER THESIS ECONOMETRICS & MANAGEMENT SCIENCE

Specialization Operations Research & Quantitative Logistics

Author:
M. S. Timmermans
Studentnumber:
458353

Supervisor:
Dr. T.A.B. Dollevoet

Co-reader:
T. Breugem MSc.

Erasmus University of Rotterdam
Erasmus School of Economics

October 21, 2018

Abstract

In this thesis we investigate the potential benefit of applying resource aggregation strategies in a column generation algorithm used to solve a set-covering problem (SCP) with additional constraints. The corresponding pricing problem is mathematically formulated as a resource constrained shortest path problem (SPPRC) and solved by a dynamic programming approach. The additional resource constraints are incorporated in the problem to take the preferences of the employees into consideration such that a fair distribution of the workload is obtained. We exploit the idea of resource correlation to develop aggregation strategies instead of considering the individual resources in the SPPRC.

We develop a framework to solve the pricing problem arising from the application of column generation. The framework consists of a sequence of dynamic programming algorithms based on a generic version of a label-correcting algorithm for solving a shortest path problem. The basic algorithm is adapted such that we can incorporate each of the developed resource aggregation strategies. The proposed resource aggregation strategies are tested on three simulated data instances with a varying degree of correlation, deducted from a part of the actual rail transit network in The Netherlands.

The computational results show that some of the developed aggregation strategies perform well for the dataset with medium correlated resources. A reduction of the computation time up to approximately 36% is achieved using a well-chosen parameter setting. Results show that the same parameter setting results in a minimal fairness error according to the used fairness criterion. The use of an aggregation strategy with the well-chosen parameter settings in a column generation context may result in a more efficient approach to incorporate the preferences of employees and increase their happiness and welfare.

Contents

1 Introduction

Technology is all around us and is developing continuously. This also causes the world around us to change all the time. Scientist believe that because of the recent technological trends such as machine learning techniques and data-driven decision making, very soon robots will take over many basic activities like as driving a car. These developments can be very beneficial for numerous companies. In contrast to regular personnel, robots basically do not have any preferences and there are no labor regulations an employer should cope with.

However, the technological development has not come this far yet and currently human beings are necessary for most of the labor. This also means that the focus of the organization should not be solely on the economic aspect, but also on the happiness and welfare of the employees. We have seen that in case the welfare aspect is neglected by the organization, dissatisfaction occurs among the employees which results in poor execution of tasks or in a worst-case scenario employees going on strike (Abbink et al., 2005). However, especially in case of scheduling crew in large organizations, the main focus often lies on the economic aspect. That is minimizing the total sum of costs such that all necessary tasks can be completed and labor regulations are satisfied. The preferences of employees are rarely taken into account despite the fact that multiple studies have shown a positive relation between employee happiness and the quality of work they deliver (Alexander and Ruderman, 1987; Colquitt et al., 2001).

In crew scheduling problems, the objective is to allocate employees to (different) working shifts such that enough personnel is scheduled to perform all tasks. However, the tasks could differ in labor intensity, working environment, etc. Each task can be preferred differently by the employees. For example, in Abbink et al. (2005) it is stated that the personnel of the Netherlands Railways (NS) is more frequently confronted with aggression when performing a duty in the Randstad [1] and therefore they prefer duties outside of the Randstad. They also tend to prefer working on a long distance train because, it is typically used in duties with relatively less intermediate stops and larger travel distances.

The crew scheduling problem can be mathematically formulated as a set-covering problem (SCP). The SCP is one of the oldest and most studied topics in the field of combinatorial optimization. A large variety of well-know real world problems (such as scheduling, routing, stock cutting and facility location-allocation) can be formulated mathematically using the SCP formulation (Balas, 1982). Crew scheduling can be seen as one of the most important and commonly used applications formulated as SCP.

Relatively much literature is published in which methods are presented to solve the SCP effi-

[1] The Randstad is an area in the western part of the Netherlands. It consists of a large rural area surrounded by the four largest cities of the country.

ciently. However, just a few authors considered the SCP together with additional constraints in order to incorporate the preferences of the employees. This is mainly because solving the SCP becomes in general computationally more difficult when these additional constraints are taken into account. A common (and efficient) method to solve the SCP is applying a column generation approach. The column generation approach considers a master problem and the corresponding subproblem which are solved sequentially. The subproblem can be formulated as a shortest path problem with resource constraints (SPPRC) (Desrochers and Soumis, 1988).

The resources considered in a SPPRC can be of various types such as time, fuel consumption or distance, but one may also consider non-material resources related to the duty preferences of employees. Often resources have similarities and are correlated to some extent. The previous mentioned preferences of the personnel at NS are correlated as well. Namely, long distance trains will be used more often in rural areas as the distances between cities/villages are longer and the NS personnel is also less likely to be confronted with aggression in these rural areas. We can take into account the preferences in order to increase the fairness of the solution, however increasing the number of resources typically makes it more difficult to solve the subproblem as mentioned before.

In this thesis, we investigate the possibility of solving the pricing problem more efficiently by considering the correlation between resources. The idea is to (partially) aggregate the resources using a specific technique instead of taking the resources into account individually. We consider three different strategies for the aggregation of the resources. We develop a framework to solve the pricing problem which is embedded in the column generation algorithm. The framework is a sequence of dynamic programming algorithms which will be executed in a specific order depending on the status of the program. The pricing problem framework is based on a generic version of a label-correcting algorithm for solving a shortest path problem. The basic algorithm is adapted such that we can incorporate (aggregated) resources and solve the SPPRC. The framework is evaluated on a simulated dataset. The dataset is based on the actual transit network in the region of Amsterdam, however some adaptations are made in order to obtain the desired characteristics. In this way, it is possible to test the aggregation strategies for different degrees of correlation between the resources.

The remainder of this thesis is organized as follows: in Chapter 2 a formal description of the problem is given. Chapter 3 presents a brief overview of the existing literature on this topic and the relevance of this research. In Chapter 4, we give the methodology used to solve the problem and we also provide a brief explanation of algorithms incorporated in the pricing problem framework. The different features of the simulated dataset are presented in Chapter 5. In Chapter 6, the computational results for different data instances are given and the effectiveness of the aggregation strategies is evaluated. Finally, the research is discussed and concluded in Chapter 7.

2 Problem Description

In this thesis, we consider the problem of scheduling crew for the public transport by train. The aim is to schedule the crew such that we obtain a fair solution with respect to the preferences of the crew. In this chapter, we will provide a brief explanation of the problem from both a practical and mathematical point of view. In Section 2.1, we introduce the problem considered in this thesis by defining some terminology and concepts such as the resources (Section 2.1.1) and the definition of a feasible shift (Section 2.1.2). The notation and the mathematical formulation of the problem will be given in Section 2.2. Here, we also elaborate on the adaptation of the standard SCP formulation by introducing additional constraints (Section 2.2.3).

2.1 Terminology

We need to clearly introduce the problem considered in this thesis and explain some concepts and terminology. First of all, the basic idea of the problem considered in this thesis, is the problem of scheduling crew such that all the tasks are performed. In our case the crew consists of train drivers and conductors and a task represents a trip: a train traversing from station i to station j. A shift is defined as a set of trips which are performed in consecutive order. We consider a rail transit network consisting of 5 stations which is based on (a part of) the actual transit network in the Netherlands. We consider n trips in total which should be performed by the corresponding train type for each trip respectively. The trips with larger travel distances are in general performed by long distance trains which are called 'Intercity' trains in the Netherlands. The regional trains which perform most of the short distance trips (often in urban area) are called 'Sprinter' trains. In the remainder of this research, the long distance trains and regional trains are referred to as Intercity and Sprinter trains respectively. In this research, we focus on a particular aspect of the mathematical model used to solve the crew scheduling problem. In order to perform this research it is not of any importance to consider a detailed and comprehensive test case. Therefore, we do not distinguish between different types of personnel or a varying amount of required personnel per trip for example.

2.1.1 Resources

As mentioned before, the basic idea of the problem considered in this thesis, is the problem of scheduling crew such that all the tasks are performed. Furthermore, we also aim to find a solution for the CSP which satisfies certain fairness requirements. The fairness of a certain solution is measured by the allocation of the resources $k \in \{1, 2, ..., K\}$. Each trip j will have certain characteristics and we model these characteristics in our problem formulation as resources denoted by r_j^k. Fuel is a typical example of a resource in case of a vehicle routing problem. The vehicle is restricted by the total fuel capacity of the fuel tank. Each trip accounts for a certain fuel consumption and the vehicle cannot perform a shift which requires more fuel than its capacity. In our case we consider 4 trip characteristics which are related to the preferences of the NS personnel:

$r_j^1 \in \{I, S\}$ - The type of train used to perform the trip (Intercity (I) or a Sprinter (S) train),

$r_j^2 \in \mathbb{N}$ - The number of stops during a trip,

$r_j^3 \in \{HR, LR\}$ - If there is a high or low risk (HR or LR) of aggressive passengers for a trip,

$r_j^4 \in \{DD, SD\}$ - If the trip is performed by a double-decker (DD) or single-decker (SD) train.

Different from the fuel consumption example, we do not have a specific capacity for any of the resources listed above. Instead, we consider a weighted fraction of the resource contribution in a shift to which we refer to as the resource consumption.

2.1.2 Feasible shift

The feasibility of a shift is assessed by a number of restrictions. We distinguish between the general restrictions which ensure that each shift has the correct structural elements and the restrictions which ensure a minimum fairness requirement. The feasibility of a shift will be explained with the help of Figure 2.2. The figure consists of a number of blocks where each of the blocks represents a trip. Let us for example consider the block shown in Figure 2.1. The width of a block depends on the trip duration. As explained earlier, a trip is defined as a train traversing from one station to another. The two stations which are connected by a particular trip are displayed in the center of a block. The characters each stand for a specific station. The train starts at station D and arrives at station B in 20 minutes. The departure and arrival time are displayed just below the block near the left and right corner respectively. The characteristics which are applicable for a trip are indicated in the corners of the block. Each of the corners represent a resource. The resources (see Section 2.1.1) are assigned clockwise to the corners starting at the upper left corner. It can be observed that the trip shown in Figure 2.1, is performed by a Sprinter and during the trip the train should make 6 intermediate stops. Furthermore, there is a high risk of encountering aggressive passengers for the NS personnel and it is a single-decker train.

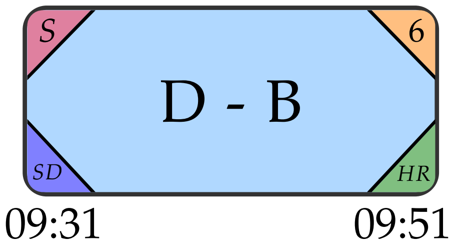

Figure 2.1: An example of a block representing a particular trip between two stations D and B. Furthermore, the departure and arrival time are displayed just below the block and each of the corners represent a characteristic of the trip.

As the interpretation of a block is now clearly defined, let us consider Figure 2.2 in order to explain the definition of resource feasibility. First of all, the figure shows an example of a feasible shift by the sequence of connected blue colored blocks. Furthermore, white colored blocks are connected by dashed lines to show other configurations of the block sequences. These configurations result in infeasible shifts due to certain restrictions.

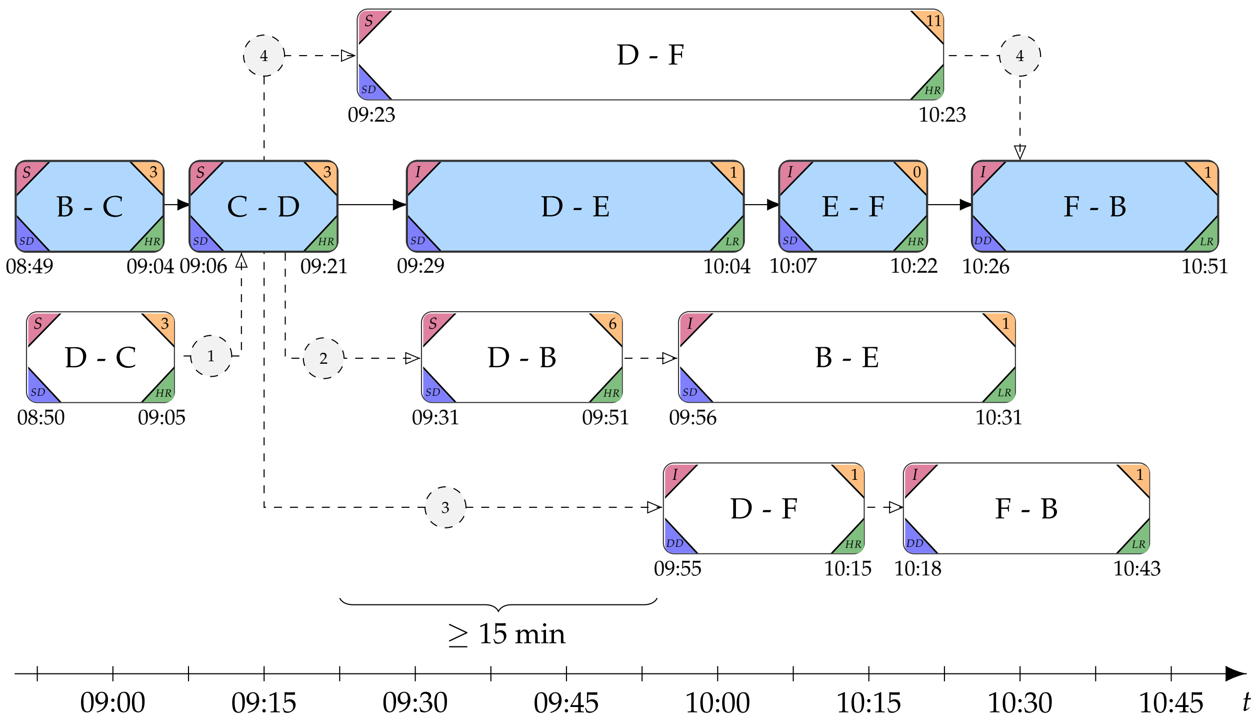

Figure 2.2: The block scheme is used to explain the definition of a feasible shift. The sequence of blue colored blocks represent a feasible shift. The other configurations of blocks connected by dashed lines represent infeasible shifts due to some restrictions.

We constructed four examples indicated by the dashed circles in order to provide an interpretation of the feasibility of a shift. The four examples are listed below.

1. A feasible shift should start at one of the depot stations. Only a few of the stations considered in the transit rail network are depot stations (see Section 5.1). In Figure 2.2, it can be observed that the feasible shift starts at station *B* which is a depot station. However, the adjusted configuration of the shift results in a shift starting at station *D*. The latter station is not considered to be a depot station and thus the adjusted configuration results in an infeasible shift.

2. Besides the requirement of starting a shift at a depot station, another important requirement is that an employee should end at the same station the shift starts at. In case of the feasible shift (the sequence of blue blocks shown in Figure 2.2), it can be observed that the shift starts and ends at station *B* whereas the shift indicated by the second dashed

line ends at station E. This also happens to be a depot station. However, the shift did not start at station E and therefore the shift is infeasible.

3 A shift is basically a sequence of trips tied together. Preferably, we have shifts with minimum time between two consecutive trips as the goal is to find the least cost combination of feasible shifts such that all trips are covered at least once. The time in between the trips have a significant contribution to the total cost of a shift. We added a restriction on the time in between two trips to be 15 minutes at maximum. We can observe that the feasible shift indeed satisfies this restriction. The third dashed line however results in an infeasible shift configuration because, there is more than 30 minutes in between two consecutive trips. Furthermore, the total duration of a shift should not exceed 8 hours.

4 We also set restrictions on the total resource consumption of a shift in order to meet the fairness requirement. The resources are examined by a weighted fraction of the resource consumption which should be smaller than some upper bound UB^k. The fourth dashed line shows a shift configuration in which two trips of the feasible shift are replaced by the trip $D - F$. Let us solely focus on the first resource r_j^1 (the type of train) in this example. The shift should satisfy the following restriction:

$$\frac{\sum_{j=1}^{n} r_j^1 T_j}{\sum_{j=1}^{n} T_j} \leq UB^1,$$

where n is the number of trips in the considered shift and T_j is the duration of trip j. With an upper bound $UB^1 = 0.5$, we find for the feasible shift:

$$\frac{T_{B-C} + T_{C-D}}{T_{B-C} + T_{C-D} + T_{D-E} + T_{E-F} + T_{F-B}} \leq 0.5,$$

while for the shift constructed by the fourth dashed line we find:

$$\frac{T_{B-C} + T_{C-D} + T_{D-F}}{T_{B-C} + T_{C-D} + T_{D-F} + T_{F-B}} > 0.5.$$

The feasible shift satisfies the resource restriction and is considered to be a fair shift if and only if it also satisfies the other three resource restrictions. The latter shift is unfair and therefore considered to be infeasible based on the first resource restriction.

2.1.3 Objective

The objective is to obtain a least cost solution consisting of shifts which are equally fair. The fairness is taken care of in the definition of a feasible shift. We want to construct a least cost set of feasible shifts such that all trips are covered at least once. The objective of the optimization would be minimizing the total cost for a fixed fairness criterion. That is, we set a fairness criterion and a shift can either be considered fair or not fair. The fairness criterion is based on the weighted resource averages of the instance and is explained in Section 6.2.5.

2.2 Crew Scheduling as a Set Covering Problem

The Crew Scheduling Problem (CSP) is formally described as follows: given a set of tasks and a group of available crew, determine a set of shifts such that all tasks are performed. The objective is to minimize the total costs corresponding to the performed shifts. This naturally leads to a set covering formulation (SCP) in which the columns a_j for all $j \in N$ of matrix A correspond to feasible shifts and the rows $i \in M$ correspond to tasks. A solution of the CSP then consists of a set of shifts (columns) such that all performed shifts together perform all tasks (rows) at least once.

2.2.1 Formal Description of the SCP

The SCP can formally be described by considering a set $S = \{1, 2, ..., m\}$. The data of the SCP consists out of finite sets $P_1, P_2, ..., P_n$ and positive numbers $c_1, c_2, ..., c_n$. Here each finite set P_j only contains elements of set S for all $j \in J$ where $J = \{1, 2, ..., n\}$. The positive numbers c_j represent the cost corresponding to finite set P_j. The objective is to find a minimum-cost selection J^* such that all elements in S are covered. We call subset $J^* \subseteq J$ a cover when $\bigcup\limits_{j \in J^*} P_j = S$. Note that in order to find a feasible solution, it must hold that $S = \bigcup\limits_{j=1}^{n} P_j$.

2.2.2 Mathematical Formulation of the SCP

In order to give a mathematical formulation of the SCP, we slightly reformulate the problem. Let A be a m x n matrix, where $a_{ij} \in \{0, 1\}$ for all $i \in M$ and $j \in N$ where $M = \{1, 2, ..., m\}$ and $N = \{1, 2, ..., n\}$. We say that a column $j \in N$ covers a row $i \in M$ if $a_{ij} = 1$. In relation to Section 2.2.1, we define $a_{ij} = 1$ if $i \in P_j$ and we cover a row $i \in M$ instead of an element of set S. Furthermore, we have an n-dimensional vector c where $c_j > 0$ for $j \in N$. The value c_j represents the cost of column j. The objective is to find a minimum-cost selection of columns such that all rows are covered at least once. A mathematical formulation of the SCP is given by

$$\min \quad \sum_{j \in N} c_j x_j \tag{2.1}$$

$$\text{s.t.} \quad \sum_{j \in N} a_{ij} x_j \geq 1 \qquad \forall i \in M \tag{2.2}$$

$$x_j \in \{0, 1\} \qquad \forall j \in N \tag{2.3}$$

where the decision variable x_j indicates whether column $j \in N$ is selected or not. That is

$$x_j = \begin{cases} 1 & \text{if column } j \in N \text{ is selected} \\ 0 & \text{otherwise.} \end{cases}$$

The first set of constraints (2.2) make sure that every row is covered by the selected columns at least once. Constraints (2.3) ensure that the decision variable is binary. Finally, the objective (2.1) minimizes the total sum of costs corresponding to the selected columns.

2.2.3 Additional Constraints of Modeling Resources

In the introduction of this chapter, it is explained how the CSP can be modeled as SCP and how the formulation should be interpreted. An important detail is that the columns a_j are considered to be *feasible* shifts. The shifts are feasible in the sense that they satisfy a set of rules and regulations depending on the application. The set of rules and regulations which are applicable in the case considered in this research are explained in Section 2.1.2. These constraints must hold for each feasible column of the problem. One of the advantages of a SCP is that the formulation implicitly takes care of the constraints concerning the feasibility of the shifts and it prevents the mathematical program to become highly complex. We introduce the set of feasible columns $\Omega = \{a_j : D^1 a_j \leq C^1, a_j \in \mathbb{B}^m\}$ [2] where a_j represents a shift in which trip i is included as $a_{ij} = 1$. D^1 is the coefficient matrix and C^1 denotes the bound on a particular aspect of the problem. It should be noted however, that it strongly depends on the way the problem is modeled whether the constraints are taken into account implicitly or as additional constraints incorporated in the classic SCP formulation (or both). For example, we can ensure that the duration of each shift does not exceed the maximum shift duration of 8 hours by the following constraint $T_j x_j \leq 8$ for all shifts $j \in N$ in which T_j denotes the duration of shift j. We can also achieve this by only considering shifts which satisfy this constraint by definition. The later is preferred as it is stricter and the resulting problem is less complicated.
Thereby, we can also have additional constraints which are not column specific. For example, there could be a restriction on the maximum number of shifts. This kind of additional constraints are not column specific and have to be incorporated in the original SCP program. The latter results in the following extended mathematical formulation of the SCP

$$\begin{aligned} \min \quad & \sum_{j \in N} c_j x_j & & \\ \text{s.t.} \quad & \sum_{j \in N} a_{ij} x_j \geq 1 & \forall i \in M & \\ & \sum_{j \in N} d^2_{ej} x_j \leq C^2_e & \forall e \in E & \qquad (2.4) \\ & x_j \in \{0,1\} & \forall j \in N & \end{aligned}$$

in which set E denotes the additional constraints to be satisfied. The additional constraints (2.4) can be interpreted as restrictions on specific resources such as the total used workforce (which is not column specific). Parameter C^2_e represents the availability of each resource and d^2_{ej} denotes the amount of resource that is used by shift j.

[2] There may be restrictions which can not be formulated as a linear combination so easily. An example would be the incorporation of a break in shifts. However, for sake of simplicity we use the notation above.

Numerous relevant aspects for the CSP could be modeled as resources. The resulting constraints can be subdivided into two categories: the *hard* and *soft* constraints. Typically, the constraints deduced from labor regulations from the government are for example considered to be *hard* constraints. Constraints concerning the fairness of shifts and equal distribution of the workload can be seen as *soft* constraints. The *soft* constraints are in practice often considered to be of minor importance compared to the *hard* constraints. A few examples of the aspects which result in *hard* and *soft* constraints, are listed below. We distinguish between column specific aspects (○) and aspects which are not column specific (●).

Hard aspects in CSP:

- ○ Max. shift duration,
- ○ A shift should include a break,
- ○ A break should have a min. duration,
- ○ In case of scheduling crew for trains; the station at which the shifts start should be the same as at which it ends,
- ● A max. number of weekly shifts,
- ● A max. percentage of night shifts,
- ● A max. percentage of short/long shifts,
- ● A max. percentage of weekend shifts.

Soft aspects in CSP:

- ○ Preferred train types,
- ○ Preferred working times (during daytime/nighttime),
- ○ Work on routes for which there is a high probability of encountering aggressive passengers,
- ● Non-repetitive work,
- ● Fair allocation of the workload among different stations or groups of crew for example.

In this research, we will mainly focus on the *soft* constraints and how to efficiently incorporate these aspects in the problem. The resources considered in the problem (see Section 2.1.1) are modeled as column specific constraints. In this way, each feasible shift (column) is a fair combination of trips. A combination of fair shifts is also considered to be fair such that we automatically satisfy the constraint of a fair allocation of the workload among different stations. Since we want to focus on the *soft* aspects in this research, we do not include many *hard* aspects in the CSP as it will not be of any added value in this case. In order to properly perform the research, we discard constraints on the number of available crew members at each depot station for example. In this way, we avoid the problem to become unnecessarily complex and keep the solutions easy to interpret.

Next, we evaluate literature about (sub)topics of the problem considered in this research before we are going to describe the methodology used to solve the described problem.

3 Literature Review

In this chapter relevant literature on different subjects related to the research topic will be discussed and evaluated. In Section 3.1 a few papers are evaluated that discuss methods and algorithms used to solve the SCP. Furthermore in Section 3.2 literature is reviewed in which column generation is used to solve the SCP. Different applications of the SPPRC will be discussed in Section 3.2.1. In Section 3.3, we evaluate papers which present theories on distributive justice and how to incorporate this as an organization. Finally in Section 3.4, we conclude with the relevance of this research.

3.1 Set-Covering Problem (SCP)

The Set-Covering Problem (SCP) is an important formulation as it can be used to model and solve a large variety of real world problems such as scheduling, manufacturing and service planning. Balas (1982) contains a broad survey on the application of the SCP. The SCP is known to be NP-complete shown by Karp (1972) and difficult to solve especially when we are dealing with large scale instances.

3.1.1 Heuristics Approaches

Over the years researchers have been studying heuristics and exact approaches in order to obtain high quality solutions for the SCP. Since the SCP is considered to be hard to solve exactly and in general rather time consuming, many heuristics have been developed in order to obtain near-optimal solutions in less computational time. One of the most basic heuristics for the SCP is the greedy heuristic proposed by Chvatal (1979). This greedy algorithm basically selects the column with largest ratio, which is the number of rows covered, divided by the costs of that particular column. The algorithm continues selecting the columns with the largest ratio until all rows are covered. The greedy approach is easy to implement and has minor computation time, however it rarely creates high quality solutions.

The idea of Lagrangian Relaxation is basically to relax some of the more difficult constraints. Violations of these constraints are incorporated as penalties in the objective function. In his research (Beasley, 1990), Beasley discovered that the Lagrangian based heuristic outperformed the greedy algorithms known at that moment in time. Shortly thereafter, numerous heuristics are developed based on Lagrangian relaxation in combination with sub-gradient optimization (Ceria et al., 1998; Lorena and Lopes, 1994; Caprara et al., 1999).

Also some research has been done on the development of more randomized heuristics, the so-called meta-heuristics. The papers by Jacobs and Brusco (1995) and Beasley and Chu (1996) propose a genetic algorithm and an algorithm based on simulated annealing respectively to solve the SCP. These meta-heuristics have shown to be able to obtain high quality results compared to the Lagrangian-based heuristics.

3.1.2 Exact Approaches

Brand-and-bound algorithms in which the lower bounds are in general obtained by the LP relaxation of the SCP, can be seen as the most effective exact approaches to solve the SCP. The exact algorithms presented in the papers (Balas and Carrera, 1996; Beasley, 1987; Beasley and Jörnsten, 1992; Fisher and Kedia, 1990) are based on this idea. Caprara et al. (2000) compared multiple algorithms on the SCP and it appeared that among the exact algorithms CPLEX performs the best. The CPLEX solver uses a branch-and-cut method to find optimal solution for mixed integer programs. Branch-and-cut is a well-known combinatorial optimization method which uses a combination of branch-and-bound and cutting planes methods (Mitchell, 2011). The integer programming problem is solved by solving a series of subproblems. The subproblems are LP relaxations which are improved using the cutting planes.

3.1.3 Common Applications of the SCP

Scheduling and vehicle routing can be considered as one of the most common large-scale integer problems. Much literature exists in which various scheduling and vehicle routing related applications modeled as a SCP often in combination with column generation based algorithms as proposed solution method. It is beyond the scope of this research to evaluate these papers here in much detail. We refer the interested reader to the following literature on crew scheduling applications; (Desrochers and Soumis, 1989; Desaulniers et al., 2002; Grötschel et al., 2003; Huisman, 2007; Abbink et al., 2011) and for vehicle routing applications; (Desaulniers et al., 2002; Desrochers et al., 1992; Löbel, 1998; Toth and Vigo, 2002; Ceselli et al., 2009).

3.2 Column Generation

The column generation technique basically considers a subset of all feasible columns when solving the LP relaxation of the original problem. Now a subproblem, called the pricing problem, is solved in order to determine promising columns to enter the basis. The new column is then added to the LP relaxation after which the LP-relaxation is solved again. This process is repeated until no promising columns can be added anymore. Column generation can be very efficient for large-scale problems in particular since not all columns have to be taken into account explicitly.

In Desrosiers et al. (1984) an approach is introduced in which column generation techniques are used into a linear program based brand-and-bound framework for solving a vehicle routing problem with time window constraints. This can be seen as an important step in the design and development of exact algorithms to solve large integer programs. Shortly thereafter, Lavoie et al. (1988) proposed a column generation based algorithm for efficiently solving a crew pairing problem formulated as SCP. The foundations of the column generation technique have been laid around 60 years ago (Ford Jr and Fulkerson, 1958; Dantzig and Wolfe, 1960). However, some time was needed to develop improved methods as well as improved computers in order to realize the usefulness of the technique. Especially for large-scale integer problems it has been proven to be a successful approach as shown in Barnhart et al. (1998) where column generation is used in Branch-and-Cut and Branch-and-Price methods.

3.2.1 Shortest Path Problem with Resource Constraints (SPPRC)

The SCP has various applications as mentioned before, but the most common and widely used applications can be considered to be crew scheduling and vehicle routing. Typically these problems can be of enormous scale and therefore column generation is often used to solve problems like this (Barnhart et al., 1998). An important part of the column generation approach is solving the pricing problem which in case of a SCP, can be formulated as a SPPRC for all well-known applications (Dror, 1994).
The SPPRC was first presented in Desrochers and Soumis (1988) as subproblem of a bus driver scheduling problem. The classical shortest path problem (SPP) is modeled such that each path from the source to the sink represents for example a feasible workday or route. The goal is to find the shortest (cheapest) path possible. In case of the extended classical shortest path problem, i.e. the classical SPP with additional resource constraints, the optimal path does not only depend on the costs but also multiple resources should be taken into account. This causes the problem complexity to change from a polynomially solvable problem to a $\mathcal{NP}$-hard problem (Dror, 1994).

3.3 Incorporation of Fairness

The resources considered in a SPPRC can be of various types such as time, fuel consumption or distance. In certain applications one may also consider resources related to the shift specifications as some shifts may be preferred over others. This relates to allocation of duties such that equal fairness is perceived among all employees. Relatively little literature appears to be available concerning the incorporation of social aspects in duty allocation. In Breugem et al. (2017), the authors made an explicit trade-off between the attractiveness and the fairness in the construction of cyclic crew rosters for the Netherlands Railways. The Fairness-oriented Crew Rostering Problem (FCRP) is introduced and based on this formulation a Branch-Price-and-Cut solution method is developed. In Freling et al. (2004) a branch-and-price based decision support system for crew planning in passenger transportation is presented which creates rosters that satisfies both economic and social criteria. They introduced the so-called welfare objectives which enhance an equally spread workload among the crew for example, but they do not exactly show how the trade-off between the social and economic aspects is made. The downside is that unless the size of the problem was moderate, it is still a complex problem which the authors concluded to be a consequence of the highly complex objectives and constraints. A similar approach with multiple classes of objectives is presented in Maenhout and Vanhoucke (2010), however here a hybrid scatter search heuristic is proposed to solve the (personalized) rostering problem.

In Hartog et al. (2009) a method to solve the cyclic crew rostering problem is proposed. The problem is modeled as a set partitioning model with additional constraints. The fairness of the duty allocation is taken into account by penalizing an unfavorable sequence of duties. The objective is to minimize the total sum of the penalties. A similar approach is presented in Borndörfer et al. (2015), in which the authors distinguish both hard rules (labor regulations) and soft rules (preferences). The soft rules, e.g., minimizing unfavorable sequences of duties, are enforced by penalties. Nishi et al. (2014) proposed a decomposition method for crew rostering which incorporates the fairness. This approach is based on the observation that the implementation of fair working conditions by min-max type problems, cannot be applied easily. However, this study only considers the distribution of the workload whereas Hartog et al. (2009) and Borndörfer et al. (2015) also considered other duty specifications.

In social psychology, distributive justice is basically the perceived fairness of how rewards and costs are shared among group members (Forsyth, 2018). For example, when employees perform the same job and get paid a different salary, some employees may feel like that there is no distributive justice. Fairness is seen as a important issue recognized by organizations. It is not surprising that theories of social and interpersonal justice have been applied to understand behavior of employees in organizations. However, the first presented theories of social justice such as equity theory (Adams, 1963), were not focused on organizations in particular, but at justice in general social interaction. Thereafter, literature is published in which the role of fairness in a workplace environment is described (Goodman and Friedman, 1971; Greenberg, 1987; Weick and Nesset, 1968). The distribution of organizational rewards such as salary, promotion or performance evaluations can have powerful effects on the job satisfaction, organizational effectiveness and quality of work life (Lawler, 1977). Similar relations are described in Alexander and Ruderman (1987) and Colquitt et al. (2001), which indicates the importance of organizational justice.

3.4 Relevance

In summary, over the years much research has been done on the SCP and its applications. Various heuristics and exact methods have been developed in order to obtain high quality solutions for large-scale problems such as CSP and VRP. Literature shows that column generation in particular is very suitable for solving this kind of problems. Also relatively much literature has shown the importance of fairness and distributive justice in organizations. Surprisingly, literature on the incorporation of fairness in crew scheduling problems seems to be scarce. The constraints concerning the fairness of the solution are typically described as ‘soft’ constraints and adding these additional constraints may increase the complexity of the initial problem. This may be one of the reasons it is not a ‘popular’ action to incorporate the soft constraints concerning the fairness for example. However, in order to avoid incidents like the strike of the NS personnel (Abbink et al., 2005) and to make sure the employees are satisfied with their job, it can be very beneficial to incorporate these aspects in your problem. In this research, we investigate the possibility of efficiently taking into account the fairness in crew scheduling problems such that a fair solution is obtained.

4 Methodology

In this chapter, we elaborate on the methodology which is used to solve the SCP and we also explain the methods used to aggregate the resources. In Section 4.1, we first explain the fundamentals of the solution method used in this research namely the column generation method. The associated pricing problem is commonly modeled as a shortest path problem with resource constraints (SPPRC) which is explained in Section 4.2. We propose a framework for the pricing problem based on a dynamic programming approach in order to solve the SPPRC. In Section 4.3, we evaluate the pricing framework and we explain the underlying idea. Finally, we present the different aggregation strategies used to aggregate the resources in Section 4.4.

4.1 Column Generation

The simplex method is a classic method for solving the LP relaxation of linear integer programs such as the mathematical formulation of a SCP given in Section 2.2.2. However, the complete enumeration of all feasible shifts is computationally intractable for large problem instances. Column generation is a frequently used method for successfully solving a wide variety of problems (such as vehicle routing and crew scheduling problems see Section 3).

4.1.1 The Idea of Column Generation

As mentioned above, column generation is a commonly applied technique to solve large (integer) linear problems which are too large to consider all variables explicitly. The method exploits the idea that many variables will have a corresponding value of zero in most of the feasible solutions, that is, the corresponding columns (shifts) will not be used frequently in the construction of feasible solutions. Therefore, they are excluded in the original master problem formulation. The resulting problem is in fact the original master problem in which only a subset $N' \subseteq N$ of the variables is taken into account. We refer to this problem as the restricted master problem (RMP).
The column generation method decomposes the problem into the RMP and the pricing problem. The subset $N' \subseteq N$ of columns is extended with columns $j \in N \backslash N'$ generated by solving the pricing problem. The idea is that only columns (and the corresponding variables) are added to the RMP which improve the objective function. This procedure is repeated until no such columns can be determined anymore. The benefit of this method is that we take into account a (much) smaller subset of columns N' which makes it computationally more tractable.

Algorithm 1.1 shows the pseudocode of the column generation method implemented in this research. One can observe that the RMP and the pricing problem are solved in step 2.1 and 2.2 respectively. The pricing problem is solved using a pricing framework which is indicated by the rectangle in 1.1. The pricing framework is explained in Section 4.3.

Algorithm 1.1 Column Generation Algorithm

```
    Input : A dataset containing a number of $n$ tasks which should be performed.
    Output : A least cost set of shifts such that each task is performed at least once.
 1: Step 1 : Initialization
 2: Initialize the master problem by adding a couple of feasible shifts such that a feasible solu-
    tion can be found and initialize $obj_{pricing} = -1$.
 3:
 4: Step 2 : Generate columns (shifts)
 5: while $obj_{pricing} < 0$ do
 6:     Step 2.1 : Solve RMP
 7:     Solve the RMP and update the value of the dual variables on the corresponding arcs.
 8:     Step 2.2 : Solve Pricing Problem
 9:     $sol \leftarrow$ Algorithm 1
10:     if NotFeasible(sol) then
11:         $sol \leftarrow$ Algorithm 2
12:         if NotFeasible(sol) then
13:             $sol \leftarrow$ Algorithm 3
14:         end if
15:     end if
16:     if $obj_{pricing} < 0$ then
17:         Add new column to the RMP.
18:     else
19:         break
20:     end if
21: end while
22:
23: Step 3 : Solution
24: We obtain the LP relaxation of the original problem consisting of a set of shifts which
    together cover the tasks at least once.
```

4.1.2 Restricted Master Problem

The RMP is basically the same as the original problem (see Section 2.2.2) but, the integrality constraints (2.3) are relaxed and a restricted set of columns $N' \subseteq N$ is taken into account. The purpose of the RMP is to provide a vector of dual variables u which is passed on to the subproblem. A mathematical formulation of the RMP corresponding to the original problem is given by

$$\min \quad \sum_{j \in N'} c_j x_j \tag{4.1}$$

$$\text{s.t.} \quad \sum_{j \in N'} a_{ij} x_j \geq 1 \qquad \forall i \in M \tag{4.2}$$

$$0 \leq x_j \leq 1 \qquad \forall j \in N' \tag{4.3}$$

The column generation method solves the LP relaxation of the original problem. Integer solutions are obtained by combining column generation with a Branch-and-Bound approach which is also known as Branch-and-Price.

4.1.3 Pricing Problem

The subproblem of the column generation method is of great importance as it generates possibly promising variables. In case the newly generated variable has negative reduced costs (for a minimization problem), it will be added to the RMP. The pricing problem can be mathematically formulated by

$$\bar{c}^* := \min\{c(a) - u^T a | a \in \mathcal{A}\} \tag{4.4}$$

in which $c(a)$ is a function to calculate the cost corresponding to column a and $\mathcal{A} \neq \emptyset$ is a set consisting of feasible columns a_j for all $j \in N$. Furthermore, u represents the dual variables corresponding to constraints (4.2). The resulting pricing problem for the particular problem considered in this research can be formulated as a SPPRC which will be explained in Section 4.2.

4.2 Shortest Path Problem with Resource Constraints

The shortest path problem (SPP) corresponds to the problem in which one should find a path between two nodes such that the associated costs are minimized. In case of a resource constrained shortest path problem (SPPRC), the path must also satisfy a set of constraints related to a set of resources. A resource corresponds to quantities such as fuel and time for example. In case the original problem corresponds to a crew scheduling problem (vehicle routing problem), a feasible path represents a set of tasks (vehicle route).

The SPPRC can be formally described using graph theory. We consider a directed graph $G = (V, A)$ where V and A corresponds to a set of $|V| = n$ nodes and $|A| = m$ arcs respectively. Furthermore, we introduce a source node s and target node t. Each arc $a \in A$ has costs c_a and uses r_a^k units of resource k for all $k \in \{1, 2, ..., K\}$. The maximum capacity of each resource k is defined as C^k. Costs and resources are assumed to be non-negative. The objective of the SPPRC is to find a least cost shortest path from the source to the target. A mathematical formulation of the SPPRC is given by

$$
\begin{aligned}
\min \quad & \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} & & \qquad (4.5)\\
\text{s.t.} \quad & \sum_{i \in V} \sum_{j \in V} r_{ij}^k x_{ij} \leq C^k & \forall k \in \{1, 2, ..., K\} & \qquad (4.6)\\
& \sum_{i \in V} x_{ij} = \sum_{i \in V} x_{ji} & \forall j \in V \backslash \{s, t\} & \qquad (4.7)\\
& \sum_{j \in V} x_{sj} = 1 & & \qquad (4.8)\\
& \sum_{i \in V} x_{it} = 1 & & \qquad (4.9)\\
& x_{ij} \in \{0, 1\} & \forall i, j \in V & \qquad (4.10)
\end{aligned}
$$

where the decision variable x_{ij} indicates whether arc $(i, j) \in A$ is selected or not. That is

$$
x_{ij} = \begin{cases} 1 & \text{if arc } (i, j) \text{ belongs to an optimal path,} \\ 0 & \text{otherwise.} \end{cases}
$$

Constraints (4.6) ensure that the total resource consumption of a path does not exceed the maximum resource capacity. Constraints (4.7) make sure that for every node $j \in V \backslash \{s, t\}$ the indegree equals the outdegree. Furthermore, constraints (4.8) and (4.9) ensure that the indegree and outdegree for the source and target node respectively equals one. Finally, constraints (4.10) make sure that the solution is binary.

Numerous exact and non-exact solution methods for the SPPRC have been described in the literature over the past few years. An overview of these methods is given in Appendix A. After studying the various methods to solve the SPPRC, it has been determined to implement a labeling algorithm to solve the SPPRC in a column generation context. An advantage of the labeling algorithm is the fact that it is clearly interpretable and therefore, it is a suitable method for the insertion of potential resource aggregation extensions. In Section 4.3, we focus on the implementation of a solution method for the SPPRC, but we first give some more context on how we define the associated graph used to solve the pricing problem.

4.2.1 Graph Formulation

In this research the pricing problem contains trips which both have a fixed starting and a fixed ending time. Arcs are only constructed when trips are compatible. In this way, it is ensured that there cannot exist any cycles in the constructed graph. In our case, the graph consists of the following characteristics:

- Directed graph,
- The graph is acyclic,
- The graph is incomplete,
- The 'costs' for each arc can either be positive or negative.

We consider a graph $G = (V, A)$ with source node $s \in V$ and target node $t \in V$. Node $i \in V \backslash \{s, t\}$ represents a trip i which has a departure time t_i^d from departure station S_i^d and an arrival time t_i^a at arrival station S_i^a. We connect node i with node j if it holds that $t_i^a \leq t_j^d$ and $S_i^a = S_j^d$. This automatically ensures that it is a directed graph since the trips will only be connected if they are compatible with respect to the arrival and departure times. It also guarantees that no cycles will occur in the graph. The connection between two nodes is called an arc which we denote by x_{ij}. We introduce arcs x_{si} for all nodes $i \in V \backslash \{s, t\}$ for which it holds that the departure station $S_i^d \in \overline{D}$ where $\overline{D}$ represents a set with all the depot stations. Every shift must start at a depot station as all the trains will be shunted at the depot stations. This also holds for ending a shift which means that we introduce an arc x_{it} for all $i \in V \backslash \{s, t\}$ for which it holds that $S_i^a \in \overline{D}$. The cost c_{ij} on arc x_{ij} are determined using the following relation in which C_{init} denotes the initial costs accounted for a shift, C_{labor} denotes the labor costs and u_j denotes the dual variable corresponding to node j.

$$c_{ij} = \begin{cases} C_{init} + 2C_{labor}(t_j^a - t_j^d) - u_j & \text{if node } i = s \text{ and } j \neq t \\ C_{labor}(t_j^d - t_i^a) + 2C_{labor}(t_j^a - t_j^d) - u_j & \text{if node } i \neq s \text{ and } j \neq t \\ 0 & \text{otherwise.} \end{cases}$$

It can be observed that the personnel is twice as expensive during working time compared to time in between two trips. We pre-process the graph by eliminating all arcs for which $t_j^d - t_i^a > 15$. This means that it is not allowed to have more than 15 minutes in between two tasks which is not desirable anyway since we consider a cost C_{labor} per minute waiting time. By eliminating these arcs, we reduce the number of potential paths and the complexity of the problem instance.

4.2.2 Multiple Pricing Problems

In the constructed network, we are dealing with multiple depot stations. The NS personnel must end a shift at the same station as where the shift started. The pricing problem becomes more complicated if we adapt the generic version of the labeling algorithm such that it is applicable for a problem in which we consider multiple depot stations. Therefore, we choose to split the pricing problem into multiple pricing problems corresponding to each depot. We construct a new graph every time we switch from depot station in which we only consider arcs between the source/sink node and this particular depot station. This means that it holds that $|\overline{D}| = 1$ for the graph considered in each pricing problem. The algorithm changes from depot station if no more paths with negative reduced costs can be found.

4.3 The Pricing Framework

In this section, we elaborate on the implementation of the algorithm to solve the SPPRC. The implementation can be described as a framework in which we can distinguish three algorithms. The algorithms are based on a class of dynamic programming algorithms namely the labeling algorithm. We provide a generic labeling algorithm and the adaptations we made concerning the incorporation of the resources. In this section, we explain the pricing framework by first considering the individual elements of the framework after which we connect the elements and explain the underlying idea.

4.3.1 Basis: A Generic Labeling Algorithm

The idea of the dynamic programming approach is to create labels at each node for all feasible partial paths consisting of the associated costs and resource consumption. Labels can be eliminated using dominance rules. We distinguish between two types of labeling algorithms: label-correcting and label-setting. In label-correcting methods nodes can be treated more than once contrary to label-setting algorithms nodes will be treated at most once. The label-setting algorithm uses a Best-Search-First rule and is basically an extension of Dijkstra's algorithm (Dijkstra, 1959). A requirement for the application of Dijkstra's algorithm is to have only positive costs on the arcs. As mentioned in Section 4.2.1, we consider a graph in which the arcs can also have negative costs so this means that it is not possible to apply a label-setting algorithm.

Algorithm 1.2 shows a generic version of the label-correcting algorithm. Note, that this algorithm only considers the costs and not any of the resources. The labels consist of two elements: $L_i = (d_i, parent_i)$ where $parent_i$ is the parent node and d_i the total costs to reach node i. Step 1 basically consists of the initialization of all the labels L_i and the $NodeList$. The $NodeList$ is initialized by adding the source node s with corresponding label $L_s = (0, s)$ since the starting cost of an optimal path is set to zero. The label for each of the other nodes are initialized by $L_i = (\infty, i)$. Note, that each node has only a single label and the labels are (possibly) updated throughout the course of the algorithm. The fact that we chose to consider one label for each node has to do with the difficulty of applying dominance rules explained in Section 4.3.4.

The next step can be seen as the core of the algorithm in which the labels of the nodes are updated sequentially following a certain procedure. At the start of each iteration a node is selected from the *NodeList* following some selection rule. A well known selection rule is the FIFO rule for example. The selected rule could be of great impact on the performance of the algorithm. In Section 4.3.2, several rules are discussed. The *NodeList* should be updated after a node is selected. We look for adjacent nodes and determine if a new path connecting the adjacent node results in a better solution (cheaper path) than we found so far. If this is the case, we say that the new path dominates the previous found path. Only if this is the case, the label of the adjacent node is updated and the adjacent node is added to the *NodeList* if and only if it is not already in there. This procedure is repeated until the *NodeList* is empty. The optimal path can then be obtained by backtracking the parent nodes starting from the sink node t. The label of the sink node contains the objective value corresponding to the obtained $s - t$ path denoted by d_t.

4.3.2 Node Selection Rules

The labeling algorithm considers a single node in each iteration. It looks for possible extensions of the path from the current node i to its adjacent nodes. If a cheaper path is determined for one of its adjacent nodes, the corresponding label is updated and the path is extended. The considered selection rule might have some impact in the performance of the algorithm. Certain selection rules tend to select nodes which result in poor solutions (expensive paths) and typically more computation time is needed until the cheaper paths are determined as well. Other selection rules tend to select interesting nodes much faster.
Line number 8 of Algorithm 1.2 shows the selection of a node. The generic labeling algorithm uses the FIFO rule as can be observed in the pseudocode. Several possibilities concerning a node selection rule are listed below:

- FIFO - *First In First Out*
  The FIFO rule is probably the most common selection rule and simply selects the node which has been in the *Nodelist* the longest.

- LSFO - *Latest Start time First out*
  The LSFO rule selects the node from the *Nodelist* with the latest start time. This rule is quite similar to a Depth-First search in which nodes further in the graph will be selected first. This is a good selection rule if it is required to find a path very quickly, however it might be not the most efficient way to find good solutions.

- BSFO - *Best Search First Out*
  This selection rule is also known as Dijkstra's method. The rule selects the node with the best (partial) solution found so far. Note that this rule is only applicable in case you have nonnegative arc lengths.

Algorithm 1.2 Label-Correcting Algorithm

```
Input : A directed graph G = (V, A) with source node $s \in V$ and target node $t \in V$.
The cost of the arc $c_{ij}$ can be either positive and negative. Set $V_i^-$ contains all the
outgoing arcs of node $i$.
Output : The shortest path from $s$ to $t$.
1:  Step 1 : Initialization
2:  Set NodeList = {s} and corresponding label $L_s = (0, s)$
3:  For each node $i \in V \backslash \{s\}$: set label $L_i = (d_i, parent_i) = (\infty, i)$.
4:
5:  Step 2 : Labeling
6:  while $NodeList \neq \emptyset$ do
7:      Step 2.1 : Node Selection
8:      Select a node $i$ from NodeList according to the FIFO rule.
9:      $NodeList \leftarrow NodeList \backslash \{i\}$
10:
11:     Step 2.2 : Update labels
12:     for all $j \in V_i^-$ do                                  ▷ Evaluate all outgoing arcs
13:         if $d_j > d_i + c_{ij}$ then
14:             $parent_j \leftarrow i$
15:             $d_j \leftarrow d_i + c_{ij}$
16:             Set label $L_j \leftarrow (d_j, parent_j)$
17:
18:             Step 2.3 : Update NodeList
19:             if $j \notin NodeList$ and $j \neq t$ then
20:                 $NodeList \leftarrow NodeList \cup \{j\}$     ▷ Nodes cannot enter the list if already listed
21:             end if
22:         end if
23:     end for
24: end while
25:
26: Step 3 : Solution
27: Total cost of the (optimal) shortest $s - t$ path $\leftarrow d_t$
28: The nodes in the optimal path can be obtained by backtracking starting from $parent_t$
```

We explained the basis of the pricing framework and provided a generic label-correcting algorithm (see Algorithm 1.2). In the following sections (Section 4.3.3, 4.3.4 and 4.3.5), we elaborate more on the details of the adjusted versions of the basis algorithm such that we can incorporate resources for example.

4.3.3 A Cost Based Labeling Algorithm

The generic label-correcting algorithm 1.2 is in fact based on costs. The FIFO selection rule is used and the algorithm continues iterating until the *NodeList* is empty. The algorithm solves the pricing problem and obtains the cheapest path among all feasible paths. The labels L_i consist of two elements namely, the total costs to reach node i and the parent node. The resources are not taken into account in this algorithm.

4.3.4 A Resource Based Labeling Algorithm

We adjusted Algorithm 1.2 such that besides the costs, we can also take into account the resources (see Algorithm 1.3). The labels are extended and contain the resource information of the corresponding path. In this case the labels consist of the following elements: $L_i = (d_i, parent_i, \vec{R}, T)$. The resource 'consumption' $\vec{R} = (R^1, R^2, ..., R^K)$ and the total duration T of the corresponding path are now to be updated as well.

The algorithm is used to find resource feasible paths with a corresponding negative reduced cost. The considered resources are not the typical type of resources such as time or fuel, but they are related to the preferences of the NS personnel (see Section 2.1.1). In the end we aim to generate shifts with an equally fair distribution of the resources. The difficulty that comes with the considered problem and the type of resources is that we are not able to consider a maximum capacity of a particular resource. For example, we do not have a maximum capacity of Intercity trains in each shift. Instead, we want to generate shifts which have a particular fraction of working hours on an Intercity train. The resource consumption $\vec{R}$ and the shift duration T can be used to determine the fraction of each resource for the associated path. The desirable fraction of each resource is determined by the instance average which is calculated by

$$\overline{R}^k = \frac{\sum_{i \in M} r_i^k T_i}{\sum_{i \in M} T_i}, \qquad \forall k \in \{1, 2, ..., K\},$$

where T_i is the duration of trip $i \in M$ and r_i^k is the value of resource k corresponding to trip i. The resources are modeled in such a way that they are not preferred by the NS personnel which results in an upper bound UB_r^k for every resource k when assessing the resource feasibility.

In Figure 4.1, we illustrated a small example concerning the assessment of resource feasibility. The numbered nodes each denote a specific trip which we consider to have an equal trip duration for sake of simplicity. In this example, we only take into account one of the resources such that the gray (white) colored nodes correspond to trips performed by an Sprinter (Intercity) train.

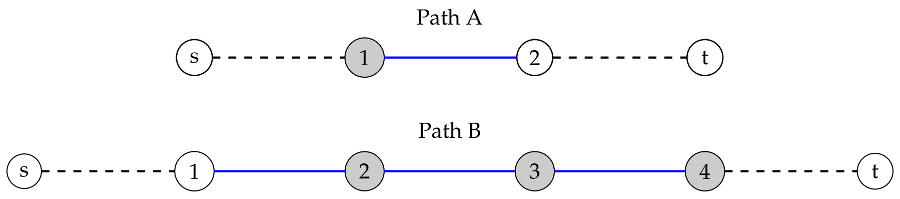

Figure 4.1: Two possible path configurations are displayed with four and six nodes respectively. The gray nodes correspond to Sprinter trips whereas the white (numbered) nodes are considered to be Intercity trips. Node s and t denote the source and sink node respectively.

In case we assess the resource feasibility for the paths in this example, we find the *Train Type* resource fraction for both path A and B to be:

$$\overline{R}_A^{Type} = \frac{R_A^{Type}}{T_A} = 0.5,$$
$$\overline{R}_B^{Type} = \frac{R_B^{Type}}{T_B} = 0.75.$$

Let us consider to have a desired resource fraction $UB_r^{Type} = 0.6$, it means that path A is considered to be resource feasible whereas path B is not.

The consequences of the dominance rules are important in the implementation of this algorithm. In case we use dominance rules based on either costs or resources, this results in losing a lot of high potential paths. For example, if we solely focus on costs, the cheapest path might be resource infeasible while there could be resource feasible paths which are a bit more expensive. On the other hand, if we focus on the dominance of paths based on the resources, we may find a path with the smallest resource fractions. However, it is possible that this particular path has positive reduced costs while there might exist many paths which do have negative reduced costs and also satisfy the resource bounds. In order to avoid losing resource feasible paths with negative reduced costs, we implemented the LFSO selection rule in combination with no dominance rules. We simply always consider the current path to dominate the path corresponding to an adjacent node such that the label is updated in all cases. In combination with the LSFO rule, which considers the node with the latest start times first, it can be determined that in this way no feasible path is excluded. The algorithm continues iterating until we find the first resource feasible path with negative reduced cost.

In theory all paths will be enumerated until a resource feasible path with negative reduced costs is generated and the path (column) and the associated variable will be added to the RMP. The bounds on the resource fractions cannot be used straight away to eliminate certain (partial) paths. We refer to Figure 4.1 to illustrate what happens in case we prune (partial) paths based on the bounds. When generating path A, we find that after a single node the resource fraction of the *Train Type* resource is 1 (Sprinter) which means that the path is pruned straight away since $UB_r^{Type} < 1$. However, path A appears to become resource feasible when adding the second node. The opposite is observed for path B and we conclude that this is not an efficient pruning strategy.

We should focus on the possibility that a resource infeasible partial path becomes resource feasible within a certain time period. We developed a pruning strategy such that the complete enumeration of all possible paths is avoided. Paths will be pruned if at time t_i^a the resource consumption does not meet the resource constraints *and* if there is no possibility to satisfy the constraints in the remaining time $t_{end} - t_i^a$ by extending the path. The path will be pruned if for one of the resources, the following inequality does not hold:

$$\frac{R^k}{T + (t_{end} - t_i^a)} \leq UB_r^k, \qquad \forall k \in \{1, 2, ..., K\}.$$

The best case scenario for the resource fraction at time t_i^a is calculated by dividing the weighted resource R^k by the maximum shift duration. The latest ending time of a duty t_{end} can be determined beforehand and may differ among the depot stations. We can prune the path in case the fraction is larger than the upper bound UB_r^k. Furthermore, paths will be pruned in case the maximum duration (8 hours) of a shift will be exceeded.

4.3.5 An Aggregation Based Labeling Algorithm

The aim of aggregating resources is to obtain less criteria which should be taken into account when determining the feasibility of a path which makes it computationally more tractable. The adaptation of Algorithm 1.2 such that we are able to incorporate the aggregated resources is similar as explained in Section 4.3.4. The main difference is that we update the aggregated resource consumption instead of the regular resource consumption. A label looks as follows: $L_i = (d_i, parent_i, \vec{R_{ar}}, T)$ where $R_{ar} = (R_{ar}^1, R_{ar}^2, ..., R_{ar}^{\hat{K}})$. Another difference is the fact that the domain of the aggregated resources r_{ar}^k differs compared to the regular resources r^k due to the aggregation methods (see Section 4.4). The regular resources are binary or integer valued whereas the aggregated resources are real numbers, $r_{ar}^k \in [\underline{r_{ar}}^k, \overline{r_{ar}}^k]$. The minimum value of every regular resource is zero, but this might not be the case for the aggregated resources. Therefore, we adapted the pruning strategy as follows:

$$\frac{R_{ar}^k + \underline{r_{ar}}^k (t_{end} - t_i)}{T + (t_{end} - t_i)} \leq UB_{ar}^k, \qquad \forall k \in \{1, 2, ..., \hat{K}\}.$$

The upper bound on the aggregated resource k is denoted by UB_{ar}^k. Note, that in this case we consider $\hat{K}$ aggregated resources for which it holds that $\hat{K} < K$.

Algorithm 1.3 Label-Correcting algorithm based on resources

```
    Input : A directed graph G = (V, A) with source node $s \in V$ and target node $t \in V$.
    Output : A resource feasible path from $s$ to $t$ with negative costs.
 1: Step 1 : Initialization
 2: Set $NodeList$ = {s} and corresponding label $L_s = (0, s, 0, 0, 0, 0, 0)$
 3: For each node $i \in V \backslash \{s\}$: set label $L_i = (d_i, parent_i, R_i^1, R_i^2, R_i^3, R_i^4, T_i) = (\infty, i, 0, 0, 0, 0, 0)$.
 4: Initialize a boolean variable $Feas = false$.
 5: Step 2 : Labeling
 6: while $NodeList \neq \emptyset$ & $Feas = false$ do
 7:     Step 2.1 : Node Selection
 8:     Select a node $i$ from $NodeList$ according to the LSFO rule.
 9:     $NodeList \leftarrow NodeList \backslash \{i\}$
10:     Step 2.2 : Update labels
11:     for all $j \in V_i^-$ do                                  ▷ Evaluate all outgoing arcs
12:         $parent_j \leftarrow i$
13:         $d_j \leftarrow d_i + c_{ij}$
14:         $R_j^k \leftarrow R_i^k + (t_j^a - t_j^d) r_j^k \qquad \forall k \in \{1, 2, 3, 4\}$
15:         $T_j \leftarrow T_i + (t_j^a - t_j^d)$
16:         Step 2.3 : Pruning
17:         if $isPruned(R_j^1, R_j^2, R_j^3, R_j^4, T_j)$ then                ▷ Some pruning function
18:             Path is eliminated $\rightarrow$ continue with next outgoing arc
19:         else
20:             Set label $L_j \leftarrow (d_j, parent_j, R_j^1, R_j^2, R_j^3, R_j^4, T_j)$
21:             Step 2.4 : Update NodeList
22:             if $j \notin NodeList$ and $j \neq t$ then
23:                 $NodeList \leftarrow NodeList \cup \{j\}$
24:             end if
25:             Step 2.5 : Check feasibility
26:             if $j = t$ then
27:                 if $ResourceFeasible(R_j^1, R_j^2, R_j^3, R_j^4)$ & $d_j < 0$ then     ▷ Evaluate feasibility
28:                     $Feas \leftarrow true$
29:                 end if
30:             end if
31:         end if
32:     end for
33: end while
34:
35: Step 3 : Solution
36: The cost corresponding to the resource feasible $s - t$ path $\leftarrow d_t$
37: The nodes in the optimal path can be obtained by backtracking starting with $parent_t$
38: The fraction of each resource $(\frac{R_t^1}{T_t}, \frac{R_t^2}{T_t}, \frac{R_t^3}{T_t}, \frac{R_t^4}{T_t})$.
```

4.3.6 The Idea of the Framework

The subproblem of the column generation method is solved using the pricing framework which consists of 3 different labeling algorithms. The pseudocode from Algorithm 1.1 corresponding to the pricing framework is shown in Figure 4.2 from which it can be observed that the algorithms are executed in a specific order.

The pseudocode corresponds to the following labeling algorithms discussed in the previous sections:

1. Algorithm 1 : a cost based labeling algorithm (see Section 4.3.3),
2. Algorithm 2 : an aggregation based labeling algorithm (see Section 4.3.5),
3. Algorithm 3 : a resource based labeling algorithm (see Section 4.3.4).

```
...
Step 2.2 : Solve Pricing Problem
sol ← Algorithm 1
if NotFeasible(sol) then
    sol ← Algorithm 2
    if NotFeasible(sol) then
        sol ← Algorithm 3
    end if
end if
...
```

Figure 4.2: The pseudocode for the pricing framework obtained from Algorithm 1.1.

The algorithms are executed in a specific order until a feasible path is found for each column generation iteration. The corresponding variable is added to the restricted master problem if and only if the path improves the objective function of the RMP. Let us exclude the second algorithm from the pricing framework for now. The third algorithm results in resource feasible paths with negative reduced costs, but it may take relatively much iterations of the column generation method to solve the RMP to optimality since the algorithm mainly focuses on the resource feasibility in the sense that it stops when it finds the first resource feasible path. The first algorithm results relatively fast in the cheapest path possible, however very often this will not result in a resource feasible path. The pricing framework first starts with the labeling algorithm based on costs as it is relatively fast and produces high quality paths. It can be seen as a bonus if the path appears to be resource feasible and otherwise we invoke algorithm 3.

The idea of adding algorithm 2 in between these two algorithms is to find resource feasible paths making use of a resource aggregation strategy. Typically when resources are aggregated, less criteria should be taken into account when determining the feasibility of a path which makes it computationally more tractable compared to algorithm 3. Algorithm 2 results in resource feasible paths if the aggregation strategy appears to be successful. In this way the insertion of algorithm 2 could have positive impact on the overall performance of the SPPRC solution method. We use a benchmark which is obtained by solving the instance while excluding algorithm 2 in order to compare and measure the impact of algorithm 2.

4.4 Aggregation strategies

Algorithm 2 of the pricing framework is used to incorporate a resource aggregation strategy when solving the pricing problem. The theory behind the different resource aggregation strategies is evaluated in this section.

4.4.1 Principal Component Analysis (PCA)

A principal component analysis is a statistical procedure which transforms a dataset consisting of (correlated) variables into a set of principal components. The principal components are uncorrelated linear combinations. The idea is that the first component summarizes the largest amount of variability in the dataset. The main objective for a principal component analysis is to reduce the number of variables in your dataset and to improve the interpretation. In this case we consider a dataset $X' = [X_1, X_2, X_3, X_4]$ consisting of four resources. We denote the covariance matrix of the data by Σ with eigenvector and eigenvalue pairs (e_1, λ_1), (e_2, λ_2),(e_3, λ_3) and (e_4, λ_4) where $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4 \geq 0$. The principal components are constructed by a linear combination of the data $X' = [X_1, X_2, X_3, X_4]$ and the eigenvectors e_1, e_2, e_3 and e_4:

$$Y_i = e_i'X = e_{i1}X_1 + e_{i2}X_2 + e_{i3}X_3 + e_{i4}X_4, \qquad i = \{1, 2, 3, 4\}.$$

We can write the first principal component as $Y_1 = e_1'X$. The eigenvectors are obtained from the covariance matrix of the dataset. This method is sensitive for scaling. Since the variables in our dataset have different domains, we first standardize the data as follows:

$$Z_i = \frac{X_i - \bar{X}_i}{s_i},$$

where s_i is an approximation of the standard deviation and $\bar{X}_i$ is the mean of variable i. The eigenvalues can be used to determine the appropriate number of principal components to represent the variance of the dataset. When we sort the eigenvalues in descending order and plot the magnitude of the eigenvalue against the number, we obtain a so-called scree plot. Typically in a scree plot an elbow is observed at some eigenvalue number i. This is an indication that

i principal component(s) will approximately summarize the total variance of the dataset. An example of such a scree plot is given in Figure 4.3 (left hand side). The figure also contains a graph (right hand side) which shows the cumulative proportion of variance explained by i principal components.

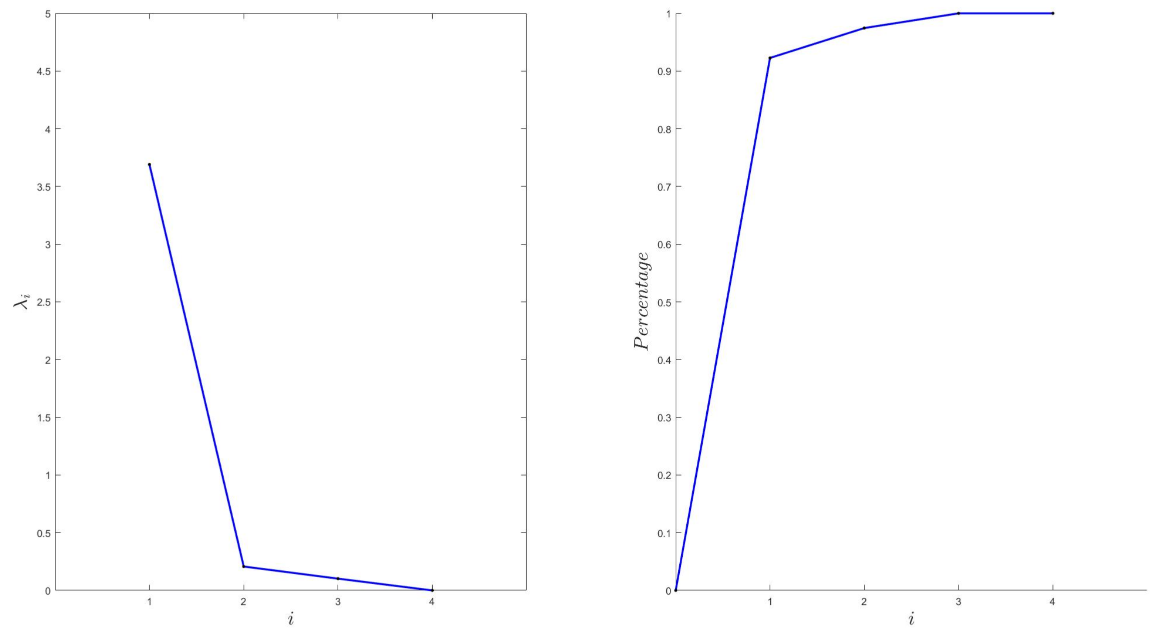

Figure 4.3: Left: an example of a scree plot in which we clearly observe the elbow. Right: the corresponding graph of the cumulative proportion of variance explained by i principal components.

The bounds used in the resource based labeling algorithm are the weighted resource averages of the instance. Due to scaling, it is not possible to obtain new bounds on the aggregated resources by simply using the same linear combination used for the aggregation of the resources. Instead, the desired fraction of the aggregated resource $\overline{Y}_1$ is obtained in similar fashion as for the regular resources:

$$\overline{Y}^1 = \frac{\sum_{i \in M} Y_i^1 T_i}{\sum_{i \in M} T_i},$$

in which Y_i^1 represents the aggregated resource (first principal component) and T_i represents the duration of trip $i \in M$. The desired fraction of the aggregated resource $\overline{Y}^1$ is comparable to $\overline{R}^k$, but the interpretation disappeared due to the aggregation method.

4.4.2 Ad-hoc Method

The ad-hoc aggregation method uses the correlation coefficients in the calculation of the weights. The weights determine the importance of each resource and in which degree it should be present in the aggregated resource. Firstly, we calculate the correlation matrix for the resources X. The resource which has the strongest correlation with the other resources is called the *dominant* resource. The correlation coefficients of the *dominant* resource are used to obtain an aggregated resource.

Table 4.1: Correlation matrix

	Type	Decker	Stops	Aggression
Type	1	0.54	-0.70	-0.41
Decker	0.54	1	-0.41	-0.22
Stops	-0.70	-0.41	1	-0.18
Aggression	-0.41	-0.22	-0.18	1

For example, in Table 4.1 a correlation matrix given and the *Type* row is highlighted since it is the *dominant* resource k_d in this case (it has the highest correlation coefficients $\sum_{k\in\{1,2,\ldots,K\}} |\rho_{k_d,k}|$). We define a weight v_k for each resource as follows

$$v_k = \frac{w_k}{w_t} \qquad \forall k \in \{1, 2, \ldots, K\}$$

where

$$\begin{aligned} w_k &= (1 - |\rho_{k_d,k}|)^n, \\ w_t &= 1 + \sum_{k\in K} w_k. \end{aligned}$$

Now the aggregated resource r_{ar} is obtained by:

$$r_{ar} = \sum_{k\in\{1,2,\ldots,K\}} v_k r^k$$

where r^k represents the original resource k. The underlying idea of this aggregation strategy is that the resources which are strongly correlated with the *dominant* resource, will have a less significant contribution to the obtained aggregated resource since they will be more or less represented by the *dominant* resource. The resources with minor correlation have a larger weight and in this way they have a slightly larger contribution to the aggregated resource. Different values of the power n result in a different distribution of the weights. In Figure 4.4, the weight distribution is given for different values of n in case we have two original resources. It can be observed that the resources which are slightly correlated become more 'important' for higher values of n compared to the stronger correlated resources.

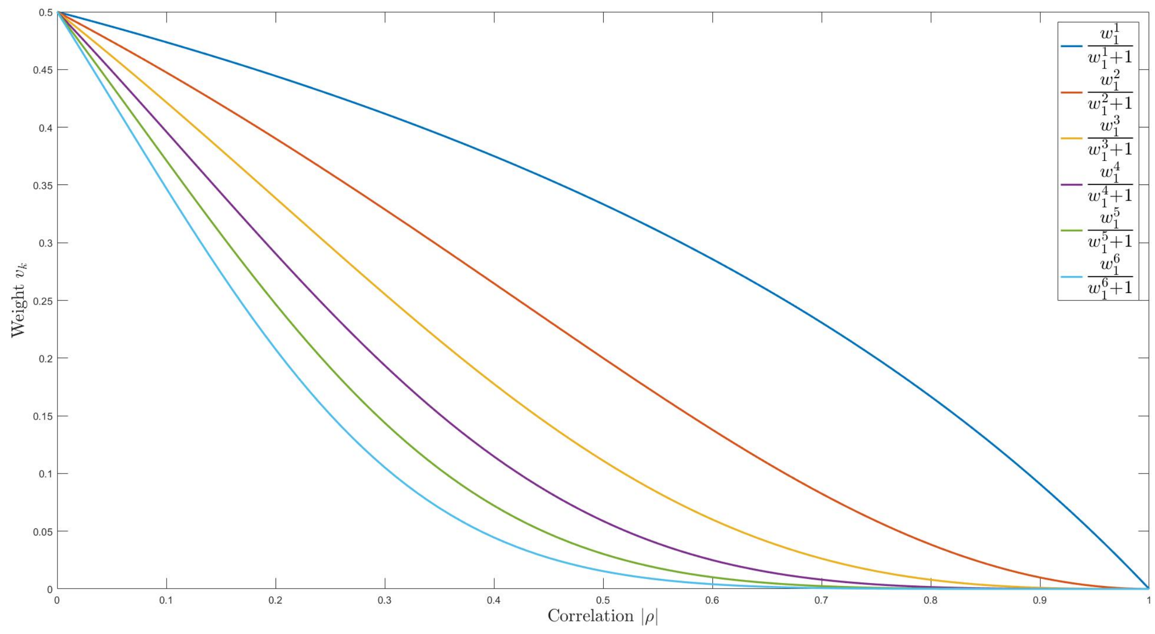

Figure 4.4: The relation is shown of the dependence of the weights v_k vs. the correlation coefficient ρ_k for different values of n

4.4.3 Basic Method

The third (aggregation) strategy is very straight forward. The idea is to simply consider only one or a selection of the original resources and still obtain a fair distribution for all of the resources. If two resources are strongly correlated, $|\rho| \geq 0.7$, one could for example choose to only focus on a single resource. This strategy will probably be most successful when some resources are strongly correlated.

5 Data Description

In order to evaluate the performance of the proposed algorithms and aggregation strategies, it is preferred to apply the algorithm to multiple data instances. Furthermore, to investigate the influence of certain data characteristics, such as the inter-correlation of resources, we create data instances for which these characteristics differ. Simulating the data enables us to carefully study the effects on the performance of the algorithm. The simulated data is based on a transit rail network which will be analyzed in Section 5.1. In Section 5.2, we will elaborate on the data features which will be taken into account as resources in the column generation algorithm. Section 5.3 briefly explains how the dataset is structured and in Section 5.4, we elaborate on the correlation among the resources in the datasets.

5.1 The Network

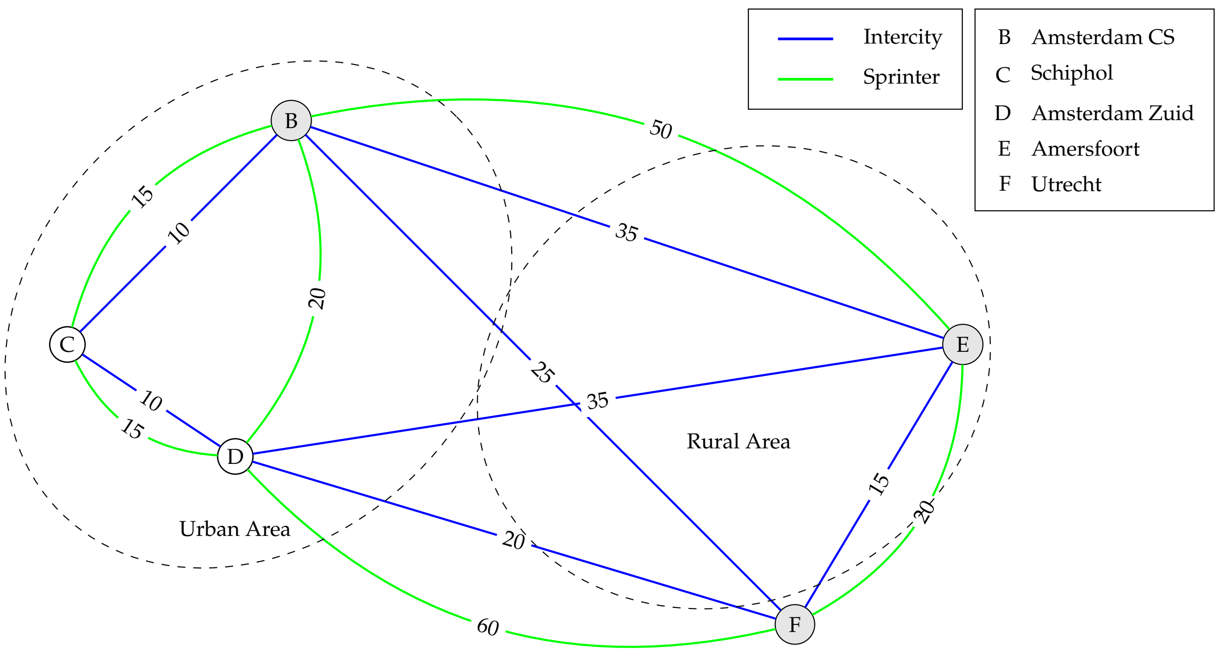

Figure 5.1: It shows the configuration of the stations and how they are connected with each other. The depot stations are indicated by the gray colored circles. Two different type of arcs can be distinguished in the figure which indicate different type of trains. The travel time corresponding to each arc in the network is given in minutes. The dashed ellipsoids indicate the type of area in which the stations are located.

We consider the following rail transit network (see Figure 5.1) for the construction of our simulated data. The selected network is a small part of the entire rail network of the Netherlands. We tried to make the case as illustrative as possible by using the actual traveling times, number of intermediate stops and frequencies as performed by the Netherlands Railways (NS). However, we simplified the network a bit and adjusted some features of the network in order to emphasize on certain characteristics.

We distinguish two different areas in the network. First of all, we consider the area of Amsterdam Central Station, Schiphol and Amsterdam Zuid as the *urban* area which has large commuter flows with relatively short distances. The fact that Schiphol is located in this area means that often trains are packed with tourists and they are in general not familiar with the Dutch railway system. This possibly results in more intensive work shifts for the NS personnel in this specific area. Furthermore, when traveling from Amsterdam Zuid to Utrecht, the train also stops at the station near the stadium of football club Ajax. Typically, these trips have a relative high risk of encountering aggressive passengers during match days. The other area we distinguish, is located between Amsterdam and Utrecht. This is mostly rural area in the sense that a large part of the area is used for agricultural purposes. The travel distances are somewhat larger and there are not many intermediate stops. The trips with larger travel distances are in general performed by an Intercity train (long distance train). The Sprinter train (regional train) performs most of the short distance trips (often in urban area).

The following tables show the travel distances in minutes, the number of intermediate stops between the stations and the frequency of each trip. The tables present the information for both the Intercity and Sprinter trains. Note that '−' means that there is no connection between these stations for that specific train type.

Table 5.1: Travel distances (min) for Intercity (left) and Sprinter (right) trains.

	B	C	D	E	F
B	x	x	x	x	x
C	10	x	x	x	x
D	–	10	x	x	x
E	35	–	35	x	x
F	25	–	20	15	x

	B	C	D	E	F
B	x	x	x	x	x
C	15	x	x	x	x
D	20	15	x	x	x
E	50	–	–	x	x
F	–	–	60	20	x

Table 5.2: Number of intermediate stops for Intercity (left) and Sprinter (right) trains.

	B	C	D	E	F
B	x	x	x	x	x
C	0	x	x	x	x
D	–	0	x	x	x
E	1	–	1	x	x
F	1	–	1	0	x

	B	C	D	E	F
B	x	x	x	x	x
C	3	x	x	x	x
D	6	3	x	x	x
E	10	–	–	x	x
F	–	–	11	4	x

Table 5.3: The frequency of each trip (per hour) for Intercity (left) and Sprinter (right) trains.

	B	C	D	E	F
B	x	x	x	x	x
C	1	x	x	x	x
D	–	2	x	x	x
E	2	–	2	x	x
F	4	–	2	4	x

	B	C	D	E	F
B	x	x	x	x	x
C	4	x	x	x	x
D	2	5	x	x	x
E	2	–	–	x	x
F	–	–	2	3	x

The dataset is in principle obtained by using (a part of) the actual passenger railway schedule performed by the NS. However, we made some adaptations which also involve adding or removing certain connections. In case we added a connection, we used the distribution of the number of stops shown in Table 5.4. It is observed from the actual railway schedule, that on average an Intercity train stops every 10 minutes when it performs a shift in the Randstad and it stops every 20 minutes in the areas outside of the Randstad. Therefore, we use 15 minutes for the average travel time between two consecutive stops for Intercity trains in our simulated dataset.

Table 5.4: Distribution of the number of stops

Duration (min)	0	5	10	15	20	25	30	35	40	45	50	55	60
Intercity	0	0	0	1	1	1	2	2	2	3	3	3	4
Sprinter	0	1	2	3	4	5	6	7	8	9	10	11	12

5.2 Resources

In the previous section, we described the network used for the construction of our dataset. An important aspect of our dataset are the resources. The features of the data which we consider to be the resources are listed below:

- Type of train (Intercity / Sprinter),
- Number of stops,
- The risk of encountering any aggression,
- Double-decker vs single-decker train.

We already elaborated on the first two resources in Section 5.1. The possibility of encountering aggression is another important feature of our data. Unfortunately, we do not have the permission to access the exact data from NS concerning aggression related incidents. Therefore, we simulated this data based on the information we could find about the different regions. For each trip, we assign a value of 1 in case it has a relative high risk for the NS personnel to encounter any aggression and we assign a value of 0 in case the trip has a relative low risk. The possibility of encountering aggression during a duty is specified based on the overall probability which can be calculated by the total time of a duty in which a high risk is measured divided by the total time of the duty.

Table 5.5: Trips with high risk of aggression (1) and low risk (0) for Intercity trains (left) and Sprinter trains (right).

	B	C	D	E	F
B	x	x	x	x	x
C	1	x	x	x	x
D	–	1	x	x	x
E	0	–	0	x	x
F	0	–	1	1	x

	B	C	D	E	F
B	x	x	x	x	x
C	1	x	x	x	x
D	1	1	x	x	x
E	0	–	–	x	x
F	–	–	0	1	x

Table 5.5 gives the risk of encountering aggression during each trip. The trips with a start-station and end-station located in the region of Amsterdam or in the urban area in general, are considered to be trips with a relative high risk of aggression. Furthermore, the intercity trip between Amsterdam Zuid and Utrecht is also considered to be a trip of high risk as it has an intermediate stop at Amsterdam Bijlmer-Arena (the station located near the stadium of football club Ajax). The equivalent sprinter trip does not stop at Amsterdam Bijlmer-Arena station and is therefore not considered to be a high risk trip.

The final feature of the dataset which we take into account as a resource, is the type of train. A trip can be performed by either a double-decker train or a single-decker train. This is simply copied from the actual NS schedule as it is at the moment of writing this thesis. The table for sprinter trains consists of zeros as all Sprinter trains are built with a single deck, see Table 5.6.

Table 5.6: Trips which are performed by a double-decker train (1) or single-decker train (0) for Intercity trains (left) and Sprinter trains (right).

	B	C	D	E	F
B	x	x	x	x	x
C	1	x	x	x	x
D	–	1	x	x	x
E	0	–	0	x	x
F	1	–	1	0	x

	B	C	D	E	F
B	x	x	x	x	x
C	0	x	x	x	x
D	0	0	x	x	x
E	0	–	–	x	x
F	–	–	0	0	x

5.3 Dataset

In the proposed network, we consider three different depot stations: Amsterdam Central, Amersfoort and Utrecht Central Station. The NS personnel should start at one of the depot stations and also end at the depot station from which it started. Based on the given frequency of each trip, a list can be constructed consisting of all the trips which should be performed within one hour. The final dataset is basically a list of trips including all of the corresponding resource information. The size of the dataset can be changed into a two hour dataset by simply copying this list and adjusting the starting and ending time. Table 5.7 displays a few lines of the dataset.

Table 5.7: An example of how the dataset looks like.

Trip nr.	Start time	End time	Start-station	End-station	Sprinter	Decker	Stops	Aggression
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
23	07:26	07:41	F	E	0	0	0	1
24	07:48	08:03	F	E	0	0	0	1
25	07:21	07:36	B	C	1	0	3	1
26	07:51	08:06	B	C	1	0	3	1
27	07:01	07:16	C	B	1	0	3	1
28	07:31	07:46	C	B	1	0	3	1
29	07:17	07:32	C	D	1	0	3	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

However, it will not be possible to find a feasible solution which covers all trips if we simply consider the complete list (or multiple copies of the schedule) as mentioned above. In Table 5.7, one could observe that trip 27 consists of a Sprinter train which should depart from Schiphol at 7:01 and arrive 15 minutes later at Amsterdam Central station. In case our time horizon starts at 7:00, we have a problem as Schiphol is not a depot station and no train could be at Schiphol within 1 minute. Therefore, the dataset is extended with an initialization phase at the start and a finalizing phase at the end of the dataset. This is basically a list of trips (a subset of the complete list mentioned above) which makes sure that there are enough trains departing at the depot stations and also enough trips which will end at the depot stations. In case we consider a three hour time horizon from 7:00 until 10:00, the initialization phase ensures that from 7:00 until 7:30 only the trips are included which depart from a depot station and the finalizing phase ensures from 9:30 until 10:00 only trips are included which end at one of the depot stations.

5.4 Correlation

In this research, we develop a resource aggregation based heuristic for the SPPRC. The aggregation of the resources can be based on the correlation of the resources and therefore, we evaluate the correlation of the resources in the dataset. The correlation matrix of the dataset based on the actual traveling schedule of NS is given in Table 5.8. This dataset has a medium overall correlation of the resources. A dataset with minimal correlated resources is obtained by a random permutation of the resources in the original dataset. Furthermore, a dataset is constructed in which the resources are strongly related. This is done by manually adjusting the trips such that the dataset contains a lot similar trips with respect to the resources. The correlation matrices of the datasets with minimal and maximal correlated resources are given in Tables 5.9 and 5.10.

Table 5.8: Correlation matrix (med)

	Type	Decker	Stops	Aggression
Type	1.0000	-0.5461	0.9607	0.3039
Decker	-0.5461	1.0000	-0.4558	-0.3199
Stops	0.9607	-0.4558	1.0000	0.2550
Aggression	0.3039	-0.3199	0.2550	1.0000

Table 5.9: Correlation matrix (min)

	Type	Decker	Stops	Aggression
Type	1.0000	-0.0157	0.0437	0.1071
Decker	-0.0157	1.0000	-0.0583	0.0747
Stops	0.0437	-0.0583	1.0000	0.0309
Aggression	0.1071	0.0747	0.0309	1.0000

Table 5.10: Correlation matrix (max)

	Type	Decker	Stops	Aggression
Type	1.0000	-0.9079	0.8777	1.0000
Decker	-0.9079	1.0000	-0.7938	-0.9079
Stops	0.8777	-0.7938	1.0000	0.8777
Aggression	1.0000	-0.9079	0.8777	1.0000

6 Computational Results

In this chapter, we present the computational results of this research. We apply the algorithms and aggregation strategies to the different datasets. All algorithms have been implemented in Java using Eclipse version 4.5.3 and IBM ILOG CPLEX 12.8.0 as LP solver. A HP EliteDesk 705 G3 SFF desktop with a 3.5 GHz AMD PRO A6-9500 R5 processor with 16.0 GB RAM is used to perform the computational experiments. The Java program is initialized at the start of each experiment and no other programs or non-standard background processes were running during the course of the experiments.
The remainder of this chapter is organized as follows. In Section 6.1, the experiment setup is discussed. We present and analyze the results of the experiments in Section 6.2. Finally, we analyze the results of the experiments in a bit more detail by evaluating the performance of the aggregation strategies in Section 6.3.

6.1 Experiment Setup

Before we are able to present and analyze the results of the experiments, it is necessary to elaborate on the different settings and parameters. We want to evaluate the effect of the aggregation strategies for each instance using different combinations of parameters. The parameter settings depend on the considered instance and will be discussed in more detail in Section 6.1.2 and 6.1.3. First in Section 6.1.1, we create some structure by introducing a five-character codename for each experiment.

6.1.1 Instances

We use 'XYZ' to denote the instances used in the experiments in order to be able to easily distinguish the differences between the various instances. Each character points out a certain specification of the experiment.

$$X = \begin{cases} P & \text{in case we aggregate resources using PCA,} \\ A & \text{if the resources are aggregated by ad-hoc method,} \\ R & \text{in case we simply take into account the original resources.} \end{cases}$$

The second character, Y, represents the number of 'aggregated' resources. For example, $P2$ denotes for two aggregated resources constructed by PCA (two principal components) and $R2$ means that we consider two of the original resources. The third character, Z, indicates the type of the dataset used. Here, we refer to the degree of resource correlation for which we have three options: min, med or max.

6.1.2 Upper Bound on Regular Resources and Optimality Tolerance

The goal of the experiments is to evaluate the performance of the algorithm and the applied aggregation strategy. We introduce an upper bound on the resources in order to influence the weighted resource averages of the generated shifts. In this way, we can force to generate shifts which have comparable averages such that we obtain fair solutions concerning the workload distribution. The bound on a particular resource is determined by the weighted instance average and some constant α. The constant α is multiplied by an estimate of the standard error corresponding to the mean of the weighted resource. In this way, we account for the different degrees of variation and we make sure the upper bounds on the resources are proportionally adjusted. The upper bound on the regular resources is defined as

$$UB_r^k = \overline{R}^k + \alpha s_{\overline{r}}^k = \frac{\sum_{i\in M} r_i^k T_i}{\sum_{i\in M} T_i} + \alpha s_{\overline{r}}^k, \qquad k \in \{1, 2, ..., K\},$$

with $s_{\overline{r}}^k = \frac{s^k}{\sqrt{n}}$ where $s_{\overline{r}}^k$ is the approximation of the standard deviation of the k-th resource mean. Furthermore, s^k is the standard deviation of resource k and n is the number of observations (trips in the dataset). If we decrease α, the bounds will become tighter and the feasible region is likely to shrink. This possibly causes the optimal objective value to increase, but on the other hand an improvement of the fairness of the solution is expected. A trade off can be made between the improvement of the fairness and the increase of the total costs. The percentage of increase in the optimal objective value is called the optimality tolerance. The optimality tolerance is expressed by ε and is calculated as follows:

$$\varepsilon = \frac{z_r - z_{opt}}{z_{opt}},$$

where z_{opt} is the optimal objective value in case we do not consider any bounds UB_r^k on the resources and z_r is the optimal objective value of the restricted problem. Notice the relation between the parameter α and the optimality tolerance ε. When we use a relatively small value of α, we typically observe a large optimality tolerance. We have evaluated the impact of the aggregation strategies for six different tolerance values: $\varepsilon \in \{0.03, 0.1, 0.5, 1.0, 2.0, 5.0\}$.

6.1.3 Upper Bound on Aggregated Resources

Similar to the previous section (see Section 6.1.2), we introduce an upper bound but this time on the aggregated resource(s). The domain of the upper bound $UB_{ar}^k \in [\underline{v}, \overline{v}]$ for which the aggregation algorithm has any impact on the performance, is determined by trial and error. We keep reducing the upper bound such that at some point all of the generated shifts will be infeasible with respect to the aggregated resource consumption. On the other hand, we keep increasing the upper bound until all possible shifts satisfy the upper bound and the feasibility of a shift solely depends on the 'general' resource constraints. In other words, if the upper bound on the aggregated resource is below a certain threshold $\underline{v}$ and above a certain threshold

$\overline{\nu}$ where $\underline{\nu} < \overline{\nu}$, the aggregated resource algorithm is of no added value (might only increase the computation time). These thresholds are determined for all k aggregated resources.

6.1.4 Benchmark Computation Time

The performance of the algorithm and aggregation strategy are compared with the benchmark computation time which is obtained by excluding the aggregated resource algorithm from the pricing framework. We observe that the computation time can differ among multiple runs despite the fact that we use the same settings and have similar conditions every run. Especially in the first few iterations, we observe some significant volatility (see Figure 6.1). The figure shows the runtime for 100 runs for the medium correlated dataset and Table 6.1 contains some statistics concerning the volatility of these benchmark runs. The observed volatility is probably due to the fact that Java has to complete some initialization processes at the beginning such as loading certain libraries and packages. Therefore, we decided to start each experiment with an arbitrary 20 'warming-up' runs. The mean value $\bar{b}$ of the last 80 benchmark runs will be used when comparing and evaluating the performance of an aggregation strategy for the experiments.

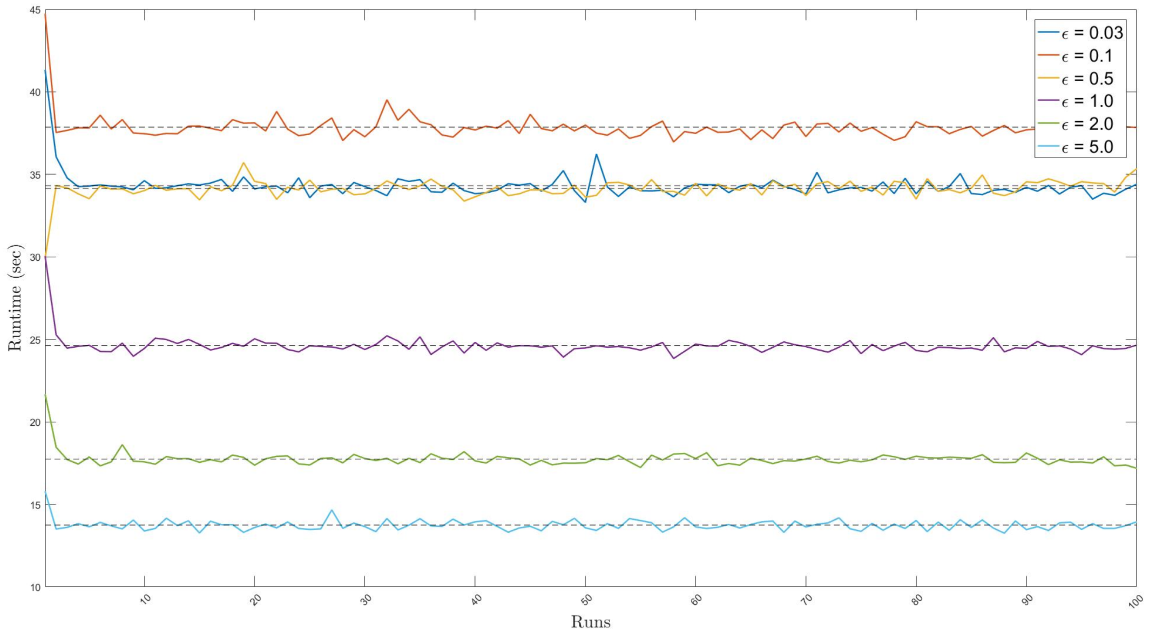

Figure 6.1: It shows 100 observations for the benchmark computation time for different values of the optimality tolerance ε. We observe that the first runs have a larger computation time due to the initialization processes of Java.

	$\bar{b}$	$\bar{b}$	s_b	s_b	$min\{b\}$	$min\{b\}$	$max\{b\}$	$max\{b\}$	size n	size n
$\varepsilon = 0.03$	34.31	34.19	0.83	0.43	33.31	33.31	41.33	36.23	100	80
$\varepsilon = 0.1$	37.87	37.79	0.82	0.46	36.97	36.97	44.74	39.52	100	80
$\varepsilon = 0.5$	34.14	34.18	0.58	0.38	29.91	33.38	35.72	35.34	100	80
$\varepsilon = 1.0$	24.61	24.54	0.61	0.25	23.84	23.84	30.05	25.22	100	80
$\varepsilon = 2.0$	17.75	17.70	0.47	0.22	17.19	17.19	21.68	18.19	100	80
$\varepsilon = 5.0$	13.74	13.73	0.33	0.27	13.25	13.25	15.79	14.67	100	80

Table 6.1: The statistics correspond to the benchmark runs performed with the dataset with maximum correlation. The statistics in the white columns corresponds to all 100 observations and the statistics in the gray colored columns correspond to the last 80 observations.

It can be observed that the mean and standard deviation of the computation time in general decrease when we exclude the warming up runs. Furthermore, the absolute difference between the minimum and maximum value of the runtime decreases as well. The tables and figures with information on the benchmark runs of the other datasets are given in Appendix B.

6.2 Results of the Experiments

In this section, we will present and analyze the results of the experiments.

6.2.1 P1 Experiments

We use the first (most important) principal component for the construction of the aggregated resource in case of the $P1$ experiments. Figures 6.2, 6.3 and 6.4 show the performance of the algorithm for different values of the upper bound UB_{ar} and the optimality tolerance ε. The red area represents the benchmark computation time. In Figure 6.2, it is observed that in general the aggregation algorithm does not outperform the benchmark algorithm for the dataset with minimum correlation. The surface plot can be described as a 'hill' parallel to the ε-axis. The upper bound thresholds $\underline{v}$ and $\overline{v}$ are located on the downsides of the hill. At the left hand side of the hill, the computation time approaches the benchmark running time which is explained by the fact that the upper bound is very tight at this point and it will not be possible to generate a shift which satisfies the aggregated resource constraints. Therefore, the pricing framework can be considered to be applied without the aggregation algorithm which is the same setup used for the benchmark computation times. For the right hand side of the hill it holds that every shift is feasible according to the aggregated resource constraints. Therefore, the algorithm is not of any added value in this case and it only causes the computation time to increase. In between the thresholds, we observe an increase of computation time as expected since we use a method based on the correlation of the resources for an instance with minimum correlation.

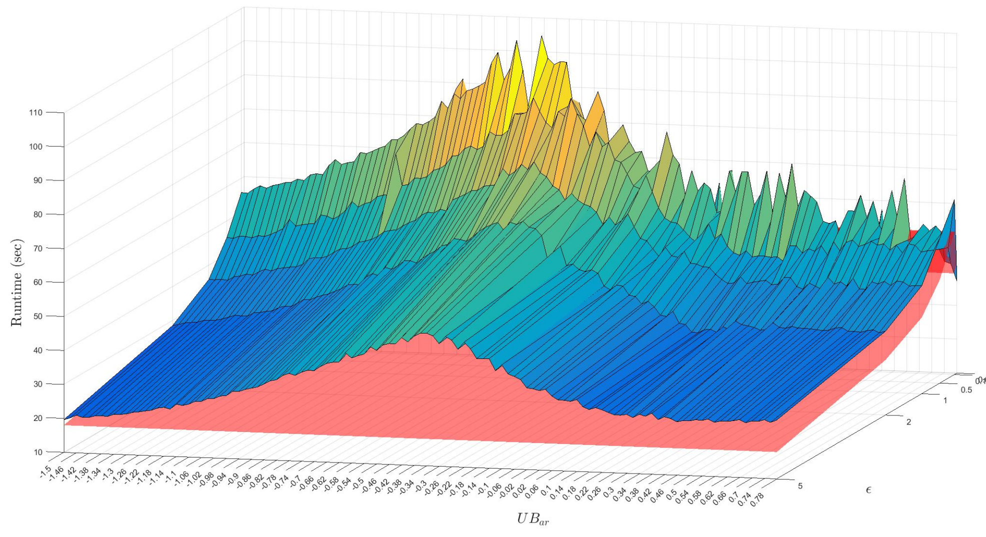

Figure 6.2: The results of experiment $P1min$. We observe that the aggregation algorithm does not outperform the benchmark algorithm.

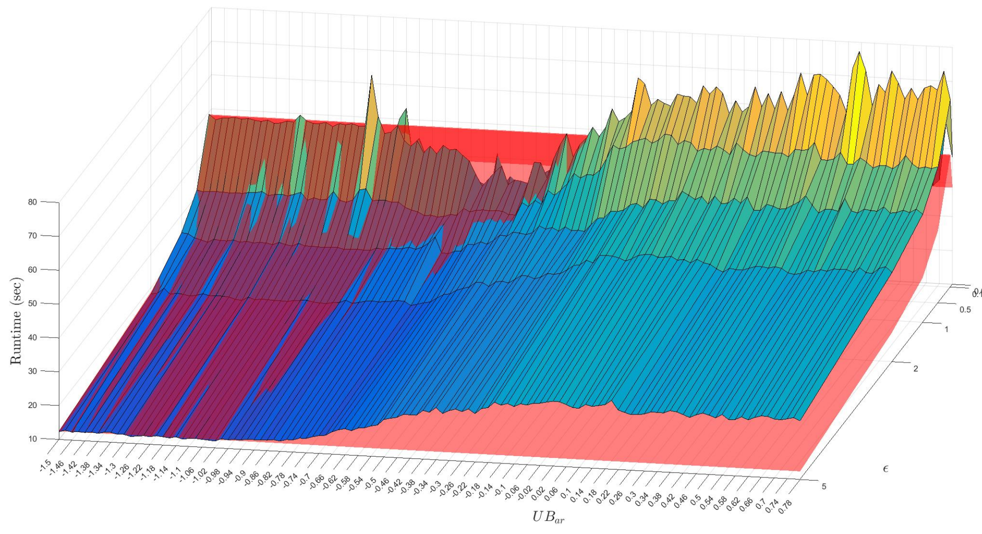

Figure 6.3: The results of experiment $P1med$. We observe that the aggregation algorithm does outperform the benchmark algorithm for $\varepsilon \in [0.1, 0.5]$ and $UB_{ar} \in [-0.8, -0.4]$.

The results of experiment $P1med$ are displayed in Figure 6.3. It appears that the aggregation strategy has a positive impact on the computation time only for relatively small values of the upper bound UB_{ar} in combination with small values of the optimality tolerance ε. Interestingly, we only observe the left hand side of the 'hill' compared to Figure 6.2. For large values of the upper bound UB_{ar}, the aggregation algorithm considers all shifts to be feasible. When lowering the upper bound UB_{ar}, the aggregation algorithm becomes relevant as it actually starts to reject shifts based on their feasibility. At this point, we consider the aggregation algorithm and the standard labeling algorithm to be 'working together'. Apparently, the computation time does not increase when this happens in case of the medium correlated instance in contrast to the minimum (and maximum) correlated instance. This is an indication that both algorithms work well together for the medium correlated instance. For the medium correlated instance, we achieve a reduction in computation time up to 34% for a parameter setting with $\varepsilon \in [0.1, 0.5]$ and $UB_{ar} \in [-0.8, -0.4]$ [3]. Note, the red surface on the left hand side of the hill does not indicate a positive impact by the aggregation strategy. Both algorithms have approximately the same results, but we observe that the benchmark algorithm has slightly larger computation times due to some volatility in the experiment. This also holds for the $P1max$ experiment (see Figure 6.4).

Figure 6.4, corresponding to experiment $P1max$, is quite similar to Figure 6.2. We again recognize the typical 'hill' shape in the surface plot which is comparable to the results of the $P1min$ experiment. However, we observe a much more flat surface in the direction of the ε-axis for this instance. Furthermore, the left and right hand side of the hill are much steeper in the direction of the UB_{ar}-axis. These differences can be explained by the high degree of correlation between the resources. Consider for example an instance consisting of 4 unique trips with respect to the distribution of resources. The instance contains many copies of these trips, but each trip has a different starting and ending time. The possible combinations of trips which result in a feasible shift concerning the resource feasibility, is now relatively limited. Therefore, the solution space is smaller compared to the other instances.

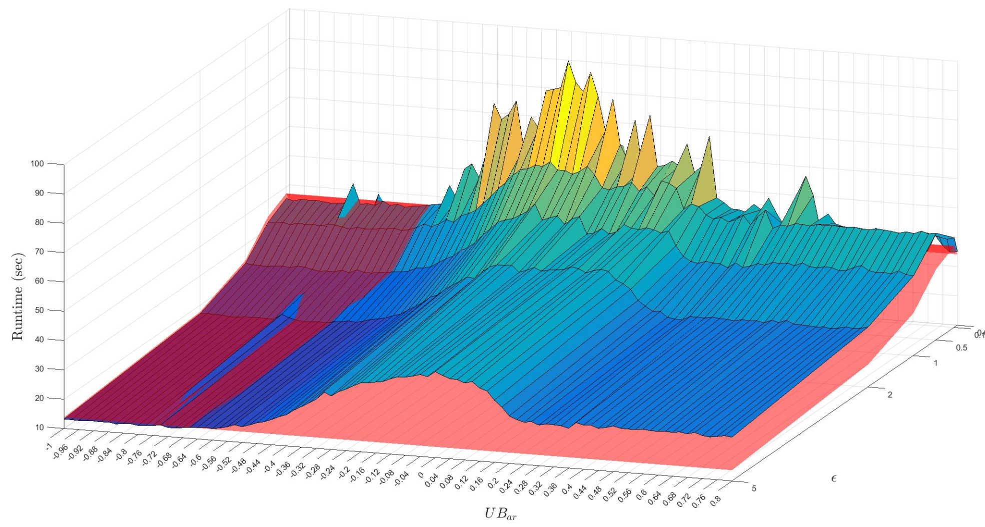

Figure 6.4: The results of experiment $P1max$. We observe that the aggregation algorithm does not outperform the benchmark algorithm.

[3]Note, that we only evaluated the computation time for certain combinations of parameters. However, because the intervals are rather closely positioned, interpolation of these values will give a good representation of its true value.

6.2.2 P2 Experiments

In the $P2$ experiments, we also include a second aggregated resource using the second principal component. We investigate the performance for different values of the upper bounds UB^1_{ar} and UB^2_{ar}. Figure 6.5 shows the results of experiment $P2med$ using 6 subplots corresponding to different values of ε. We observe that the aggregated resource corresponding to the first principal component has positive impact on the computation time for small optimality tolerances ($\varepsilon \in [0.03, 0.5]$), but this effect diminishes as ε increases. It seems like the aggregated resources corresponding to the second principal component does not have an additional impact in case of the medium correlated instance.

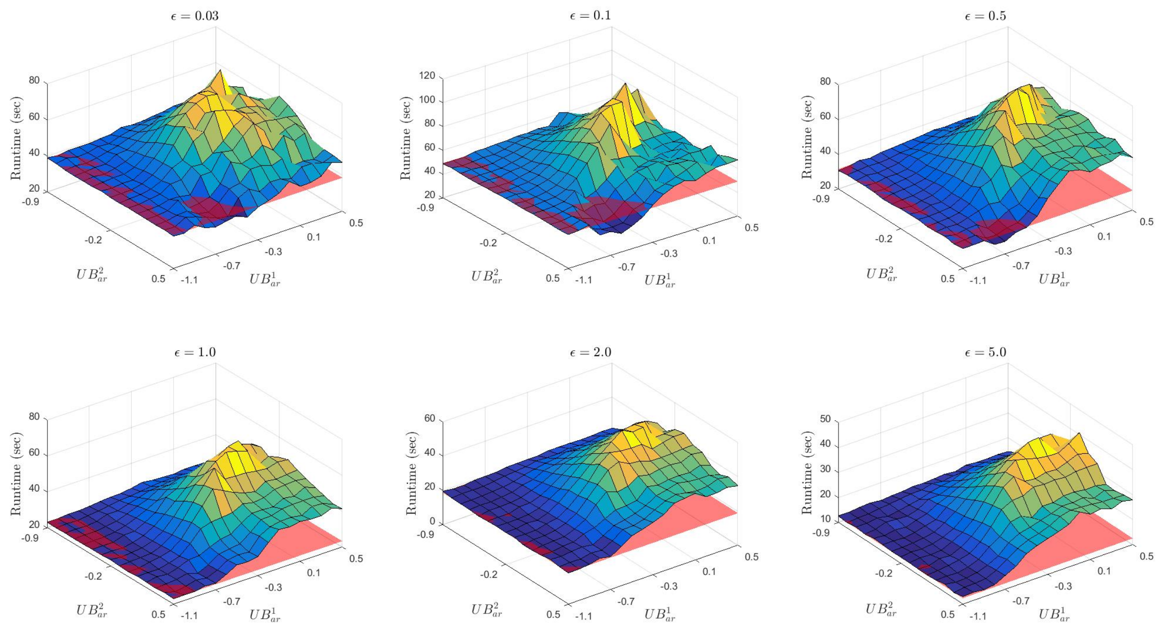

Figure 6.5: The 6 subplots display the result of experiment $P2med$. Each subplot shows the performance of the algorithm for different values of the upper bounds UB^1_{ar} and UB^2_{ar}.

We also evaluated the results of experiments $P2min$ and $P2max$, but both experiments do not have a positive impact on the computation time. The Figures C.1 and C.2 can be found in Appendix C. For most of the sub figures, we observe similar aspects as described for the $P1$ experiments. We again observe a 'hill', however for experiment $P2min$ and $P2max$ the hill has got a 90 degree turn due to the second aggregated resource (see Figure C.1 and C.2).

In case of the PCA aggregation strategy, we aim to select an appropriate number of principal components such that most of the variance is explained. A scree plot and/or a plot with the proportion of variance explained using i components, can be used to evaluate the appropriate number of components (see Figure 6.6). It can be observed that in case of the instances with medium and maximum correlation, the relative difference between the first and second most important component is quite large. Furthermore, the cumulative proportion of variance explained by the first principal component is already reasonably large. This possibly explains the fact that we do not observe an additional impact when we incorporate the aggregated resources corresponding to the second principal component as a large proportion of the data variance is represented by the first component.

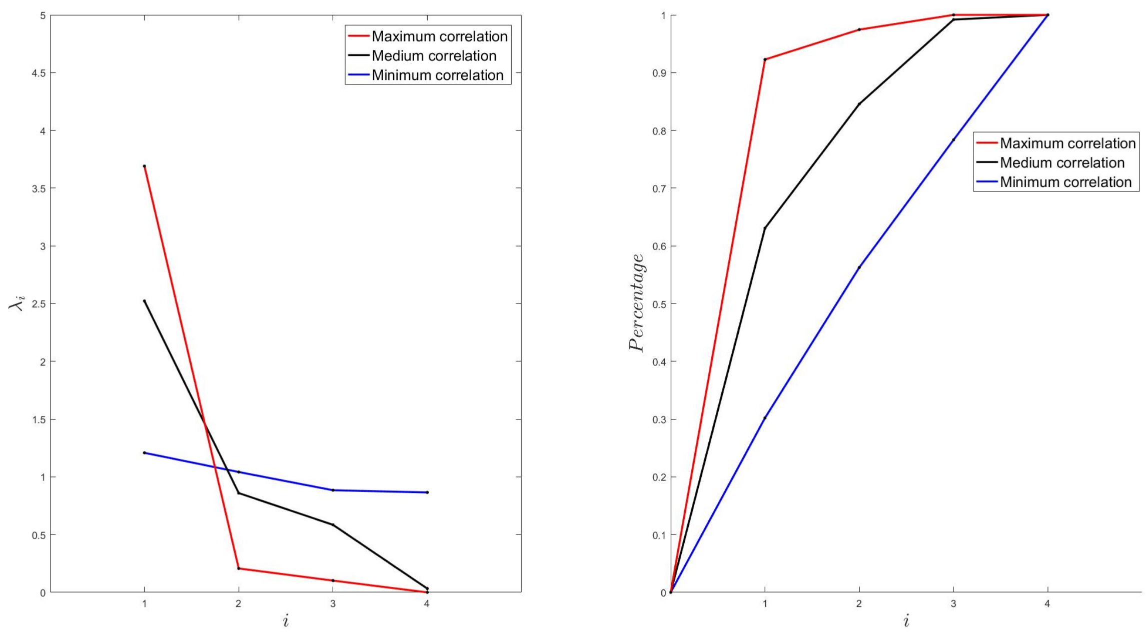

Figure 6.6: Left: a combined scree plot for the associated instances. Right: the corresponding distributions of the cumulative proportion of variance explained by *i* principal components.

6.2.3 A1 experiments

We analyze the performance of the ad-hoc aggregation strategy using the results of the $A1$ experiments. In the analysis, we only considered the medium correlated dataset based on previous experiments. We aggregated the resources using a weighting function specified in Section 4.4.2. The function can be applied with different values of n which has influence on the size of the weights. Figure 6.7 shows the results obtained for experiment $A1med$. The subplots each correspond to a different value n. It can be observed that the proposed aggregation strategy has a positive impact on the computation time in case we set $UB_{ar} \in [0, 0.4]$ in combination with an optimality tolerance of $\varepsilon \in [0.03, 1.0]$. Apparently, the parameter n does not have a exceptional influence on the results. Based on these experiments, the aggregation strategy performs best for $n \geq 1.5$ which can result in a reduction of the computation time up to 36%.

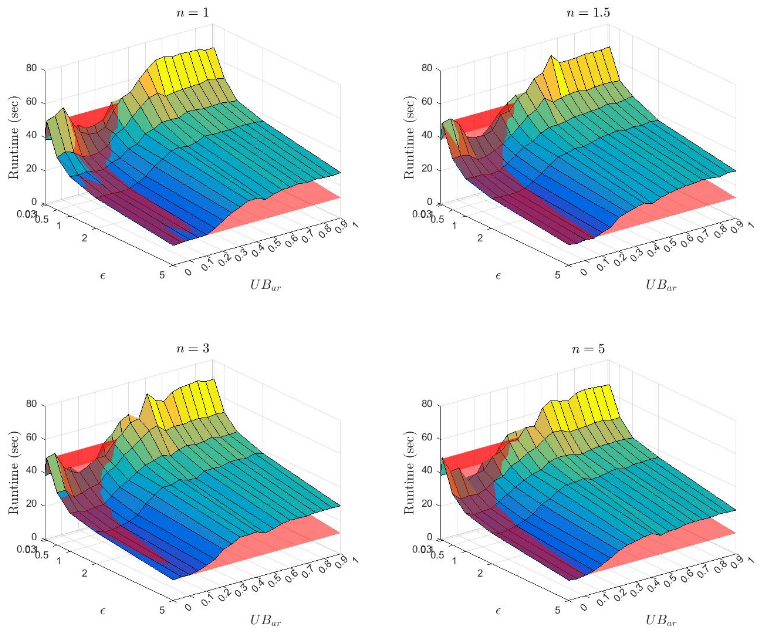

Figure 6.7: The 5 subplots display the result of experiment $A1med$. Each subplot shows the performance of the algorithm for different values of the power n used to determine the weights in the ad-hoc aggregation strategy.

6.2.4 R1 Experiments

In the $R1$ experiments, we consider a single (non-aggregated) resource. The idea is that if the considered resources in a certain data instance are related, it might be that in case we only set a restriction on one of the resources it automatically ensures that the other resources will be distributed proportionally. The resource which is most related to the other resources will be the most likely resource for these experiments. We did not consider the data instance with minimum correlation since such a resource does not exist for the considered instance. Figure 6.8 shows the results if we only take into account the resource *Train Type* for both the medium and maximum correlated instance.

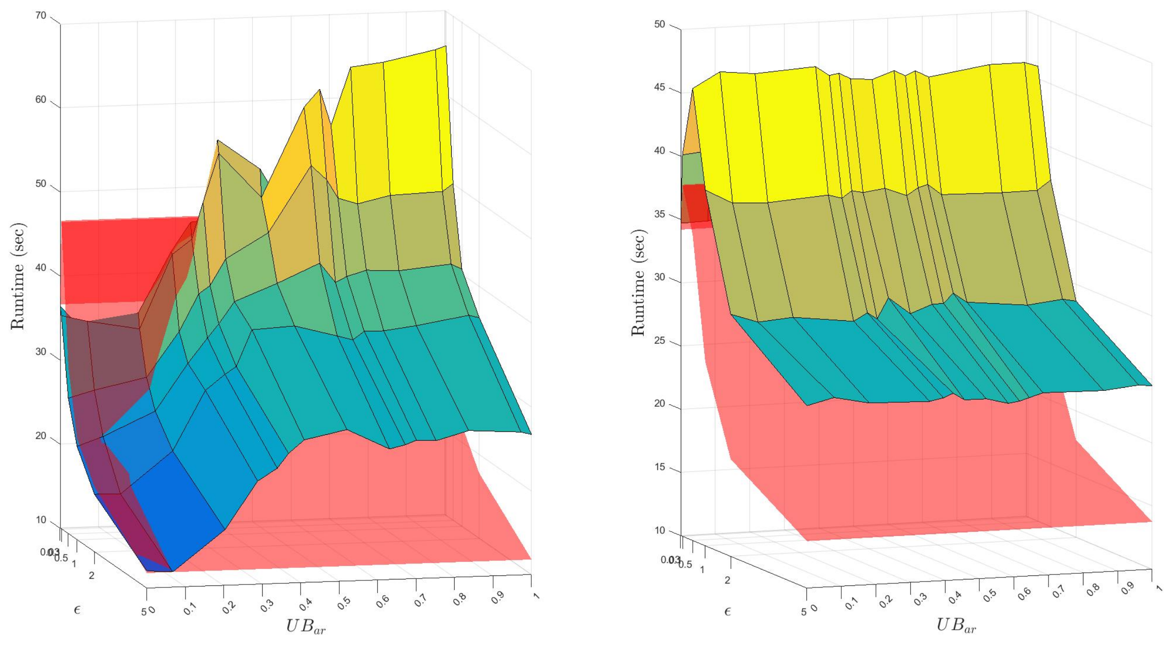

Figure 6.8: The computational results for the $R1med$ experiment (left) and the $R1max$ experiment (right) in case we only take into account the *Train Type* resource.

It can be observed in Figure 6.8 that for the medium correlated instance, we in fact find improved results with respect to the computation time. We find that a reduction of the computation time up to 28% is achieved for an optimality tolerance $\varepsilon \in [0.1, 0.5]$ and the upper bound $UB_{ar} \in [0, 0.35]$. Surprisingly, it appears that this strategy does not have any positive impact for the maximum correlated instance.

6.2.5 Fairness of the Results

The aim is to generate resource feasible shifts such that the NS personnel at each depot station has an equally fair set of shifts to perform. The fairness is an important feature of the obtained solution. We measure the fairness of a solution by a criterion based on the average resource consumption of the particular instance. We determine the error by means of the squared absolute difference between the instance average and the average resource consumption of all shifts assigned to a specific depot station. We define the following fairness criterion:

$$\delta_{error} = \sum_{d \in D} \gamma_d \sum_{k \in K} |\overline{R_d}^k - \overline{R}^k|^2, \tag{6.1}$$

in which

$$\overline{R}^k = 100 \frac{\sum_{i \in M} r_i^k T_i}{\sum_{i \in M} T_i},$$

and

$$\overline{R_d}^k = 100 \sum_{j \in N_d} x_j \left(\frac{\sum_{i \in M_j} r_i^k T_i}{\sum_{i \in M_j} T_i} \right).$$

The set N_d contains all shifts assigned to a particular depot station d and the set N is defined as $N = \bigcup\limits_{d \in D} N_d$ which contains all the shifts considered in the obtained solution. x_j denotes the amount of shift j considered in the solution. The set M_j contains all the trips corresponding to the shift j. Furthermore, γ_d is a multiplication factor to correct for the unbalanced distribution of shifts among the depot stations. The values of γ_d are the percentages of the total contribution of each depot with respect to the total number of shifts and these values vary depending on the instance and the parameter setting. Finally, we consider resources $k \in \{1, 2, .., K\}$ and T_i represents the duration of trip i.

It is interesting to investigate what happens when we increase the value of ε with respect to the fairness. In Table 6.2, it is shown that at some point the solution averages are getting close to the instance average. However, further increasing the value of ε causes the solution averages to reduce even more and start to deviate again from the instance averages. It becomes clear that in case we increase ε too much, the algorithm generates shifts which combined have a weighted resource average which is less than the instance average. This means that certain trips are used more than once in the final solution which result in extra costs. In practice it is up to the employer to decide how much to invest in the fair distribution of the work with respect to the preferences of the employees. It appears that in this case, we can already achieve our target relatively fast and large optimality tolerances are not prerequisite. This is also observed in Figure 6.9 in which the error distribution of all three instances is given. We prefer values of the optimality tolerance on the interval $[0.1, 2]$ in order to have a minimal fairness error. We observe large errors for a small optimality tolerance ($\varepsilon = 0.03$) which are reduced significantly after slightly increasing ε. However, for approximately $\varepsilon = 2$ the error starts increasing again.

The latter is not observed for the maximum correlated instance, but instead we observe the opposite behavior of the error. This can be explained by the fact that for this instance the trips all have different durations, but are very similar with respect to the corresponding resources. Therefore, it is apparently still possible to generate shifts which have a corresponding resource consumption which is close to the instance average in case we set the upper bounds very tight. For the other instances, the diversity of the trips (with respect to the associated resources) makes it more complicated to find feasible shifts when tightening the upper bounds on the constraints. Ultimately, for these instances it appears to be necessary to perform certain trips more than once in order to cover all trips.

	Sprinter	Decker	Stops	Aggression
Instance average	0.52	0.22	0.12	0.4
$\varepsilon = 0.03$	0.55	0.16	0.13	0.39
$\varepsilon = 0.1$	0.53	0.2	0.12	0.41
$\varepsilon = 0.5$	0.54	0.21	0.13	0.41
$\varepsilon = 1.0$	0.51	0.22	0.12	0.4
$\varepsilon = 2.0$	0.5	0.22	0.12	0.4
$\varepsilon = 5.0$	0.47	0.21	0.11	0.39
$\varepsilon = 10.0$	0.45	0.21	0.11	0.37

Table 6.2: The resource averages of the medium correlated instance for depot station B. Similar behavior is observed for the other depot stations.

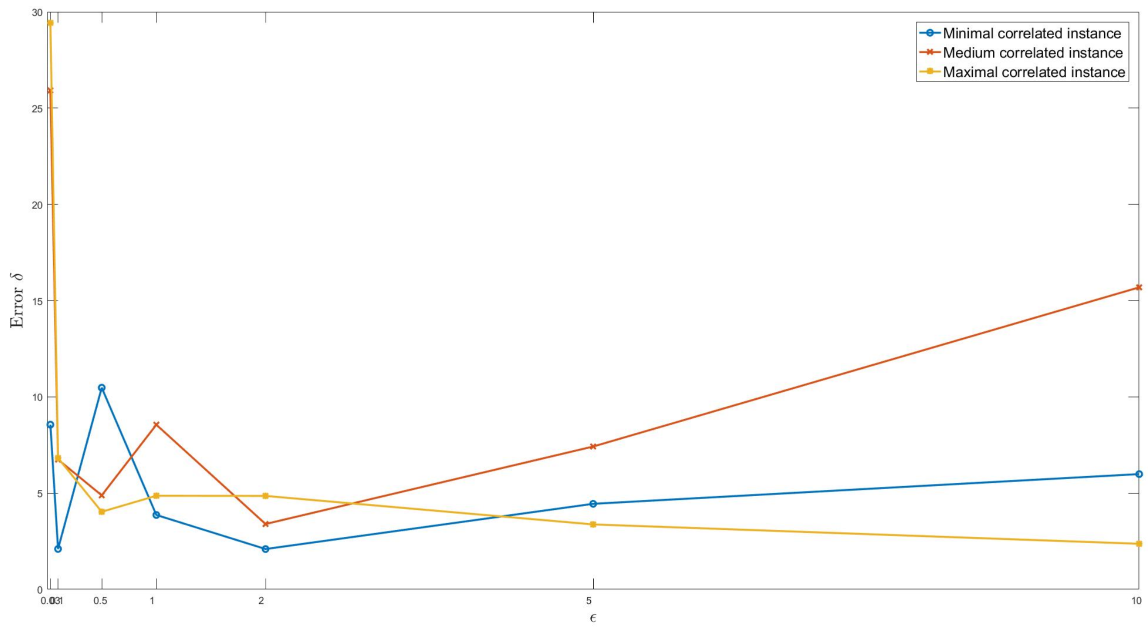

Figure 6.9: The fairness error determined with the fairness criterion (6.1) for different values of ε. The error is evaluated for all three instances.

6.3 Algorithm Performance

In this section, we evaluate the performance of the aggregation algorithm by looking into some of the details of the obtained results.

6.3.1 Tail-off Effect

Another remarkable observation which is observed in the figures corresponding to most of the experiments described above (Figures 6.2, 6.3, 6.4, 6.7 and 6.8), is the increase in computation time for $\varepsilon = 0.1$ compared to results for $\varepsilon = 0.03$. This is probably due to the fact that an optimality tolerance of 0.03 is very small and the corresponding bounds on the resources do not affect the original feasible region that much. If we choose $\varepsilon = 0.1$, it becomes slightly more difficult to find feasible shifts since the bounds become tighter. However, there are still many feasible shifts left, of which the majority has a comparable quality. This causes the computation time to increase as relatively a large number of iterations are necessary to obtain only a small improvement of the objective value, the so-called tail-off effect. This is confirmed if we have a closer look at objective value over the course of the column generation algorithm. In Figure 6.10, we show two examples in which we compare the behavior of the objective values over time for $\varepsilon = 0.03$ and $\varepsilon = 0.1$. If we consider example 1, it can be observed that at first both red lines approximately follow a similar path. However, it can be seen that the red dotted line has a much longer horizontal timespan compared to the red dashed line. The vertical sections in between the horizontal sections are caused by the switch of the depot station (pricing problem). The second subplot shows a similar example for $UB_{ar} = -0.4$. We noticed this behavior in most of the experiments.

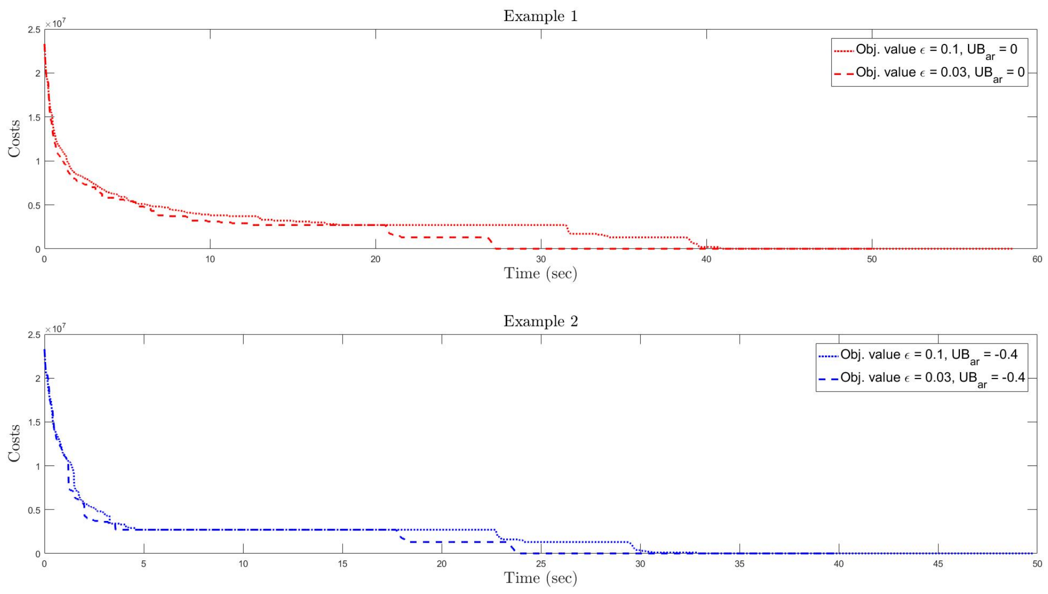

Figure 6.10: Two examples of the observed tail-off effect in case of the medium correlated instance.

6.3.2 Type I & II Errors

The aggregation algorithm should generate shifts which are feasible with respect to the constraints on the aggregated resources. If the aggregated resource is a good representation of the original resource, it also means that the shifts satisfy the (regular) resource constraints. An interesting output statistic is to what degree these constraints (on both types of resources) lead to similar outcomes. We distinguish between four different possibilities.

- The shift is feasible for both cases,
- The shift is feasible, however it is not feasible according to the aggregated resource constraints (type I error),
- The shift is not feasible, however it is feasible according to the aggregated resource constraints (type II error),
- The shift is not feasible for both cases.

Ideally, you would like to have the amount of type I and II errors as low as possible. Note, that we also want to find a minimal number of shifts which are infeasible according to both set of constraints, however we want to focus on the imparities between the aggregated resources and the regular resources for now. Each iteration (except the last) of the pricing framework results in a shift which is feasible according to both sets of resource constraints. Prior to the generation of this feasible shift, the algorithm possibly generated a number of shifts which have been rejected because of one of the reasons listed above. We generated a couple of figures in order to provide some insight in the performance of the aggregation algorithm for the $P1med$ experiment and we especially emphasize on the reason of rejection. The Figures 6.11, 6.12 and 6.13 are generated using the dataset with medium correlated resources, a fixed optimality tolerance ($\varepsilon = 0.5$) and different values for the upper bound UB_{ar} ($UB_{ar} \in [-0.6, -0.3, -0.1]$). A few additional figures can be found in Appendix D.
Let us consider Figure 6.11. The figure is a stacked bar plot in which the vertical axis gives the number of shifts and the horizontal axis displays the corresponding iteration number. Only the relevant iterations of the pricing framework are included in this analysis, that is, the iterations for which the aggregation algorithm ultimately generated a feasible shift. The total height of the stacked bars for iteration i corresponds to the total number of shifts which have been generated until a feasible shift is found. Note that there are two moments at which a sudden 'drop' is observed in the stacked bar plot due to the fact that we chose to split the pricing problems into multiple pricing problems each with a different depot station.

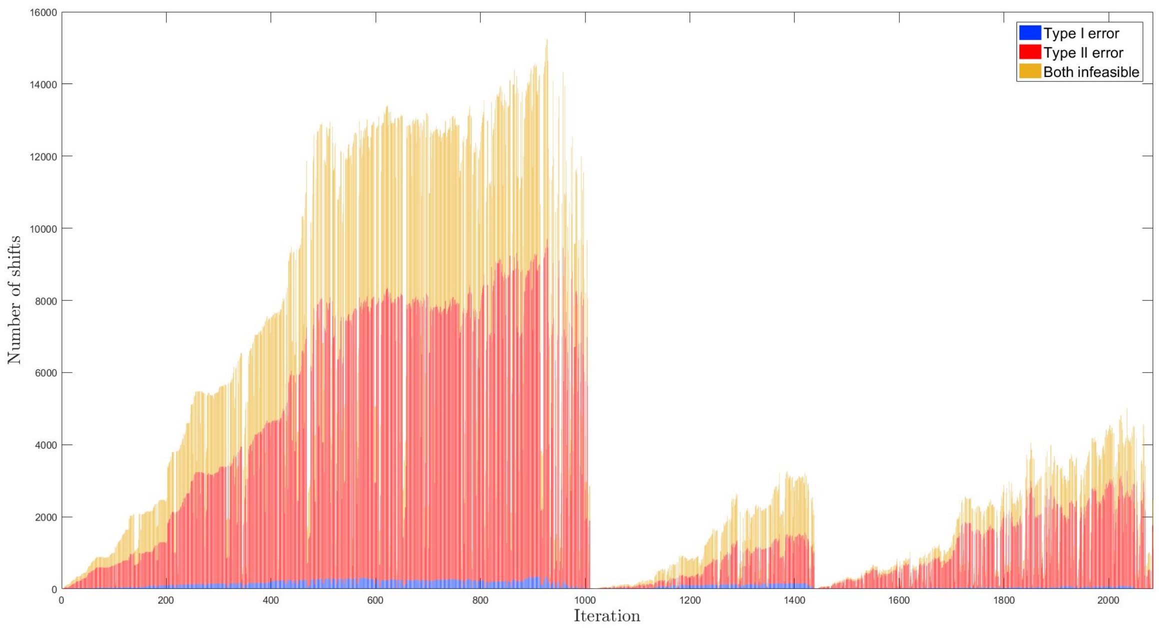

Figure 6.11: A stacked bar plot of the error types for $\varepsilon = 0.5$ and $UB_{ar} = -0.1$.

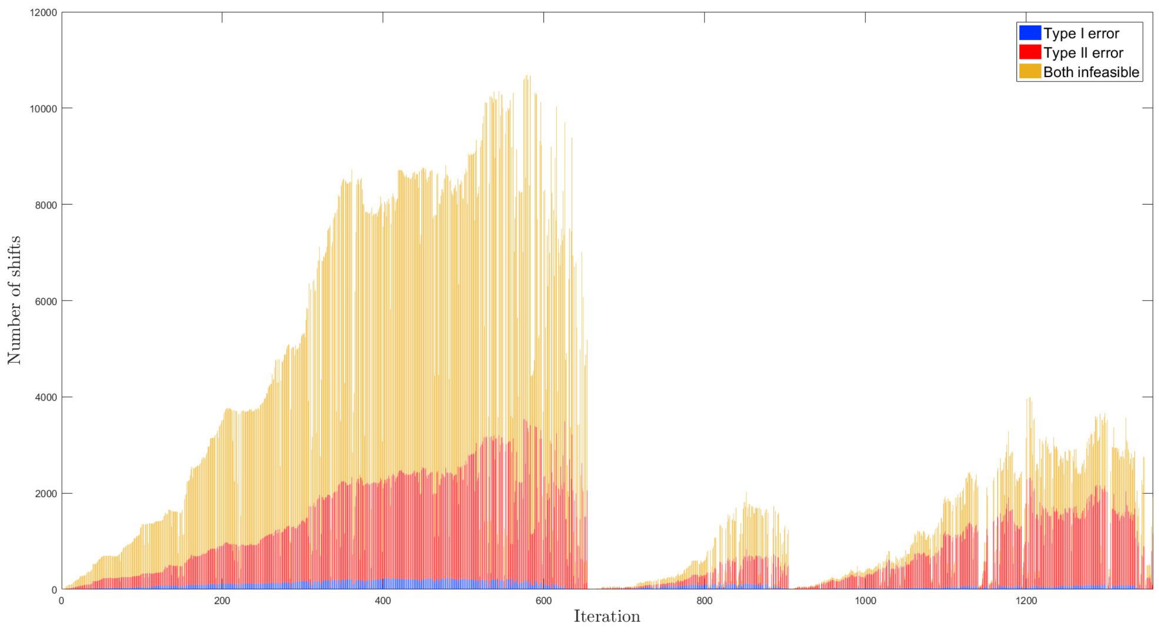

Figure 6.12: A stacked bar plot of the error types for $\varepsilon = 0.5$ and $UB_{ar} = -0.3$.

When comparing the figures, we notice a couple of interesting features. First of all if we compare Figures 6.11 and 6.12, we observe that the total number of shifts as well as the maximum number of shifts generated in a single iteration are somewhat reduced for Figure 6.12. Figures in the Appendix D show the number of errors made for each iteration. However, we standardized it in order to visualize the distribution of the cumulative proportion of the errors. The percentage of the type II errors is reduced significantly when decreasing the value of UB_{ar}, see Figure D.1 and D.2. Furthermore, it appears that the computation time decreases together with the decrease of the type II errors, see Figure D.4.

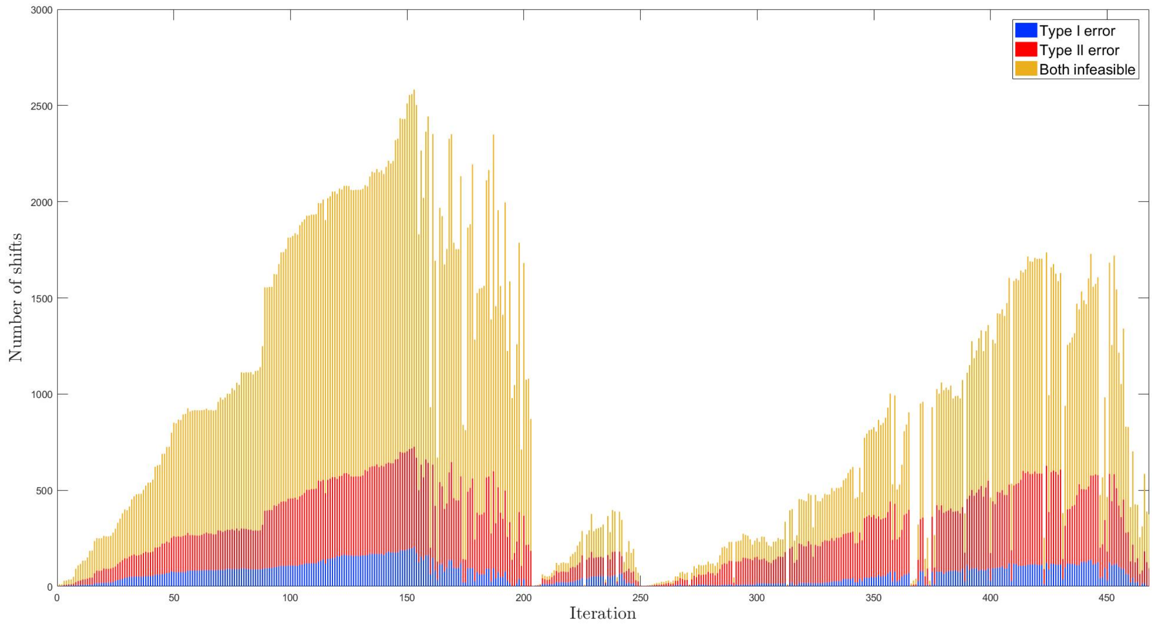

Figure 6.13: A stacked bar plot of the error types for $\varepsilon = 0.5$ and $UB_{ar} = -0.6$.

Secondly, if we study the differences between Figure 6.12 and 6.13 (using an even stricter upper bound $UB_{ar} = -0.6$), we observe the opposite effect of previous comparison. Obviously, the total number of generated shifts as well as the maximum number of shifts generated in a single iteration is not of similar proportions. Furthermore, the average percentage of type II errors for each iteration is comparable for both upper bounds, see Figure D.3. We also observe an increase in the percentage of type I errors compared to the previous figures. A interesting observation is the reasonably small percentage of type I errors among all three figures. When lowering the upper bound, it is apparently more likely a shift becomes infeasible for both sets of constraints instead of the occurrence of a type I error.

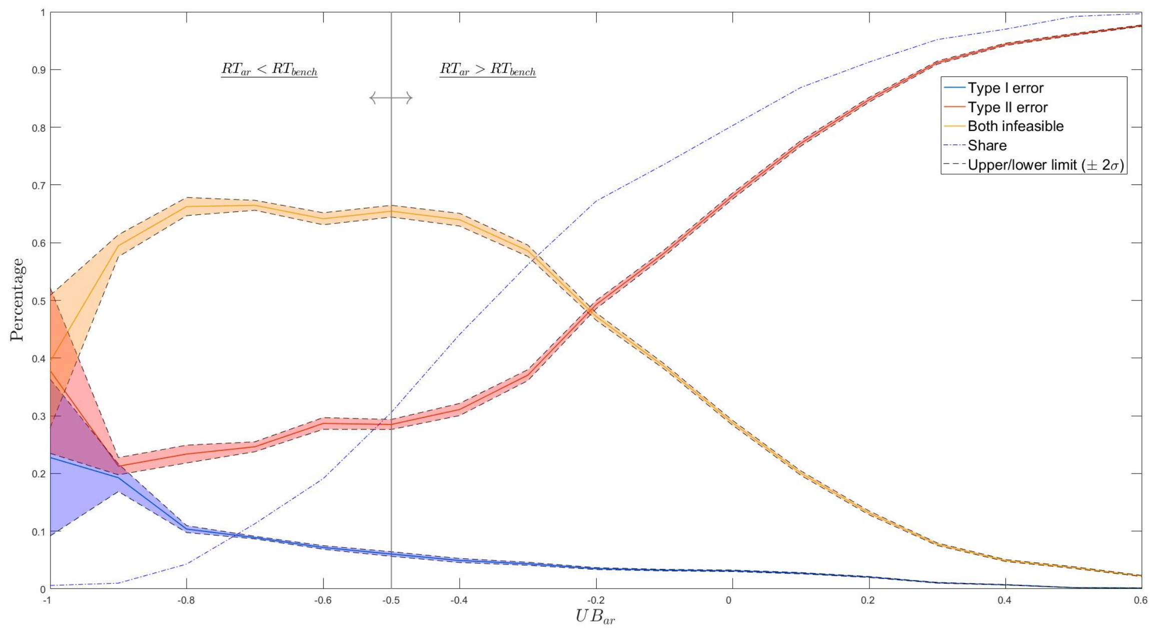

Figure 6.14: It shows the average percentage of the error types including the corresponding confidence intervals. The left (right) hand side of the vertical black line shows for which values of UB_{ar} the computation time of the aggregation algorithm RT_{ar} is smaller (larger) than the computation time of the benchmark algorithm RT_{bench}. The blue dashed line indicates the share (%) of the total number of shifts generated by the aggregation algorithm. The figure is generated for an optimality tolerance $\varepsilon = 0.5$.

In Figure 6.14, the percentage of the error types are displayed for different values of the upper bound. The confidence intervals are included since we are dealing with a different number of observations (generated shifts by the aggregation algorithm) for each value of UB_{ar}. The figure shows that the percentage of type I errors appears to reduce when increasing the upper bound, but remains fairly constant for $UB_{ar} \in [-0.4, 0.6]$. It is clearly visible that lowering the upper bound causes the percentage of the type II errors to reduce. The percentage of shifts being infeasible in both cases shows exactly the opposite behavior. The figure also includes a vertical black line which indicates for which values of the upper bound the aggregation algorithm performs better compared to the benchmark algorithm. Furthermore, the blue dashed line represents the percentage of shifts generated by the aggregation algorithm. We also evaluated this figure for other values of the optimality tolerance (see Figures D.5 and D.6) and we noticed that the vertical black line is located at the point where the type II errors and the 'both infeasible' line starts to increase and decrease respectively. At this point, it appears that the percentage of shifts which are 'both infeasible' is minimal 70%.

We want to emphasize on the fact that in general it becomes harder to find shifts which satisfy the aggregated resource constraints if we choose a smaller value for the upper bound UB_{ar}. Typically, the share of the aggregation algorithm in the generation of feasible shifts decreases when a stricter upper bound on the aggregated resource is used. Based on the insights in the performance of the aggregation algorithm, we distinguish a positive effect in case the percentage of type I and II errors is reasonably small and a negative effect otherwise. The share of the algorithm influences whether these effects become visible. The observed reduction in computation time is caused by a conjunction of both the effects and the total share of the algorithm. If a specific setting of the parameters leads to mostly positive effects for example, it is preferable to have a large share in the total number of generated shifts and vice versa.

7 Conclusions

This thesis addresses resource aggregation strategies and their implementation in a column generation framework used to solve a set-covering problem (SCP) with additional constraints. The corresponding pricing problem is mathematically formulated as a SPPRC and solved by a dynamic programming approach. Instead of considering the individual resources in the SPPRC, we developed resource aggregation strategies based on the correlation between the resources in order to investigate the possibility of solving the pricing problem more efficiently. A pricing framework consisting of multiple dynamic programming algorithms is proposed in which we can incorporate each of the developed resource aggregation strategies. The first strategy makes use of a principal component analysis whereas another strategy is described as an ad-hoc method. Lastly, we apply a basic strategy which simply considers a single (or multiple) non-aggregated resource(s).

A large variety of well-known real world problems such as scheduling or vehicle routing, can be formulated and solved mathematically using the SCP formulation. In literature the SCP is sparsely considered together with additional constraints in order to incorporate the preferences of the employees. The incorporation of the preferences enables us to obtain equally fair solutions which can be of major importance when preventing the occurrence of dissatisfaction among employees for example. The proposed resource aggregation strategies are tested on three simulated data instances deducted from a part of the actual rail transit network in the region of Amsterdam. The distribution of the resources among the tasks has been manipulated to some extent in order to generate instances with a varying degree of resource correlation.

The computational results show that some of the developed aggregation strategies are able to have a positive impact on the performance of the algorithm. A reduction of the computation time up to approximately 36% is achieved for both the PCA and the ad-hoc aggregation strategy when applied to the medium correlated instance using a parameter setting with an optimality tolerance $\varepsilon \in \{0.1, 0.5\}$ and relatively small values of the upper bound UB_{ar}. Moreover, the ad-hoc aggregation strategy shows similar results for different values of the parameter n. Simply considering a single original resource appears to be a slightly less effective strategy, but still can achieve a reduction up to approximately 28%. In general, we have noticed that the aggregation strategies do not have a positive impact for the data instances with minimum and maximum correlation of the resources. Furthermore, including a second principal component in case of the PCA aggregation strategy also does not have a positive impact. For all strategies, it holds that we should determine the parameter setting for which the best result are obtained. Preferably, we should set the parameters such that the occurrence of type I and type II errors are minimized.

We are able to obtain equally fair solutions with a relatively small optimality tolerance according to the used fairness criterion. We prefer values of the optimality tolerance on the interval $[0.1, 2]$ in order to minimize the fairness error. This interval partially coincides with the interesting tolerance region with respect to the reduction of the overall computation time. The use of an aggregation strategy with the correct parameter settings in a column generation context may result in a more efficient approach to incorporate the preferences of employees and increase their happiness and welfare. The fact that we obtained the best results for the data instance which is a close representation of a real world situation using only a single (aggregated) resource instead of four, is promising for applying the aggregation strategies to problems with comparable properties such as vehicle routing.

7.1 Future Research

Whether we are able to take advantage of the positive effects of the aggregation algorithm, depends on multiple aspects such as the total share of generated shifts by the algorithm and the percentage of type I and II errors. There appears to be some relation between these aspects and the moment for which we observe a positive impact of the aggregation strategy. It would be very interesting to investigate whether it is possible to derive an explicit relation such that we are able to make a good first guess for the parameter setting. It would be interesting to test this approach on other data instances.

The characteristics of our problem made it difficult to use certain pruning strategies. In particular, because of the fact that we had to consider restrictions on the associated resource fractions of a shift. Therefore, further research on additional, preferable more efficient, pruning strategies and further development of the labeling algorithm is recommended in order to increase the impact of the aggregation strategies.

In this thesis, we investigated the possibility of obtaining fair solutions more efficiently by making use of the resource correlations. The fairness of a solution is defined by an equal distribution of the resources among the depot stations. In the proposed pricing framework, we tried to achieve this by assessing the fairness of each shift. Alternatively, further research can be done on the possibilities of assessing the fairness differently such as measuring the overall fairness of the shifts assigned to a depot station for each iteration of the column generation method.

Lastly, the SPPRC can be solved by numerous exact and non-exact solution methods. In this thesis, a dynamic programming approach is chosen mainly because of the fact that the idea is clearly interpretable and therefore, it was valued as a suitable method for the insertion of resource aggregation extensions. However, additional research can be done on alternative solution methods or adaptations of the current dynamic programming approach such as a bidirectional labeling algorithm.

Acknowledgements

This thesis was written in fulfillment of the master program Econometrics and Management Science, with a specialization in Operations Research and Quantitative Logistics at the Rotterdam Erasmus University.

I would like to acknowledge several people for their contribution to this thesis. First of all, I would especially like to thank my supervisor dr. Twan Dollevoet for his great advice and expertise during the past few months. He always provided my with detailed feedback which I appreciate a lot. Furthermore, I would like to thank the second reader Thomas Breugem MSc. for his time and effort looking into this thesis.

Finally, I want to thank my family, girlfriend and friends who were there for me during the past few months.

References

Abbink, E., Fischetti, M., Kroon, L., Timmer, G., and Vromans, M. (2005). Reinventing crew scheduling at netherlands railways. *Interfaces*, 35(5):393–401.

Abbink, E. J., Albino, L., Dollevoet, T., Huisman, D., Roussado, J., and Saldanha, R. L. (2011). Solving large scale crew scheduling problems in practice. *Public Transport*, 3(2):149–164.

Adams, J. S. (1963). Towards an understanding of inequity. *The Journal of Abnormal and Social Psychology*, 67(5):422.

Alexander, S. and Ruderman, M. (1987). The role of procedural and distributive justice in organizational behavior. *Social justice research*, 1(2):177–198.

Balas, E. (1982). A class of location, distribution and scheduling problems: Modeling and solution methods. In *Proceedings of the Chinese–U.S. Symposium on Systems Analysis*. eds. P. Gray and L. Yuanzhang (Wiley, 1983).

Balas, E. and Carrera, M. C. (1996). A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research*, 44(6):875–890.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329.

Beasley, J. E. (1987). An algorithm for set covering problem. *European Journal of Operational Research*, 31(1):85–93.

Beasley, J. E. (1990). A lagrangian heuristic for set-covering problems. *Naval Research Logistics (NRL)*, 37(1):151–164.

Beasley, J. E. and Chu, P. C. (1996). A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392–404.

Beasley, J. E. and Jörnsten, K. (1992). Enhancing an algorithm for set covering problems. *European Journal of Operational Research*, 58(2):293–300.

Borndörfer, R., Reuther, M., Schlechte, T., Schulz, C., Swarat, E., and Weider, S. (2015). Duty rostering in public transport-facing preferences, fairness, and fatigue. In *CASPT*.

Breugem, T., Dollevoet, T., and Huisman, D. (2017). Is equality always desirable?: Analyzing the trade-off between fairness and attractiveness in crew rostering. *Econometric Institute Research Papers*, (EI2017-30).

Caprara, A., Fischetti, M., and Toth, P. (1999). A heuristic method for the set covering problem. *Operations Research*, 47(5):730–743.

Caprara, A., Toth, P., and Fischetti, M. (2000). Algorithms for the set covering problem. *Annals of Operations Research*, 98(1-4):353–371.

Ceria, S., Nobili, P., and Sassano, A. (1998). A Lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81(2):215–228.

Ceselli, A., Righini, G., and Salani, M. (2009). A column generation algorithm for a rich vehicle-routing problem. *Transportation Science*, 43(1):56–69.

Chvatal, V. (1979). A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235.

Colquitt, J. A., Conlon, D. E., Wesson, M. J., Porter, C. O., and Ng, K. Y. (2001). Justice at the millennium: a meta-analytic review of 25 years of organizational justice research. *Journal of applied psychology*, 86(3):425.

Dantzig, G. B. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8(1):101–111.

Desaulniers, G., Desrosiers, J., and Solomon, M. M. (2002). Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In *Essays and surveys in metaheuristics*, pages 309–324. Springer.

Desaulniers, G., Desrosiers, J., and Solomon, M. M. (2006). *Column generation*, volume 5. Springer Science & Business Media.

Desrochers, M., Desrosiers, J., and Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354.

Desrochers, M. and Soumis, F. (1988). A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR: Information Systems and Operational Research*, 26(3):191–212.

Desrochers, M. and Soumis, F. (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13.

Desrosiers, J., Soumis, F., and Desrochers, M. (1984). Routing with time windows by column generation. *Networks*, 14(4):545–565.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, 42(5):977–978.

Fisher, M. L. and Kedia, P. (1990). Optimal solution of set covering/partitioning problems using dual heuristics. *Management Science*, 36(6):674–688.

Ford Jr, L. R. and Fulkerson, D. R. (1958). A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101.

Forsyth, D. R. (2018). *Group dynamics*. Cengage Learning.

Freling, R., Lentink, R. M., and Wagelmans, A. P. (2004). A decision support system for crew planning in passenger transportation using a flexible branch-and-price algorithm. *Annals of Operations Research*, 127(1-4):203–222.

Goodman, P. S. and Friedman, A. (1971). An examination of adams' theory of inequity. *Administrative Science Quarterly*, pages 271–288.

Greenberg, J. (1987). A taxonomy of organizational justice theories. *Academy of Management review*, 12(1):9–22.

Grötschel, M., Borndörfer, R., and Löbel, A. (2003). Duty scheduling in public transit. In *Mathematics—Key Technology for the Future*, pages 653–674. Springer.

Hartog, A., Huisman, D., Abbink, E. J., and Kroon, L. G. (2009). Decision support for crew rostering at ns. *Public Transport*, 1(2):121–133.

Huisman, D. (2007). A column generation approach for the rail crew re-scheduling problem. *European Journal of Operational Research*, 180(1):163–173.

Jacobs, L. W. and Brusco, M. J. (1995). Note: A local-search heuristic for large set-covering problems. *Naval Research Logistics (NRL)*, 42(7):1129–1140.

Jütte, S. and Thonemann, U. W. (2015). A graph partitioning strategy for solving large-scale crew scheduling problems. *OR Spectrum*, 37(1):137–170.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer.

Lavoie, S., Minoux, M., and Odier, E. (1988). A new approach for crew pairing problems by column generation with an application to air transportation. *European Journal of Operational Research*, 35(1):45–58.

Lawler, E. E. (1977). Reward systems. *Improving life at work*, 163:226.

Löbel, A. (1998). Vehicle scheduling in public transit and lagrangean pricing. *Management Science*, 44(12-part-1):1637–1649.

Lorena, L. A. N. and Lopes, F. B. (1994). A surrogate heuristic for set covering problems. *European Journal of Operational Research*, 79(1):138–150.

Maenhout, B. and Vanhoucke, M. (2010). A hybrid scatter search heuristic for personalized crew rostering in the airline industry. *European Journal of Operational Research*, 206(1):155–167.

Mehlhorn, K. and Ziegelmann, M. (2000). Resource constrained shortest paths. In *European Symposium on Algorithms*, pages 326–337. Springer.

Mitchell, J. E. (2011). Branch and cut. *Wiley Encyclopedia of Operations Research and Management Science. New York: Wiley*.

Nishi, T., Sugiyama, T., and Inuiguchi, M. (2014). Two-level decomposition algorithm for crew rostering problems with fair working condition. *European Journal of Operational Research*, 237(2):465–473.

Potthoff, D., Huisman, D., and Desaulniers, G. (2010). Column generation with dynamic duty selection for railway crew rescheduling. *Transportation Science*, 44(4):493–505.

Pugliese, L. D. P. and Guerriero, F. (2013). A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200.

Righini, G. and Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51(3):155–170.

Toth, P. and Vigo, D. (2002). *The vehicle routing problem*. SIAM.

Weick, K. E. and Nesset, B. (1968). Preferences among forms of equity. *Organizational Behavior and Human Performance*, 3(4):400–416.

Zhu, X. and Wilhelm, W. E. (2012). A three-stage approach for the resource-constrained shortest path as a sub-problem in column generation. *Computers & Operations Research*, 39(2):164–178.

Ziegelmann, M. (2001). Constrained shortest paths and related problems.

Appendices

A. Solution Methods of the SPPRC

Several methodologies have been developed in order to solve the SPPRC such as heuristics, Lagrangian relaxation and dynamic programming. The SPPRC can be very time consuming to solve even with sophisticated algorithms. In the column generation approach, the subproblem does not necessarily need to be solved to optimality every single iteration. In general it is sufficient to only solve it to optimality in order to show that there does not exist a path with negative reduced costs. That is, in the last pricing step of the column generation algorithm. This implies that it is sufficient to find near-optimal solutions of the SPPRC in precedent iterations i.e., heuristics can be applied to find feasible paths with negative reduced costs. It is possible to distinguish different types of heuristics by looking at the different areas of application such as: pre-processing, direct search and dynamic programming.

When considering methods to solve the SPPRC, we can distinguish between exact methods and non-exact methods:

- Exact

  1. **Constraint programming**
     Whereas the unconstrained shortest path problem can be efficiently solved in polynomial time, adding a single constraint makes the problem $\mathcal{NP}$-complete. Therefore constraint programming is not considered to be efficient as it is time costly and as explained before, it is not necessary to solve the subproblem to optimality in every iteration.
  2. **Lagrangian relaxation**
     Lagrangian relaxation is often applied to the resource constraints. LR is a well-known method to obtain upper and lower bounds for the problem. Afterwards, branch-and-bound based algorithms are applied to close the gap between the bounds. Also the bounds can be used in the pre-processing phase in which arcs and nodes are removed because they are only used in solutions which are considered to be infeasible (based on the bounds).
  3. **Dynamic programming**
     Dynamic programming is considered to be the standard approach for solving the SPPRC with pseudo polynomial complexity. A well known dynamic programming approach is the labeling algorithm. This method basically uses a certain data structure to label the nodes in a graph. A node v can have multiple labels. Each label corresponds to a certain path from node s to node v which stores the costs and resource usage corresponding to the $s - v$ path. The algorithm uses two sets with

labeled nodes and unlabeled nodes respectively. There exists different rules for selecting the next node to label such as Breadth-First (FIFO), Depth-First (LIFO) and Best-First-Search. Dijkstra's algorithm (Dijkstra, 1959) is a good example of a label-setting algorithm in which a Best-First-Search strategy is applied. Important to note here is that the Dijkstra's algorithm is not applicable for graphs with both positive and negative costs on the arcs.

    In (Pugliese and Guerriero, 2013) several methods for solving some extension of the SPP are evaluated. In case of the SPPRC in combination with an acyclic graph, the authors point out a solution approach proposed by Zhu and Wilhelm (Zhu and Wilhelm, 2012). They propose a 3-stage approach to solve the SPPRC to optimality in a column generation context. In the first stage they pre-process the original graph by reducing and removing nodes and arcs which are part of infeasible paths. The second stage converts the resulting graph into an enlarged graph by considering several copies of each node and in the third stage they apply a label-setting approach for solving the problem.

4. **Path Ranking / k-shortest paths**
   This method basically enumerates k solutions and ranks them in non-decreasing cost order. The first path which satisfies the resource constraints then is considered to be the optimal solution (with respect to the costs). Simply enumerating a number of solutions is usually not very efficient and furthermore it is known that the number of paths to be ranked before finding a feasible path may be exponential in size of the graph (Ziegelmann, 2001).

- Non-exact

  1. **Heuristics**
     Heuristics are based on the following three topics:
     - Pre-processing: that is simplifying the actual instance such that it is easier to solve. For example solving a series of restricted networks which only have a limited number arcs.
     - Dynamic programming heuristics: techniques which basically try to speed up the dynamic programming algorithms as described before. This can be done by implementing rules such as only allowing a max number of labels or stronger dominance rules etc.
     - Local search: start from a feasible path and add, switch or delete a node or arc in order to find an improving feasible path.

B. Benchmark Computation Time

Besides the fact that we observe volatility in the first few iterations, we also observe some deviations in general. Especially for smaller values of ϵ it might be that the algorithm is less stable. This is in accordance with our observation of erratic surfaces for smaller values of ϵ in the surface plots.

B.0.1 Benchmark computation time for the minimum correlated dataset

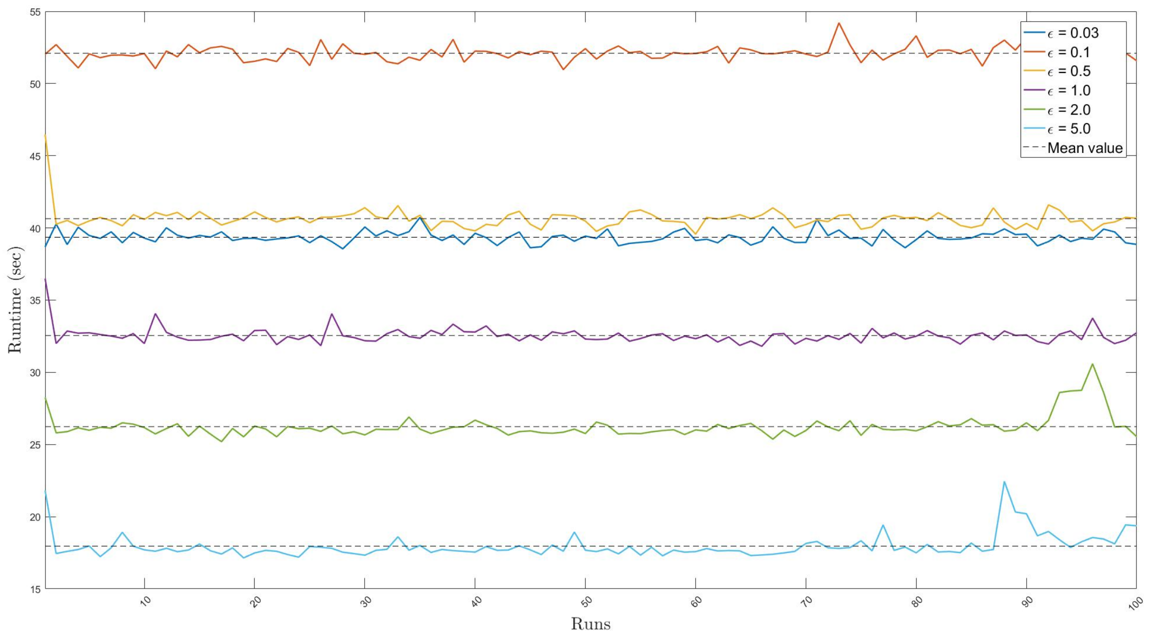

Figure B.1: It shows 100 observations for the benchmark computation time for multiple values of ϵ.

Table B.1: The statistics correspond to the benchmark runs performed with the dataset with minimum correlation. The statistics in the white columns corresponds to all 100 observations and the statistics in the gray colored columns correspond to the last 80 observations.

	$\overline{b}$	$\overline{b}$	s_b	s_b	$min\{b\}$	$min\{b\}$	$max\{b\}$	$max\{b\}$	size n	size n
$\epsilon = 0.03$	39.35	39.34	0.42	0.42	38.55	38.55	40.73	40.73	100	80
$\epsilon = 0.1$	52.10	52.13	0.53	0.54	50.96	50.96	54.21	54.21	100	80
$\epsilon = 0.5$	40.64	40.57	0.72	0.44	39.57	39.57	46.48	41.6	100	80
$\epsilon = 1.0$	32.55	32.50	0.57	0.39	31.80	31.80	36.49	34.06	100	80
$\epsilon = 2.0$	26.24	26.27	0.77	0.81	25.19	25.37	30.59	30.59	100	80
$\epsilon = 5.0$	17.96	17.97	0.83	0.79	17.15	17.20	22.44	22.44	100	80

B.0.2 Benchmark computation time for the medium correlated dataset

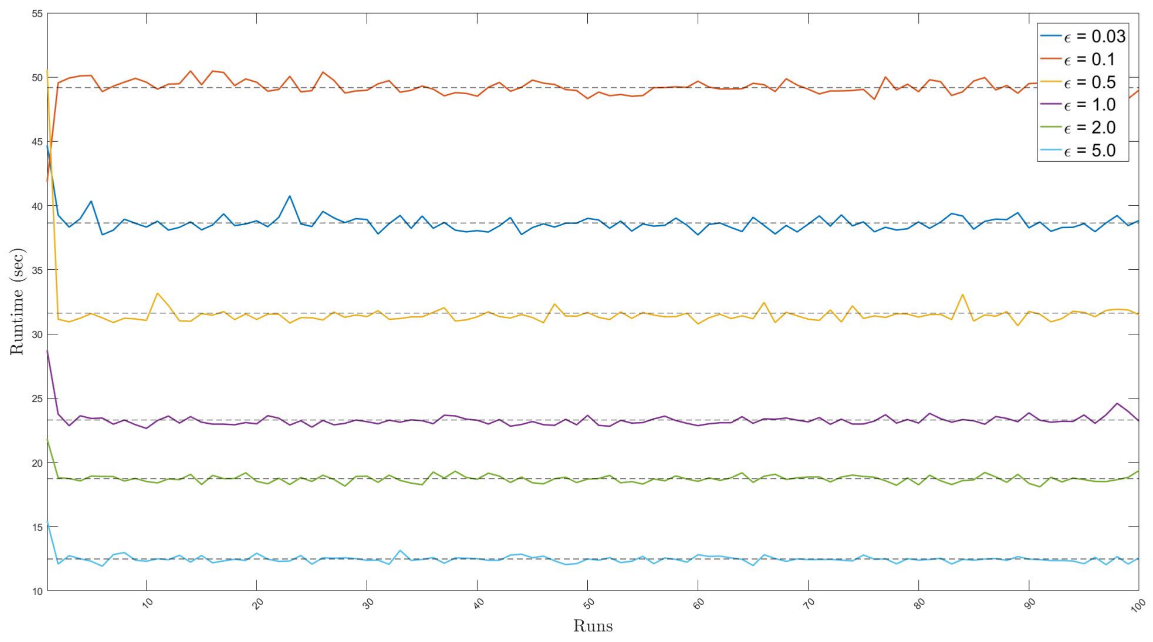

Figure B.2: It shows the 100 of runs for all values of ϵ. Clearly in the first run some variation in the computation time is observed.

Table B.2: The statistics correspond to the benchmark runs performed with the dataset with medium correlation. The statistics in the white columns corresponds to all 100 observations and the statistics in the gray colored columns correspond to the last 80 observations.

	$\bar{b}$	$\bar{b}$	s_b	s_b	$min\{b\}$	$min\{b\}$	$max\{b\}$	$max\{b\}$	size n	size n
$\epsilon = 0.03$	38.64	38.57	0.80	0.50	37.71	37.71	44.69	40.76	100	80
$\epsilon = 0.1$	49.18	49.15	0.89	0.44	41.84	48.27	50.48	50.39	100	80
$\epsilon = 0.5$	31.62	31.44	1.96	0.38	30.64	30.64	50.59	33.09	100	80
$\epsilon = 1.0$	23.30	23.26	0.63	0.31	22.65	22.75	28.73	24.6	100	80
$\epsilon = 2.0$	18.74	18.70	0.41	0.28	18.10	18.1	21.81	19.37	100	80
$\epsilon = 5.0$	12.48	12.44	0.38	0.22	11.92	11.97	15.45	13.15	100	80

B.0.3 Benchmark computation time for the maximum correlated dataset

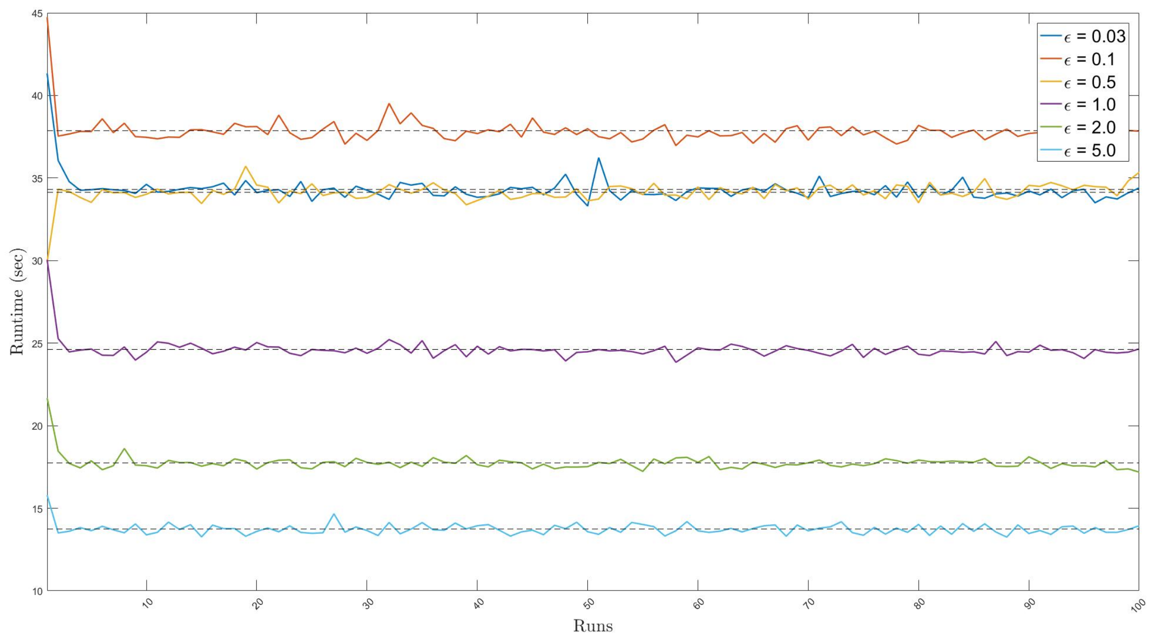

Figure B.3: It shows 100 observations for the benchmark computation time for different values of ϵ.

	$\bar{b}$	$\bar{b}$	s_b	s_b	$min\{b\}$	$min\{b\}$	$max\{b\}$	$max\{b\}$	size n	size n
$\epsilon = 0.03$	34.31	34.19	0.83	0.43	33.31	33.31	41.33	36.23	100	80
$\epsilon = 0.1$	37.87	37.79	0.82	0.46	36.97	36.97	44.74	39.52	100	80
$\epsilon = 0.5$	34.14	34.18	0.58	0.38	29.91	33.38	35.72	35.34	100	80
$\epsilon = 1.0$	24.61	24.54	0.61	0.25	23.84	23.84	30.05	25.22	100	80
$\epsilon = 2.0$	17.75	17.70	0.47	0.22	17.19	17.19	21.68	18.19	100	80
$\epsilon = 5.0$	13.74	13.73	0.33	0.27	13.25	13.25	15.79	14.67	100	80

Table B.3: The statistics correspond to the benchmark runs performed with the dataset with maximum correlation. The statistics in the white columns corresponds to all 100 observations and the statistics in the gray colored columns correspond to the last 80 observations.

C. Results Experiments *P2min* and *P2max*

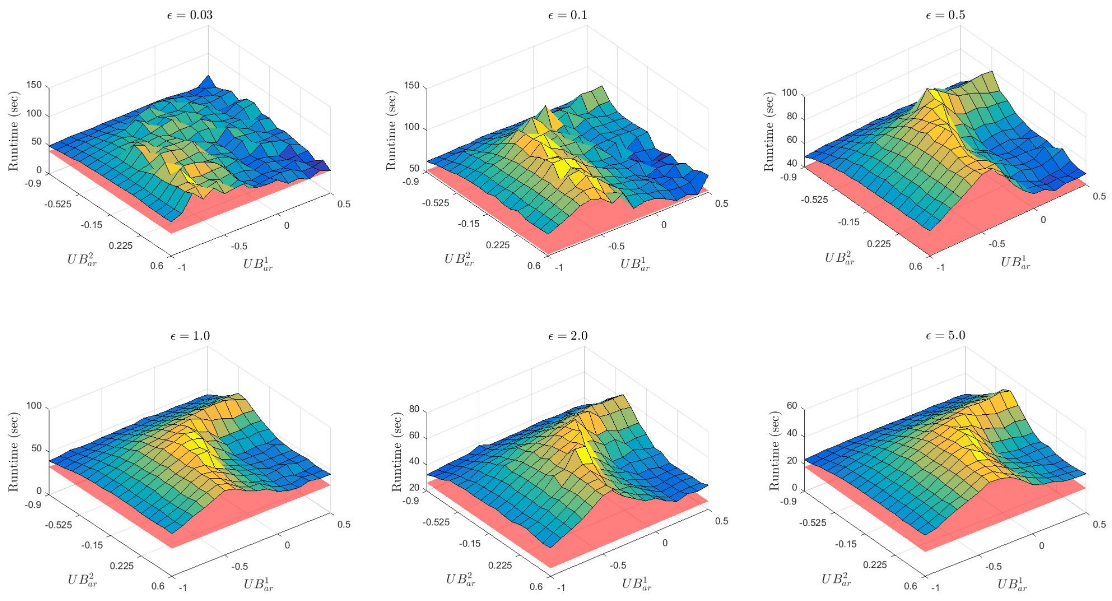

Figure C.1: The 6 subplots display the result of experiment *P2min*. Each subplot shows the performance of the algorithm for different values of the upper bounds UB^1_{ar} and UB^2_{ar}.

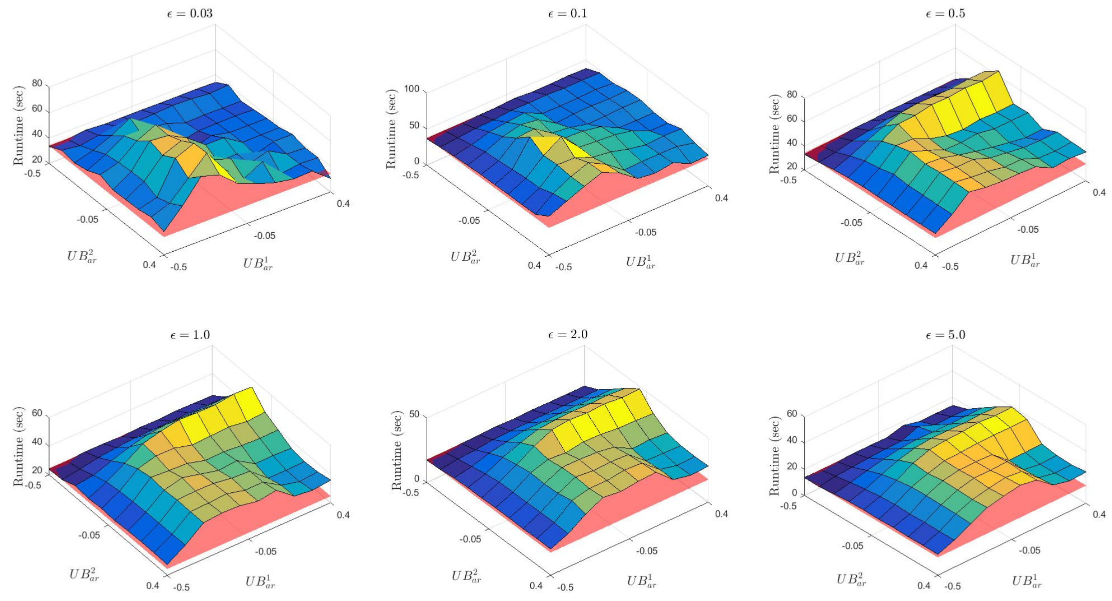

Figure C.2: The 6 subplots display the result of experiment *P2max*. Each subplot shows the performance of the algorithm for different values of the upper bounds UB^1_{ar} and UB^2_{ar}.

D. Additional Figures for the Algorithm Performance

This appendix contains a few additional figures for the aggregation algorithm performance evaluated in Section 6.3.

Figures D.1, D.2 and D.3 show the cumulative proportion of the errors for different values of UB_{ar}. Note, that in order to generate these figures we excluded the column generation iterations in which only a small number of shifts are generated. The cumulative proportions of the errors corresponding to these iterations are considered to be inaccurate since they are based on only a few shifts and typically a large volatility in the proportions is observed. We use those iterations in which the number of shifts generated, is larger than 10% of the maximum number of shifts generated in a single iteration for that particular instance.

In Figures D.5 and Figure D.6, the distribution of the error proportions is evaluated for $\varepsilon = 0.03$ and $\varepsilon = 0.1$ respectively. The error distribution is not evaluated for $\varepsilon \geq 1$ since the aggregation algorithm does not have a positive impact on the computation time for these values of ε.

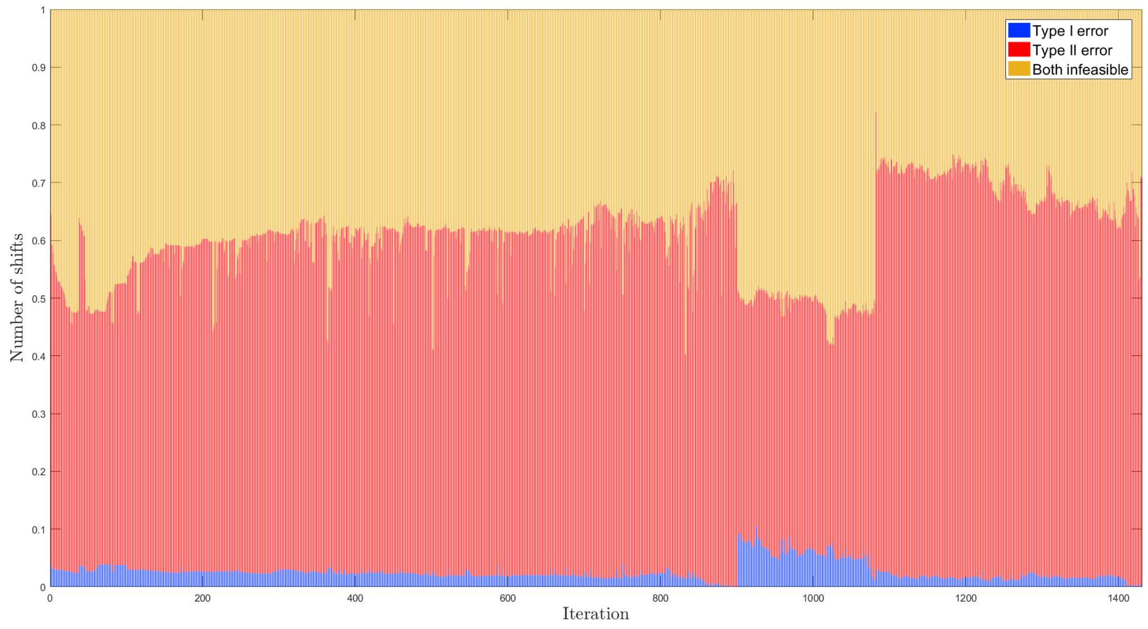

Figure D.1: It shows the cumulative proportion of the errors for $UB_{ar} = -0.1$.

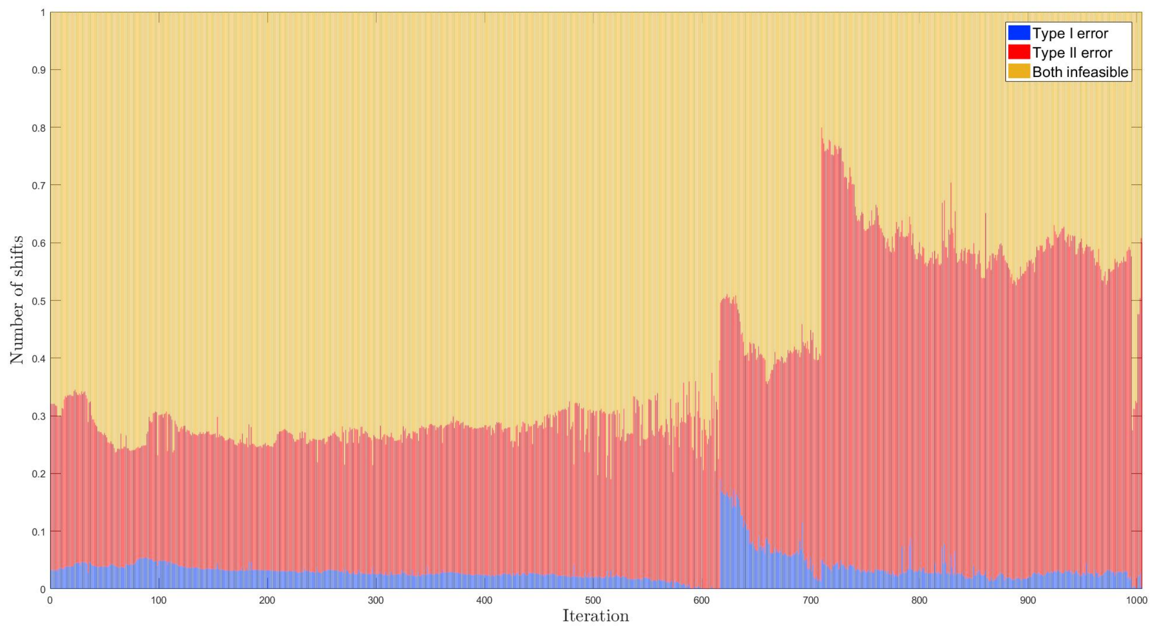

Figure D.2: It shows the cumulative proportion of the errors for $UB_{ar} = -0.3$.

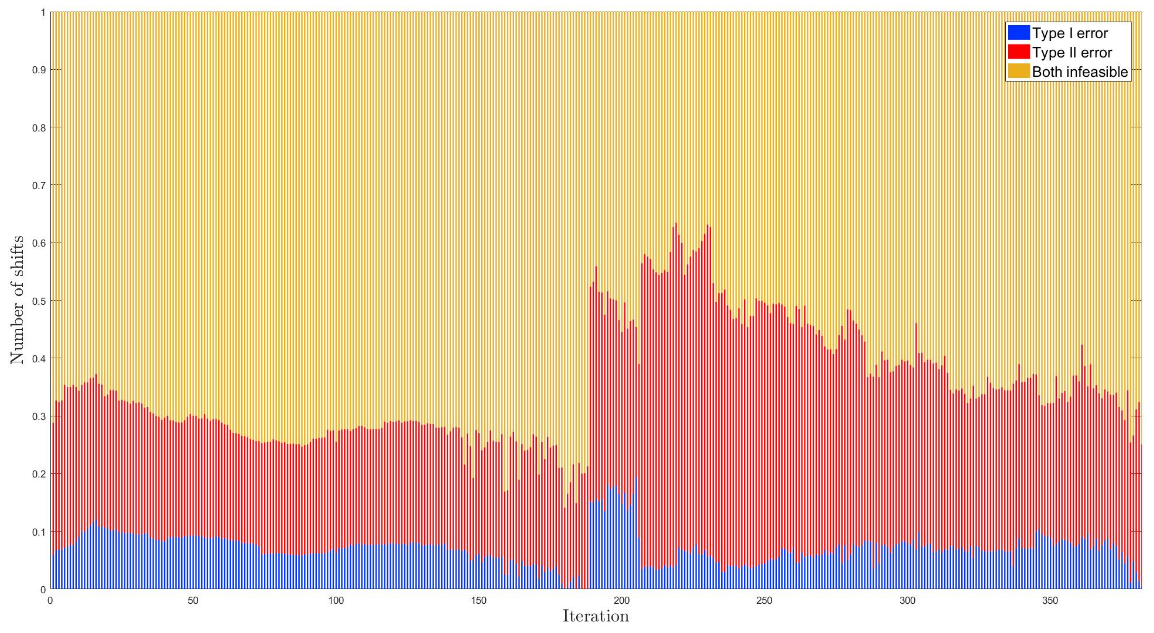

Figure D.3: It shows the cumulative proportion of the errors for $UB_{ar} = -0.6$.

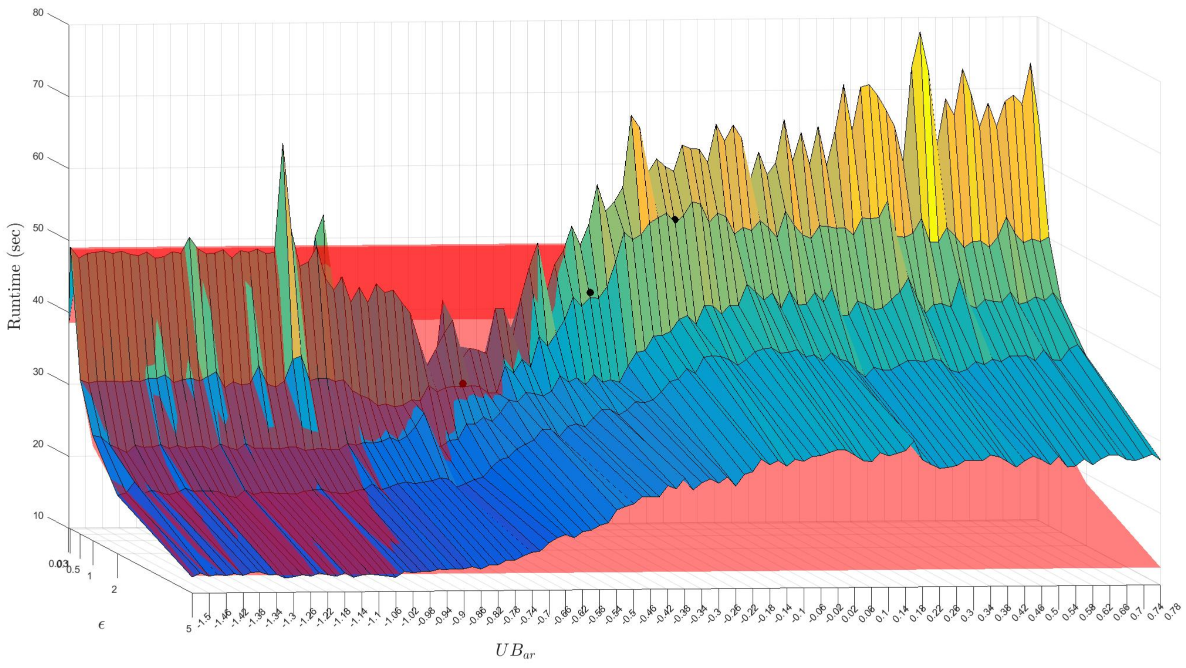

Figure D.4: The three black dots correspond to the three stacked bar plots (see Figures 6.11, 6.12 and 6.13). The black dot at the right-hand side correspond to Figure 6.11 and Figure 6.13 correspond to the black dot at the left-hand side.

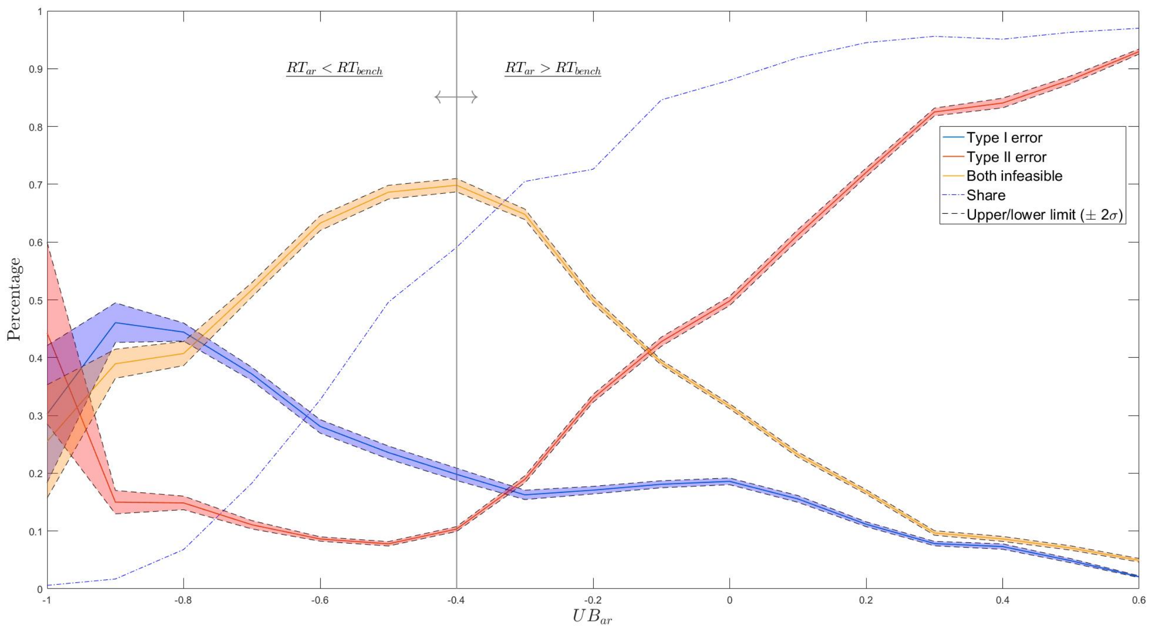

Figure D.5: It shows the average percentage of the error types including the corresponding confidence intervals. The left (right) hand side of the vertical black line shows for which values of UB_{ar} the computation time of the aggregation algorithm RT_{ar} is smaller (larger) than the computation time of the benchmark algorithm RT_{bench}. The blue dashed line indicates the share (%) of the total number of shifts generated by the aggregation algorithm. The figure is generated for an optimality tolerance $\varepsilon = 0.03$.

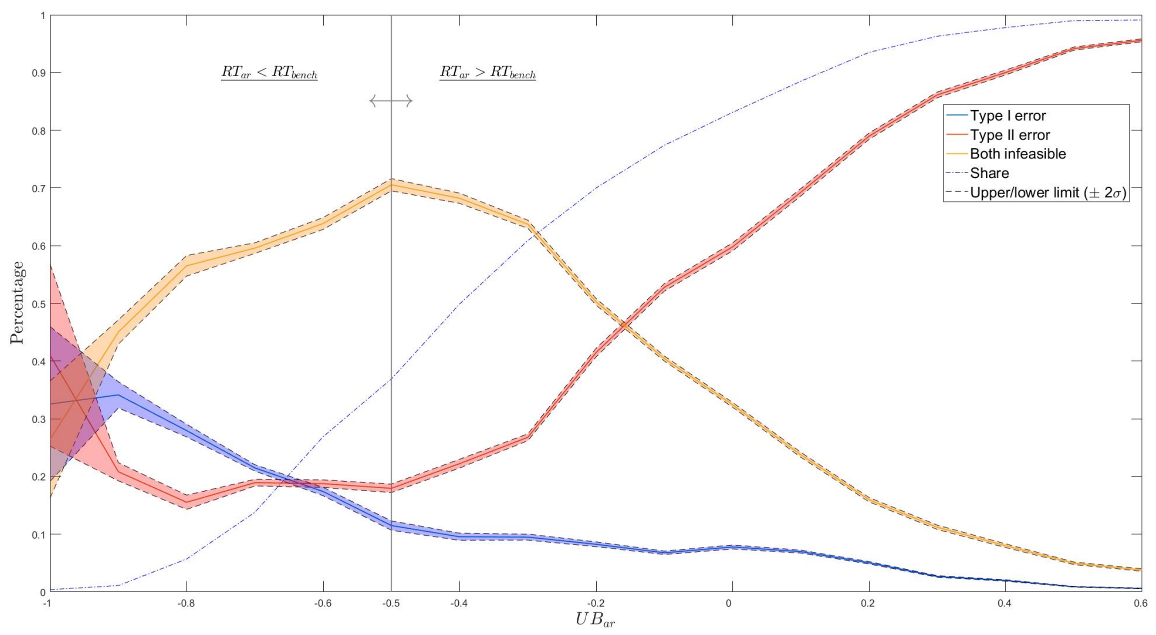

Figure D.6: It shows the average percentage of the error types including the corresponding confidence intervals. The left (right) hand side of the vertical black line shows for which values of UB_{ar} the computation time of the aggregation algorithm RT_{ar} is smaller (larger) than the computation time of the benchmark algorithm RT_{bench}. The blue dashed line indicates the share (%) of the total number of shifts generated by the aggregation algorithm. The figure is generated for an optimality tolerance $\varepsilon = 0.1$.