Department of Mechanical Engineering
Dynamics and Control Research Group
Manufacturing Systems Engineering Master Track

Development of a Vehicle Scheduling Tool for Large-Scale Electric Bus Transit Networks

Eindhoven University of Technology

Master Thesis

In partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering

J.W.M. Wijnheijmer

TU/e Supervisors:

dr. ir. A.A.J. Lefeber
prof. dr. H. Nijmeijer

VDL-ETS Supervisor:

ir. S.J.A. Rutten

Dynamics and control number: DC 2020.017

Eindhoven, February 2020

Summary

To mitigate the problems caused by global warming, it is important to transition to alternative fuel vehicles. Electric buses are a solution to reduce the emissions of public transport. Besides the environmental and legislative forces to transfer to electric vehicles, lower costs of ownership will be a benefit in the future. However, bus operators are hesitant to implement these alternative fueled vehicles because of fears of running out of energy. It is therefore important for VDL to advise their customers about the scheduling of electric vehicles. However, due to charging, difficulties in scheduling arise. The goal of this thesis is to develop a tool that schedules electric vehicles.

Previously, tools have been developed by VDL to aid manual scheduling. Furthermore, a scheduling heuristic was developed, which did not support multiple depots and the solution needed manual alterations. Besides this heuristic, a tool is developed in the work of Monhemius [1] that solves the scheduling problem to optimality. However, the method used is computationally expensive and not practical for any but the smallest problems.

In this research, a heuristic is implemented based on the work of Adler [2]. This is called the concurrent scheduler and has been first developed by Bodin, Rosenfield, and Kydes [3]. In the work of Adler [2], the possibility of charging is added to this method. For the concurrent scheduler, the trips in the timetables are sorted on starting time. The method assesses the service trips consecutively, where the cheapest bus is assigned to the current service trips. This is why the method is called the concurrent scheduler.

The concurrent scheduler is implemented in MATLAB and is compared to the VDL scheduler and, for small instances, the optimal solution. For the test timetables, the concurrent scheduler gives a 12% lower costs than the VDL scheduler. Furthermore, the optimality gap for the timetables tested is small. Furthermore, the largest timetable is scheduled within two minutes. However, the solution of the concurrent scheduler is not perfect. For example, charging during rush hours is not discouraged and charging earlier than necessary is not considered.

To counteract these issues, multiple alterations to the concurrent scheduler are proposed and implemented. In the tests conducted, no benefit of these alterations became apparent and thus, it is advised to use the concurrent scheduler without these alterations. Besides the proposed alterations to improve the concurrent scheduler, an addition to the concurrent scheduler is made to support a limited number of chargers for each charging location. Besides expanding the usability with this addition, the solution of the concurrent scheduler can be used as an input for another scheduler.

Since the concurrent scheduler does not support non-linear charging or non-fixed charging times, another solution method is proposed in this research. This solution method is based on the column generation algorithm. First, using a simplified problem, a reformulation is performed to assess if this reformulation can reduce the number of necessary decision variables. It is shown that with this reformulation, the number of decision variables initially increases. However, with the implementation of the column generation algorithm, the number of necessary decision

variables is lower than the original formulation. Therefore, the reformulation in combination with the column generation algorithm is a promising technique to solve the scheduling problem. The idea behind column generation is to split up the problem into two parts. The first part is to solve the problem for the current set of columns, called the restricted master problem. The second part, to generate a new column, is called the subproblem.

Next, the simplified model is expanded to support charging, where a limit is set on the number of available chargers and to the charging power of a charging location. However, only linear charging and one depot, one charging location and one start/end location is considered. The result of this expanded model is compared with the result of the concurrent scheduler for a simplified situation, where it became clear that the column generation gives a lower costs on average. However, the computation time is longer. The most important drawback of the proposed implementation of the column generation algorithm is that the largest timetables are not solvable within reasonable time. This is caused by the definition of the columns, resulting in a hard to solve subproblem for larger timetables.

To obtain an integer solution, an integer solver is used. Since an integer solver is limited with respect to the number of decision variables, a rounding algorithm based on a linear solver is proposed to solve larger problems. For the test timetables, the rounding algorithm gives results with higher costs than when an integer solver is used. It is therefore not advised to use this rounding algorithm.

Next, it is investigated if using the solution of the concurrent scheduler heuristic as an initial solution for the column generation algorithm improves the computation time or solution quality. For simplicity, the situation without charging is considered. For the assessed test timetables, both the quality and the computation time are improved. It is therefore advised to use the solution of a heuristic as the initial solution for the column generation model in combination with an integer solver to obtain an integer solution.

Finally, the recommendations are given. The main recommendation is to continue with the development of the scheduling tool using the column generation technique with the usage of a heuristic as initial solution. It is recommended to alter the formulation of the subproblem to solve larger problems. Concluding, both the concurrent scheduler as implemented in this research and the column generation algorithm are useful techniques for electric vehicle scheduling. The concurrent scheduler gives feasible solutions quickly and can be used as a scheduling tool itself or as an initial solution for a more advanced solver. The model based on column generation as implemented in this research gives good solutions. It is expected that with reformulations the computation times can be improved. Therefore, the column generation algorithm is a promising future research direction.

Preface

The transition to renewable energy and new propulsion technologies for transport is challenging. For the benefit of the environment it is important that this transition will be completed shortly. The lack of understanding and some unwillingness of people to change their behavior is a limit to this. I am glad that with this research a step in the right direction has been made, and I hope that the framework provided here will be used to improve the quality of life of many people.

The ending of this project means that a substantial step in my personal life has been completed. My studying period has been eventful, with numerous enjoyable experiences and unfortunately some hard lessons in life. All combined I am looking back with fond memories and appreciative of all the opportunities I was given.

I would like to thank all the employees at VDL-ETS for welcoming me in the team during my project. Furthermore, I would like to thank my supervisors, for their thorough guidance throughout this project. Last but not least I would like to thank my family, for supporting me and putting things into perspective during this tough period we are going through. Without them taking over certain obligations I would not have been able to perform this research with as much commitment as I did. Thank you all!

Jan

Contents

List of Figures

List of Tables

Nomenclature

Sets

Set	Explanation	Indices
T	Set of service trips	t,τ
S	Set of charging locations	s
D	Set of depot locations	d
L	Set of locations	l
V	Set of vehicle tasks	v
V'	Subset of vehicle tasks	v
B	Set of buses	b
Z	Set of time blocks	z,ζ

Matrices

Matrix	Explanation
X_{tv}	If trip $t \in T$ is performed by vehicle task $v \in V$
S_{zv}	If a charger is used during time block $z \in Z$ in vehicle task $v \in V$
E_{zv}	How much energy is charged during time block $z \in Z$ in vehicle task $v \in V$
comp	Compatibility array
comps	Charging compatibility array

Decision variables

Group	Variable	Explanation
Primal	a_{tb}	If trip $t \in T$ is performed by bus $b \in B$
	u_v	If vehicle tasks v is used in solution
Dual	π_τ	Shadow price of trip $\tau \in T$
	θ_ζ	Shadow price of usage of charger during time block $\zeta \in Z$
	ρ_ζ	Shadow price of energy during time block $\zeta \in Z$
Subproblem	$\delta_{\tau,j}$	If trip τ is performed in new vehicle task j
	σ_ζ	If charging occurs during time block $\zeta \in Z$
	ϵ_ζ	Amount of energy charged during time block $\zeta \in Z$

Parameters

Group	Symbol	Explanation
Costs	c	
	c^b	Cost of a bus [€/day]
	c^s	Cost of a charger [€]
	c^e	Cost of energy [€/kWh]
	c^w	Cost of driving [€/km]
	c^v_v	Cost of vehicle task v [€]
Energy	e	
	e^w	Energy usage of bus [kWh/km]
	e^t_t	Energy usage of trip t [kWh]
	$e^{b,\max}$	Maximum energy level of bus [kWh]
	$e^{b,\min}$	Minimum energy level of bus [kWh]
Charging	ϵ	
	$\epsilon^{s',\max}$	Maximum amount of energy delivered in time block on charging location [kWh]
	$\epsilon^{s,\max}$	Maximum amount of energy delivered by a charger during a time block [kWh]
	$\epsilon^{b,\max}$	Maximum amount of energy added in a time block to a bus [kWh]
	$\epsilon^{b,\min}$	Minimum amount of energy added in a time block to a bus [kWh]
Time	h	
	h^{gap}	Minimum time between trips or actions [min]
	h^{start}_t	Start time of trip t [min]
	h^{end}_t	End time of trip t [min]
	$h^{deadhead}_{l_1,l_2}$	Deadhead time between location l_1 and l_2 [min]
	h^s	Charging time [min]
	$h^{\text{start},z}_t$	Time block in which trip t starts [-]
	$h^{\text{end},z}_t$	Time block in which trip t ends [-]
Location	l	
	l^{start}_t	Start location of trip t [-]
	l^{end}_t	End location of trip t [-]
Number	n	
	n^b	Number of buses used [-]
	n^s	Number of charging locations [-]
	n^z	Number of time steps [-]
	n^t	Number of trips [-]
	n^l	Number of unique locations [-]
	n^v	Number of vehicle tasks [-]
	n^d	Number of depots [-]
Power	p	
	$p^{s,\max}$	Maximum power delivered to a charging location [kW]
	$p^{b,\max}$	Maximum charging power of a bus [kW]
	$p^{b,\min}$	Minimum charging power of a bus [kW]
Distance	w	
	w_{l_1,l_2}	Distance between location l_1 and l_2 [km]
	w^t_t	Distance of service trip t [km]

Chapter 1

Introduction

1.1 Background

Global climate change is an issue which requires a transition towards alternative fuel vehicles. Traditionally, diesel powered buses are used in public transport. Besides the carbon emissions of diesel buses, also air quality in urban areas is a problem [4]. The transition to electric buses can solve these problems [5]. However, electric buses have limiting properties that have to be taken into account. Firstly, electric buses have a lower range than diesel buses and are unable to drive an entire day without recharging. Secondly, a fleet of electric buses consumes a high amount of energy. The buses are charged at a finite number of locations, causing a high power demand at specific locations. The charging location can have a limit on the available power.

Because of the problems that diesel powered buses cause, legislators are requiring the use of electric vehicles for public transport. In the Netherlands, the government has agreed with the operators that only emission free vehicles are used from 2025 on [6]. Besides the legislative force, there are also economical incentives for operators to use electric buses. Electric buses are expected to have a lower total cost of ownership than diesel powered buses in the near future.

However, operators currently have little experience using electric buses. Concerns that arise are the possibility that buses get stranded with empty batteries or that not enough chargers are available. To solve these issues, operators order more buses than required. This, of course, increases the costs considerably.

VDL is a manufacturer of city buses and is a substantial player in the bus and coach business in Europe. VDL produced the first large-scale electric bus fleet. The trend of the market is to transfer to electric city buses. In addition to providing an operator with buses, VDL can supply the peripherals as well, an example of this is the charging infrastructure. Because customers are hesitant to transfer to an electric fleet, it is wise for VDL to provide consultation and explain that electric buses are able to satisfy their requirements. With this extra service and proof that a lower total cost of ownership is possible the likelihood that VDL can sell buses increases. It is therefore useful for VDL to obtain an electric vehicle scheduling tool.

1.2 Problem definition

In this section, the problem definition is given. The problem for operators is to decide for each service trip which bus will perform that service trip. The service trips combined form the timetable, an example of a timetable can be found in Table 1.1. The timetable is composed in collaboration with the local government and therefore cannot be changed. This means that the departure and arrival locations, including the start and end times, together with the driving distances are fixed.

In the case of diesel buses the problem is a vehicle scheduling problem (VSP), which is solvable in polynomial time [7]. Electric buses cannot drive an entire day without recharging, and thus, the problem becomes a special case of the VSP, an electric vehicle scheduling and charging problem (EVSCP). The complexity of this problem is known in literature as an $\mathcal{NP}$-hard problem [8]. This means that the number of decision variables increases non-polynomially.

The problem to be solved is to allocate electric buses to all the trips without exceeding real world limitations. Some of these restrictions are: range limitations, limited number of charging locations, deadhead trips to and from the depots, and total available charging power of a charging location. The goal for VDL is to, for a given timetable, give the customer a possible allocation of buses to trips and buses to chargers such that the total costs for the customer are minimized. A possible schedule without charging is shown in Figure 1.1.

Table 1.1: Example of timetable

From	Start	End	To	Dist [m]
ehvbst	08:00	08:30	ehvapt	10000
ehvbst	08:15	08:45	ehvapt	10000
ehvapt	08:40	09:10	ehvbst	10000
ehvapt	08:55	09:25	ehvbst	10000
ehvbst	09:00	09:30	ehvapt	10000

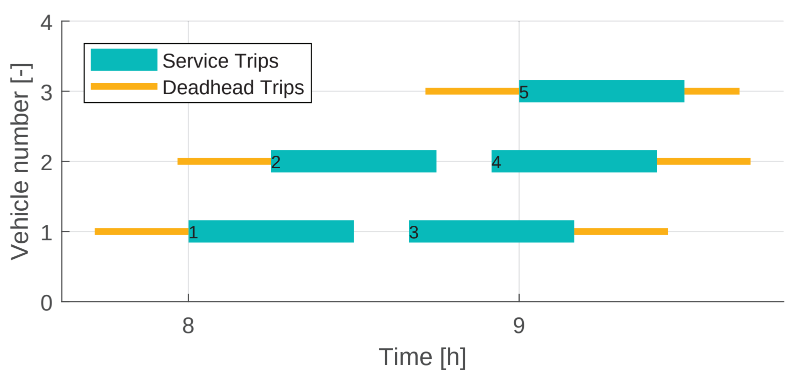


Figure 1.1: A possible schedule, without charging

Model inputs and outputs

Input

For the vehicles, the following parameters need to be known: price, depreciation, energy usage, maximum charging power, and battery capacity. Furthermore, the timetable, where buses have to be assigned to, with departure and arrival times in addition to the trip distance has to be known. For the chargers, the price and maximum charging speeds are needed. In addition to this, the depot location(s) and the proposed charging locations have to be known, together with the distance and travel time between these locations. Furthermore, for each charging location, the maximum available power that can be taken from the grid should be known. In addition to this, the cost of electricity has to be known.

Output

The output of the proposed model is the following: A feasible schedule for which the total daily costs are minimized. For each vehicle, charging actions and driven trips are determined.

1.3 Previously used scheduling methods

VDL scheduler

In recent years, VDL has developed a fleet management tool, which is used for bus scheduling. In this section, this scheduler is briefly explained. This scheduler is based on the idea that it is efficient to have few deadhead trips and thus, it is efficient to start the next service trip on the end location of the previous service trip. A deadhead trip is a trip between two location without passengers. To implement this, the first service trip is assigned to the first vehicle. After that, sets of trips are made, which are made from time compatible service trips. The size of these sets of trips is dependent on the battery capacity. Service trips are added until the battery would be empty. The deadhead trips toward the first service trip and after the last service trip of the block are not taken into account. Each subsequent service trip has to start on the end location of the previous service trip. The energy feasibility of these blocks is checked, now with the deadhead trips, and if it is energy infeasible, the last service trip in the block is removed. Charging can occur between blocks, non-linear charging is allowed and the charging time is not fixed. An attempt is made to reduce the number of vehicles charging during peak demand. If no previous service trip starts or ends at a location a new bus is taken. Furthermore, if no service trip departs from a location any more, the bus that ends in that location stays there. In the end, all the buses return to their home depot. This model does support charging on the depot and on defined start and end locations. Charging locations other than the depot or on service locations is not supported.

Linear programming scheduler

Another solution approach is to use a MILP to solve the scheduling problem to optimality. This approach is used in the work of Monhemius [1] for VDL. Here, the optimal solution is obtained for a simplified version of the scheduling problem. Even though the solution is optimal, solving any timetable with more than a few service trips is computationally expensive.

1.4 Defining the scope and solution approach

The goal is to develop a scheduling tool, that is able to schedule electric buses while taking into account their limitations. The total costs should be minimized for the operator. Furthermore, the time to compute the schedule should be reasonable. The scheduling of drivers to buses is not taken into account. A few of the most important assumptions are: the fleet is homogeneous, there is a single charger type, buses start each day fully charged, and charging is linear.

It is stated earlier that the problem is $\mathcal{NP}$-hard, and therefore, hard to solve to optimality for any but the smallest problems. To be able to provide a feasible schedule quickly a heuristic is implemented first. This heuristic is known as a concurrent scheduler and supports multiple depots, multiple charging locations and deadhead trips. One inherent drawback of heuristics is that it is not guaranteed that the solution is globally optimal, and that the optimality gap is unknown. Hereto, the optimality gap is assessed for small problem instances. Another limitation of the concurrent scheduler is that the charging time is fixed. The schedule provided by the heuristic can be used directly, or it can be used as an initial solution for another solution method.

Next, a column generation algorithm is applied to a simplified problem where energy usage and charging is neglected. To be able to apply this algorithm the problem is re-formulated first by applying Dantzig-Wolfe decomposition. After this, the problem becomes a set partitioning problem whereon column generation can be applied. Next, the simplified problem is expanded to support electric vehicles and charging. Finally, the results of the concurrent scheduler and the column generation model are compared and promising research directions are discussed.

1.5 Structure of thesis

This thesis is structured in the following way: First, the current developments in vehicle scheduling are described, including multiple solution methods. After that, the application of a heuristic is explained. Then, multiple test timetables are defined and a heuristic is used to make schedules for these timetables. The results are compared with the results of the scheduler previously developed by VDL and, for small problem instances, to the optimal solution. In addition to this, multiple methods to improve the heuristic are proposed and explored. The results of these possible improvements are also discussed.

To overcome the limitations of the heuristic, a model based on column generation is formulated that can implement features that are not implemented in the heuristic. The theory behind this model is explained using a simplified problem. After that, the simplified model is expanded to support electric vehicles and charging. Then, this model is tested on the same test timetables and the quality of the solution is assessed. Furthermore, the benefit of using the solution of the heuristic as a starting position for this solution method is explored. Finally, the conclusions are drawn and promising directions for future research are discussed.

Chapter 2

Literature review

The problem is to assign buses to a pre-determined timetable. This problem is known in literature as a Vehicle Scheduling Problem (VSP) and has been investigated extensively. There are additions to this problem, for example: with time windows, limited range, multi-depot and other additions. Ibarra-Rojas [9] made a summary and overview of multiple studies and their solution methods. Bunte and Kliewer [10] have made an overview of Vehicle Scheduling Problems and give information about the general workings of these models. A version of the VSP with vehicles that have limited range is investigated by Adler in [11] and Li in [12]. Sometimes, schedules are still made by hand. The scheduling method developed by Guedes [13] resulted in a schedule that has a 31% reduction in costs compared to the hand-made version of that problem set.

The VSP is similar to the Traveling Salesman Problem (TSP). In the research of Lenstra [7] it becomes clear that the single depot case can be solved in polynomial time $\mathcal{O}(n^3)$, and that the multi-depot case is $\mathcal{NP}$-hard. The multi-depot case can only be solved to optimality for small problem instances. In practice, almost every schedule has either multi-depots, a heterogeneous fleet or charging sessions, and thus is $\mathcal{NP}$-hard. To solve these large and/or $\mathcal{NP}$-hard problems, a multitude of (meta-)heuristics exists to solve these problems close to optimality in a reasonable time. However, a heuristic does not guarantee the globally optimal solution. An overview of heuristics that can be applied for these types of problems is made by Pepin [14]. A more detailed description of multiple heuristics has been made by Morton [15] and Gendreau [16]. The VSP also has similarities to the graph coloring problem, which is also known as an $\mathcal{NP}$-hard problem, as can been seen in the work of Pinedo [17, p. 166].

To obtain a feasible solution quickly, a concurrent scheduling algorithm has been developed by Bodin, Rosenfield, and Kydes [3]. In the work of Adler [2], this algorithm is expanded to support vehicles with limited range and with recharging possibilities. This method is used in several studies, [2, 11, 12, 18], and has an optimality gap of between 10% and 15% but is rapid. It is mostly used as an initial solution for other methods.

The Ant Colony Algorithm is another heuristic, which has been used by Wei [19] and Wang [20]. It is based on the Greedy heuristic. Colonies are distributed over the solution space. This approach is based on the application of multiple search directions where, if a direction is promising, there is a higher likelihood of the other colonies to search near that location.

An alternative approach is the Large Neighborhood Search (LNS). This approach takes a feasible solution, re-optimizes a subset of the solution and if the new total solution is better than the old solution, the new solution is saved as the best solution. When this is iterated it is probable that the solution converges to a good solution. A drawback of this approach is that it is possible to end in a local minimum. In the work of Xu [21], LNS is used as the first step in their combined heuristic. After that, a branch-and-price algorithm is applied. According to Xu [21], this solution is computationally efficient. A version of neighborhood search, called tabu search, is applied in the work of Adler [2]. Here, the last few visited solution locations are stored and the algorithm is prohibited to reach these locations again. This way, the algorithm is less likely to end up in a local minimum [14]. An electric vehicle routing problem with time windows (E-VRPTW) is solved by Schneider [22], where a combination of variable neighborhood search (VNS) and tabu search is used. The author states that the combination of VNS and tabu search is most effective for instances with large time windows.

To shorten the computation time for larger problem instances, a time-space network formulation can be used. This method is applied in the work of Kliewer [23]. Because of the formulation of the problem, the deadhead trips of each depot can be combined into fewer arcs. This state-space reduction is most beneficial for instances with a high number of trips and a low number of depots. The state-space reduction lowers the amount of decision variables and thus, makes the problem easier to solve [23]. To solve the E-VSP, Reuer [24] also used a time-space network. However, in that research, the charging time is chosen either zero or a fixed time. Lu [25] solves the problem of electric taxi fleet scheduling of reserved taxi trips. Here, also a time-space formulation is used, but a minimum charge time is imposed, making it possible to assume that the vehicle is completely recharged after the charging session. No research has been found where an E-VSP is solved using a time-space formulation where the SoC of the vehicles is accurately represented. In the work of Guedes [13] the time-space formulation is used to solve a multi-depot multi-vehicle type scheduling problem. Then, using column generation and state-space reduction the problem is solved.

In the research of van Kooten Niekerk [26], multiple solution techniques have been implemented. One of these is the column generation algorithm to obtain a good solution within a reasonable time. Column generation is a technique which enables to compute a good, but not necessarily optimal solution [27]. Alongside the column generation also Lagrangian relaxation has been used. This approach has also been used in the work of Löbel [28]. Here, one or multiple constraints are transferred to the objective function using penalties. In [26], also a label correcting algorithm is implemented in combination with the column generation. This algorithm is developed by Huang [29]. In the work of Adler [2], the column generation is combined with branch-and-bound, a combination that is known as branch-and-price. This approach is also used by Golden [30]. Li [12] used a truncated column generation to solve large problem instances.

It becomes clear that multiple solution methods are devised and used to solve the electric vehicle scheduling problem. A concurrent scheduling algorithm can provide a feasible solution quickly, which is the reason that this algorithm is used in this research. The column generation approach seems to be the most successful and therefore the most promising research direction when a higher quality solution is required. This is the reason that the column generation technique is used in this research.

Chapter 3

Concurrent scheduler

The first solution method is a heuristic, called the concurrent scheduler. With the use of a heuristic, the scheduling of electric buses to service trips as described in section 1.2, can be solved quickly. It is not guaranteed however, even unlikely, that the solution of a heuristic is the globally optimal solution. However, the solution can still be useful. For instance, it can be used to provide a solution if a schedule should be calculated quickly, or as an initial solution for a more advanced scheduling method. In this research, the concurrent scheduler heuristic is proposed as a possible improvement to the heuristic previously developed by VDL. The concurrent scheduler has been first developed by Bodin, Rosenfield, and Kydes [3]. In the work of Adler [2], the possibility of charging is added to this method.

The general idea behind the concurrent scheduler is that a good schedule can be obtained by working through the timetable a single time, where for each service trip, the cheapest available bus is selected to perform that service trip. Hereto, the service trips need to be sorted on start time.

In this chapter, the assumptions made and parameters used are explained first. After that, the theory and the application of the concurrent scheduler is discussed. To be able to compare the scheduling methods, some test timetables are constructed and presented. The final part of this chapter is the comparison of the schedulers, where for small problem instances, the optimality gap is discussed.

3.1 Assumptions

To schedule the vehicles, the parameters need to be known. The charging time, h^s, is fixed to 45 [min], because this ensures that the battery can be recharged fully regardless of the starting SoC. As a minimum time between trips, h^{gap}, 1 [min] is chosen. This is added to the concurrent scheduler to accommodate for passenger boarding/disembarking, or for starting/stopping the charging process. Furthermore, a battery capacity of 216 [kWh] is chosen, which is one of the capacities available in VDL buses. It is decided that 80% of the battery capacity is available, to reduce battery degradation. In addition to this, only a homogeneous fleet is considered, where all the vehicles have an energy usage, e^w, of 1.5 [kWh/km] regardless of weather, payload or other factors. Furthermore, it is assumed that every bus is pre-conditioned while still plugged in to the grid. This encompasses the heating/cooling of the cabin to the set-point, the heating/cooling of the battery pack and possibly, the pressurization of the pneumatic system. Moreover, it is assumed that every bus is charged fully overnight. The price of the bus is taken to be €500.000. With a 15 year use, no residual value and assumed 300 days of operation a year, the depreciation cost of the bus, c^b, is €111.11 per day. Furthermore, the cost of energy, c^e, is set at €0.20 per kWh and the variable costs, c^w for a bus are €0.10 per km. These variable costs are, as an example, to cover the maintenance of the vehicle. Note that these values are educated guesses. For the development of the scheduler, the accuracy of these parameters is not critical. The cost of a charging session during the day, c^s is set to €10. These are the costs for the use of the charger, the costs for the energy are calculated separately.

3.2 Concurrent scheduler algorithm

In this section, the algorithm is explained in detail. The code used and the documentation can be found in Appendix B.1.

3.2.1 Time and charging compatibility

It is not only important that subsequent trips can be performed by a vehicle in terms of energy limitations, but also in terms of time. In the work of Adler [2], this is solved with the usage of compatibility arrays. In this research the *comp* array is the time compatibility array and the *comps* is the charging compatibility array. Service trip i is called compatible with any other service trip j if the end time of trip i, $h^{\text{end}}(i)$ in addition to the time required to travel to the start location of trip j,$h^{\text{deadhead}}(l^{\text{end}}(i), l^{\text{start}}(j))$ and the minimum time between trips, h^{gap} is earlier than the start time of trip j, $h^{\text{start}}(j)$. If this expression is true, then the *comp* array is 1 on that location. The equation can be found below:

$$comp(i,j) = \begin{cases} 1, & \text{if } h^{\text{end}}(i) + h^{\text{deadhead}}(l^{\text{end}}(i), l^{\text{start}}(j)) + h^{\text{gap}} \leq h^{\text{start}}(j) \\ 0, & \text{otherwise} \end{cases} \quad \forall i, j \in T, \tag{3.1}$$

where T is the set of service trips. The *comp* matrix is used to check if it is possible for a vehicle to perform both service trips sequentially in terms of time, by driving directly to the start location of the next service trip. However, it is also possible that a vehicle charges between service trips. If two trips can be performed by the same vehicle in terms of time while visiting a charging location in between, these trips are called charging compatible.

$$comps(i,j,s) = \begin{cases} 1, & \text{if } h^{\text{end}}(i) + h^{\text{deadhead}}(l^{\text{end}}(i), s, l^{\text{start}}(j)) + 2h^{\text{gap}} \leq h^{\text{start}}(j) \\ 0, & \text{otherwise} \end{cases} \quad \forall i, j \in T; s \in S. \tag{3.2}$$

Equation (3.2) is used to determine if two trips are charging compatible. Note that the charging time is chosen as a constant, while ensuring that this charging time is sufficient to charge a vehicle fully, regardless of the SoC when charging starts. The charging time is added to the travel time of the incoming arcs of the charging stations $h(l^{\text{end}}(i), s)$, and thus, does not need to be added in the calculation of the *comps* array. Here, $s \in S$ is a charging location from the set of charging locations.

3.2.2 Charging Scheduling Algorithm

In this section, the Charging Scheduling Algorithm (CSA) is explained. This algorithm is used to determine which bus can perform the considered service trip with the lowest added costs, while taking time and energy constraints into consideration. Steps 1 through 4 are performed for each service trip consecutively, which is the reason why the service trips need to be sorted according to starting time. For the first service trip version (a) of step 1 is performed, where for the remaining service trip version (b) is used.

Step 1

The first step of the CSA is to generate all sequences that can lead up to the service trip. A sequence is a list of tasks that is performed consecutively. Only for the first service trip, step 1.a is performed. For subsequent service trips, step 1.b is completed.

Step 1.a As stated before, this step is only performed if the first service trip is considered. The goal of this step is to generate the sequences from the depots to the start location of the first service trip. The direct route from the depot to the start location is added. In addition to this, the routes from the depot via the charging locations to the start location are added. In Figure 3.1 these different possibilities are shown.

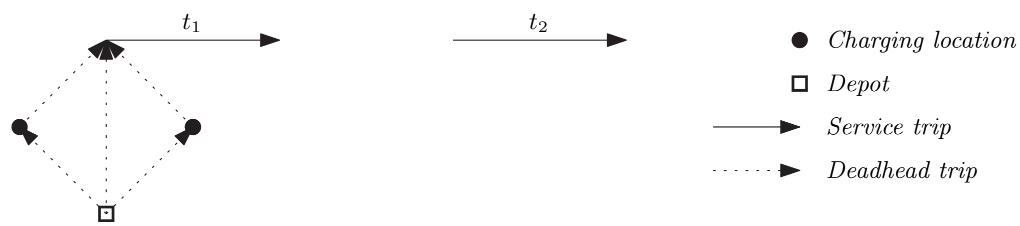


Figure 3.1: Step 1.a of CSA

Step 2

The goal of this step is to remove sequences that are not feasible in terms of energy. By removing these sequences at this point, the computation time is reduced. An arc is either a service trip or a deadhead trip. For each sequence, all the arcs are assessed and the energy level after each arc is calculated. If the arc is towards a charging location, the new energy level is set to the maximum capacity of the vehicle, while ensuring that this charging location can be reached with the energy remaining. In Figure 3.2, it is shown that the sequence that uses the left charging location is removed. Another action that can be performed at this step is the removal of dominated sequences. A sequence is dominated if there are other sequences present that are both faster and have lower costs. In the implementation of the concurrent scheduler this feature is disabled, since it increases the computation time. This means that the computation whether a sequence is dominated is more computationally expensive than to keep the sequence for the remaining steps of the CSA for the current service trip.

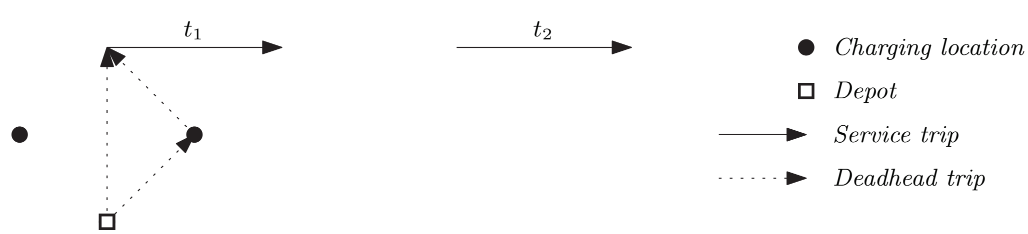


Figure 3.2: Step 2 of CSA

Step 3

This step copies the sequences of step 2 and for each charging location, adds the arc towards that charging location. Because of this step, the CSA adds the option of charging between service trips, as well as keeping the option to perform the next service trip directly after the previous service trip.

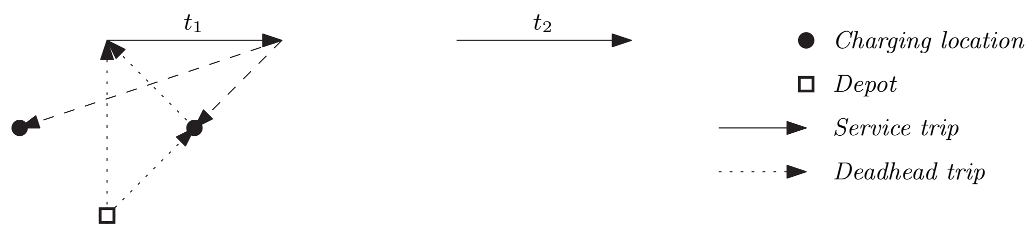


Figure 3.3: Step 3 of CSA

Step 4

This step is divided into several sub-steps. First, it is determined which sequences can still reach their home depot, as seen in Figure 3.4. For the remaining sequences, Figure 3.5, the total costs of the sequences are calculated. Then, the added costs from the previous service trip in the sequence, if that is the case, to the current service trip is calculated. The sequence with the lowest added costs is chosen and this sequence is added to the solution, Figure 3.6.

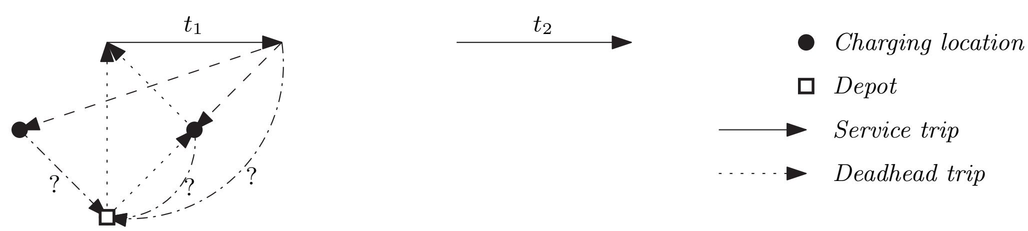


Figure 3.4: Step 4.1 of CSA

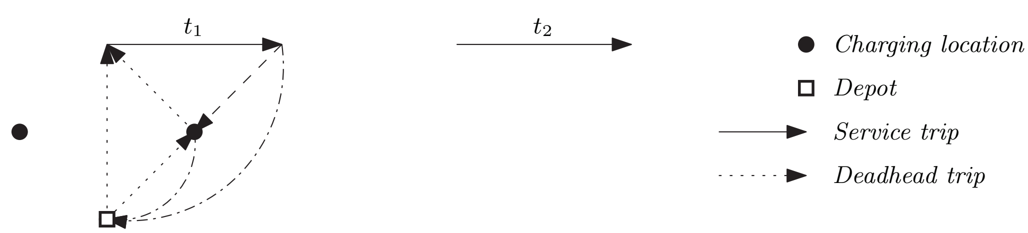


Figure 3.5: Step 4.2 of CSA

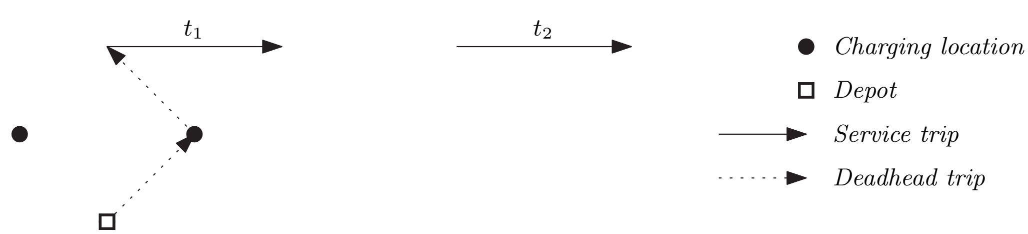


Figure 3.6: Step 4.3 of CSA

Step 1.b The goal of this step is to generate all the arc sequences from the end locations of the previously assigned buses toward the current service trip. For all the previously assigned buses, the last performed service trip is taken into account and the arcs toward the current service trips are added according to the *comp* and *comps* arrays. In addition to this, the sequences are generated to take a new bus from the depot. In Figure 3.7, the new sequences are visualized.

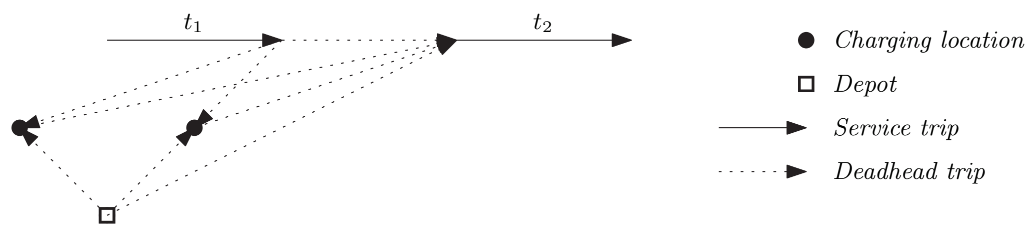


Figure 3.7: Step 1.b of CSA

3.2.3 Finishing concurrent scheduler

The steps of the CSA are repeated for all service trips. To finish the bus scheduling, the arcs toward the depots are added. Furthermore, the solution is checked on multiple factors: energy feasibility, if all service trips are performed exactly once, if the bus ends at the same depot as where it started, and if all subsequent arcs have the same start and end locations.

3.3 Test timetables

To compare the performance and quality of different solution methods, it is necessary to use the same timetables. To have a representative use case, test timetables are based on real timetables. A summary of the test timetables used in this research can be found in Table 3.1. There are three timetables with varying sizes whereof there are three versions of each timetable, with an increasing amount of charging locations. The number of charging locations for all test timetables is one, two and five.

The first three timetables are based on the schedule of Eindhoven between the train station and the airport, and consist of the first 14 trips in this schedule. To ensure that charging is necessary in this test timetable, the distances are multiplied with a factor four. In the case of a single charging location, the charger is located at the same location as the depot. For the second test timetable, the charger is located at the airport. In the third test timetable the extra chargers are located at unique locations, that where not in the timetable before.

The next three test timetables are also from Eindhoven between the train station and the airport, but span all the trips throughout the day. The charging locations are the same as in the first three schedules.

The test timetables 7 through 9 are all trips in Rotterdam, which has more start and end locations. Again, in the case of one charging location, this is located at the depot. In the case of two charging locations, the second charger is located at a location that is the start/end location for some trips. For the situation with five charging locations, the extra chargers are located randomly. A larger timetable than Rotterdam is not obtained. Note that for all test timetables, only one depot location is used, even though the concurrent scheduler supports multiple depot locations.

Table 3.1: Summary of test timetables

Test timetable number	# trips	# depots	# charging locations	# start/end locations	Lower bound # buses	Service trip distance [km]
1	14	1	1	2	3	560
2	14	1	2	2	3	560
3	14	1	5	2	3	560
4	203	1	1	2	7	1841
5	203	1	2	2	7	1841
6	203	1	5	2	7	1841
7	1096	1	1	19	43	9400
8	1096	1	2	19	43	9400
9	1096	1	5	19	43	9400
10	-	1	1	1	-	-

Table 3.2: Results of VDL scheduler on test schedules

Test schedule number	Computation time [s]	Cost per day	Number of buses used	Energy used [kWh]	Distance driven [km]	Number of charging sessions
1	2.57	€1148	7	1239	826	4
2	5.80	€870	5	993	662	5
4	7.98	€2762	14	3924	2616	16
5	7.91	€ 2202	11	3149	2099	14
7	13.90	€14730	81	18974	12649	67
8	18.17	€14682	83	17697	11789	74

The tenth and final timetable is the same as used in the research of Monhemius [1] and is also from Eindhoven, of all the service trips between the train station and the airport. Here, inbound and outbound service trips are combined such that the begin and end locations are at the train station. It can be chosen how many service trips are extracted from this timetable. The service trips in the timetable will be spread out through the day. This final timetable is used to compare the solution of the concurrent scheduler with the optimal solution.

3.4 Results and comparison of schedulers

The algorithm as described in the previous section is implemented in MATLAB. For more information regarding the implementation, see Appendix B.1 for the documentation. In this section, the results of both the VDL scheduler and concurrent scheduler on the test timetables are given[1] and compared.

3.4.1 Results of VDL scheduler

The test timetables are solved with the VDL scheduler, whereof the results can be found in Table 3.2. Note, that since the method of VDL does not support charging locations that are neither on the depot or start/end locations of service trips, test timetables 3, 6 and 9 are not scheduled.

[1]The computer used: Intel Core i7 4770HQ 2.2 Ghz, 16GB DDR3L 1600Mhz ram running on MacOS Mojave 10.14.5

As an example, the Gantt chart of the resulting schedule from test timetable four can be found in Figure 3.8. Here, one of the drawbacks of the VDL scheduler becomes clearly visible. Buses eight through fourteen are waiting to leave a location. In the timetable no service trip leaves that location and thus, these buses only perform one trip. This increases the costs considerably.

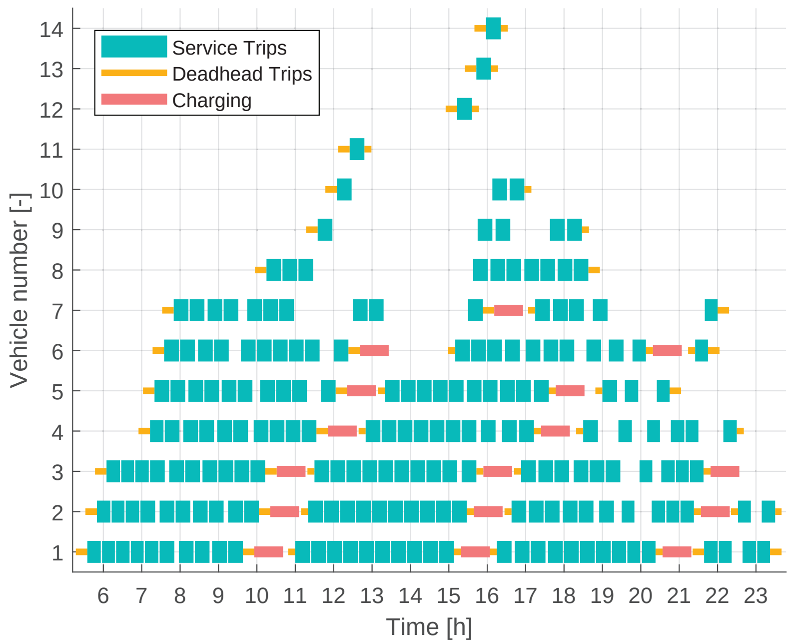


Figure 3.8: Schedule provided by the VDL scheduler for test timetable four

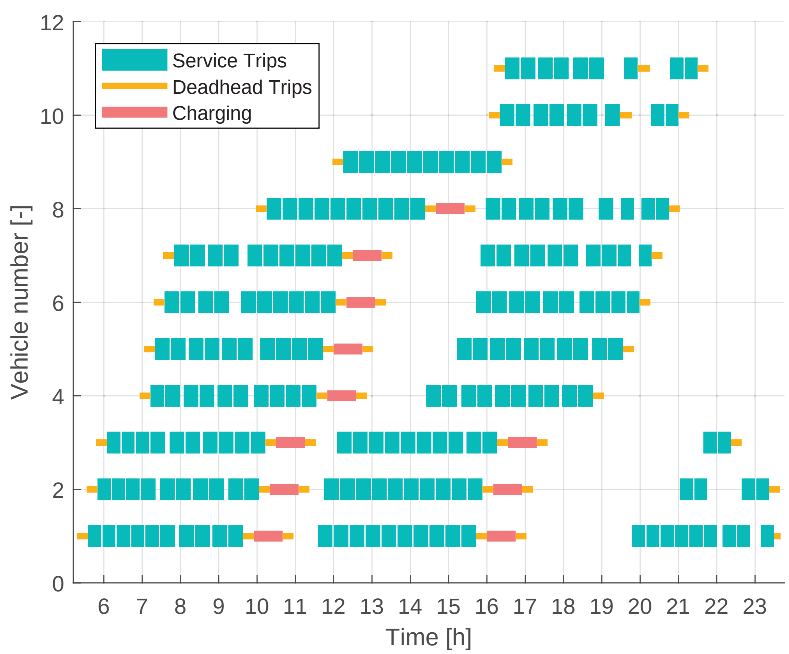


Figure 3.9: Schedule provided by the concurrent scheduler for test timetable four

In Figure 3.9 the schedule provided by the concurrent scheduler is shown. It can be seen that buses 10 and 11 drive deadhead trips to perform service trips instead of waiting.

Table 3.3: Results of concurrent scheduler on test schedules

Test schedule number	Computation time [s]	Cost per day	Number of buses used	Energy used [kWh]	Distance driven [km]	Number of charging sessions	Number of chargers used
1	1.30	€971	6	1066	710	2	2
2	1.25	€971	6	1066	710	2	2
3	1.31	€966	6	1048	699	2	2
4	3.17	€2250	11	3443	2296	11	4
5	3.63	€2057	10	3099	2066	12	3
6	6.22	€2110	10	3219	2146	14	3
7	42.08	€11665	60	16905	11270	49	14
8	56.69	€11256	62	16134	10756	62	12
9	99.35	€11136	55	16144	10763	72	15

3.4.2 Results of concurrent scheduler

The concurrent scheduler as described before has been tested on the test schedules from section 3.3. In Table 3.3, the results of the concurrent scheduler can be found. Furthermore, in Figure 3.9 the Gantt chart of a typical schedule constructed by the concurrent scheduler is given. Here, the time is on the x-axis and the vehicle number is on the y-axis. Each green block is a service trip, a red block is a charging session, and the yellow blocks are deadhead trips between locations. For the concurrent scheduler, it is important to note that it is usually cheaper to perform the next service trip directly, than to start charging, which is cheaper than taking a new bus. The concurrent scheduler continues to schedule service trips until the battery is almost empty, and then starts charging. Because in most timetables the starting times of service trips is similar, it results in similar times to start charging as well. A good example of this can be found in Figure 3.9. Here, it can be seen that during the time that buses 1-3 are charging for the second time, new buses are introduced. It could be beneficial to let some buses charge, before they are close to empty. This way, the charging demand would spread over the day, and thus, not many extra buses would be needed when other buses are charging. In addition to this, charging is not prohibited or discouraged during rush hours. Even though, at those hours the number of simultaneous service trips is the highest, and thus, charging should be limited.

From the idea that more charging locations increase the number of possible arcs, and thus lower costs are possible, one would assume that the total costs strictly decreases for more charging locations. As can be seen in Table 3.3, this is not the case. For test timetable 6, the total costs are higher than for test timetable 5, while the only difference is the number of charging locations. Figure 3.10 is used to explain this result.

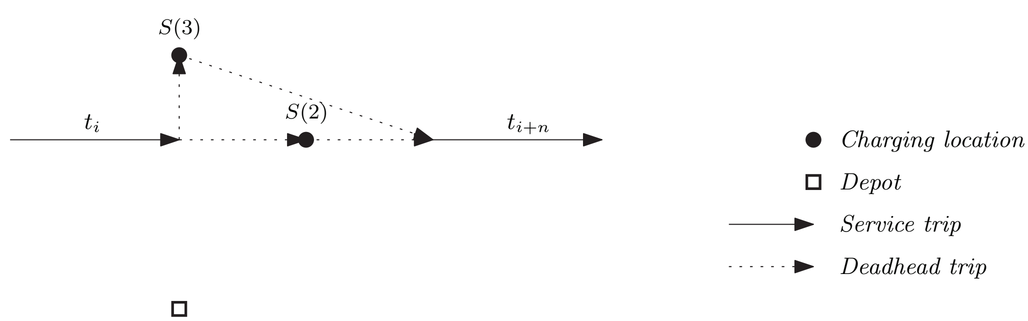


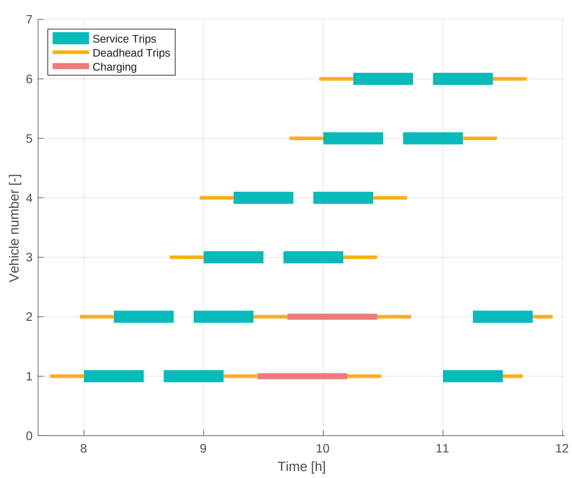
Figure 3.10: Explanation of not strictly lower costs in concurrent scheduler for more possible charging locations

For test timetable 5, charging location number 2 is present, but charging location 3 is not. For timetable 6 both are present. In the figure, the case is given where after service trip i, the bus does not have enough energy to return to the home depot. See step 4 of the CSA for more information. Because of this, the bus travels to the nearest reachable charging location. For schedule 5, this is location number 2, for schedule number 6 this is charging location number 3. As can be seen from the figure, the total travel distance is higher in the second case, this results in higher total costs.

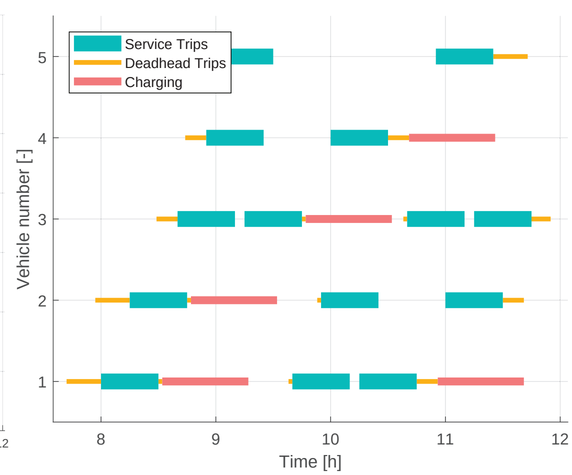
3.4.3 Comparison of concurrent scheduler and VDL scheduler

From tables 3.3 and 3.2 it becomes clear that, only for test timetable 2, the resulting schedule is cheaper with the VDL scheduler. Note that in this timetable, the service trip distance is set to a high value and it only consists of back and forth service trips with a charging location at one end. In Figure 3.11, the Gantt charts for both solution methods can be found. The resulting schedule of the concurrent scheduler is more expensive than the VDL scheduler because it uses one more bus. This bus is added because no other buses are available to perform the first service trip of bus 6. This drawback of the concurrent scheduler is explained section 3.4.2. The VDL scheduler works well for this instance because bus number 1 starts charging after a single service trip, which causes the charging sessions to be out of sync. This reduces the number of added buses because of simultaneous charging, reducing the total costs.

A charging session after just one service trip is chosen because of the following. The first service trip is added to the first bus. Next, the first service trip that is possible with regards to time from that location is added to the bus, in this case this is trip number 3. After this, it is checked if the bus can still reach the home depot, which in this case it can. Then, the next service trip is added, trip number 6. This combination is not possible with regards to energy, so trip number 6 is removed from the block. Then, it is checked if the location after service trip 3 has a charger, it has not. So trip number 3 is also removed from the schedule. This results in a schedule for bus 1 where it charges after service trip 1, as can be seen in Figure 3.11b.



(a) Concurrent scheduler



(b) VDL scheduler

Figure 3.11: Bus schedule for time table 2 for different solution methods

The solution methods are compared in Table 3.4 and the decrease in cost by using the concurrent scheduler is given. For the test timetables that can be used for both solution methods, the average cost reduction by using the concurrent scheduler as described previously is 12%

Table 3.4: Difference between VDL scheduler and Concurrent scheduler

Test schedule number	Costs VDL scheduler	Costs Concurrent Scheduler	Improvement of Concurrent Scheduler
1	€1148	€971	15.42%
2	€870	€971	-11.61%
4	€2762	€2250	18.54%
5	€2202	€2057	6.58%
7	€14730	€11665	20.81%
8	€14682	€11256	23.33%

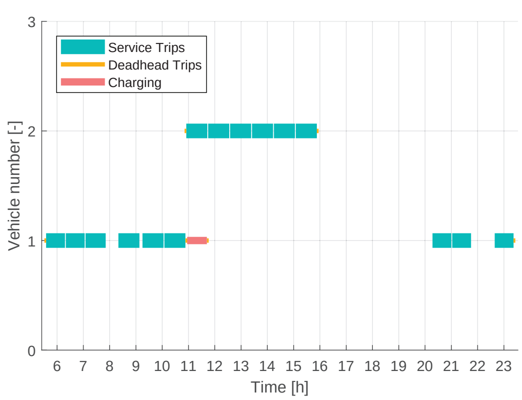
3.4.4 Optimality gap of concurrent scheduler

Up to this point, all the solutions are based on heuristics. It is therefore unknown how good these solutions actually are. In this section, the concurrent scheduler is compared with the optimal solution for small problem instances. The method of Monhemius [1] is used to determine the optimal solution.

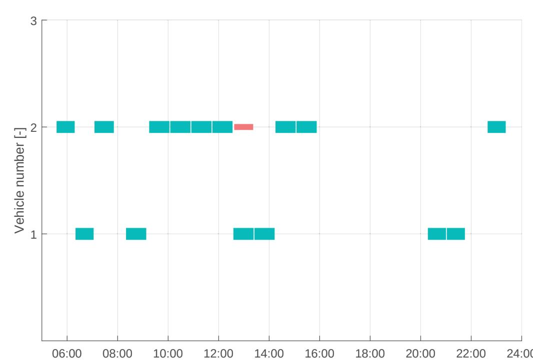
For both the concurrent scheduler and the optimal scheduler it is assumed that the energy usage of the bus is 1.5 [kWh/km], charging time is 45 [min], minimal time between trips is 1 [min], time to drive from or to the depot is 4 [min] and no energy is used for this deadhead trip. Furthermore, the battery capacity is set to 216 [kWh], of which 80% is available. The price of a bus is set to 111.11 [€/day], price of a charger is 20 [€/day] and charging costs 0.20 [€/min]. In this equation, h^{charging} is the number of minutes that the chargers are in use. The timetable that is used is timetable 10, whereof more information can be found in Section 3.3. The schedulers are tested from 5 to 18 service trips. The results can be found in Table 3.5. The cost per day is calculated by the following equation:

$$Cost = c^b \cdot n^b + 0.2 \cdot h^{\text{charging}} + c^s \cdot n^s. \tag{3.3}$$

From Table 3.5, it becomes clear that, the schedule of the concurrent scheduler gives the same daily costs as the optimal solution for the test cases with 5-7 and 9-18 service trips. This does not mean that the schedule is identical. An example of this can be found in Figure 3.12. Here, it becomes clear that the schedule is not the same, but they result in the same cost per day.



(a) Concurrent Scheduler



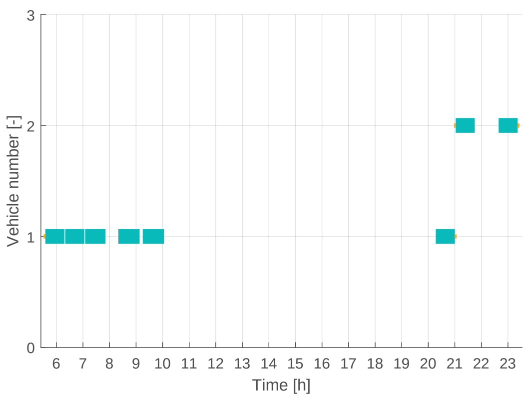
(b) Optimal schedule

Figure 3.12: Comparison between Concurrent scheduler and optimal schedule, for 15 service trips

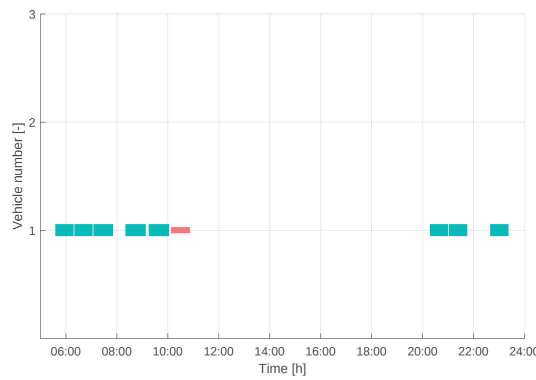
Table 3.5: Comparison between the concurrent scheduler and the optimal solution for increasing number of planned service trips

Number of service trips	Solution method	Computation time [s]	Total charging [min]	Number of buses	Number of chargers	Cost per day
5	Concurrent Scheduler	1,2	0	1	0	€111,11
	Optimum	3,8	0	1	0	€111,11
6	Concurrent Scheduler	1,3	0	1	0	€111,11
	Optimum	4,6	0	1	0	€111,11
7	Concurrent Scheduler	1,3	45	1	1	€140,11
	Optimum	4,6	45	1	1	€140,11
8	Concurrent Scheduler	1,3	0	2	0	€222,22
	Optimum	5,7	45	1	1	€140,11
9	Concurrent Scheduler	1,3	45	1	1	€140,11
	Optimum	4,7	45	1	1	€140,11
10	Concurrent Scheduler	1,4	0	2	0	€222,22
	Optimum	6,3	0	2	0	€222,22
11	Concurrent Scheduler	1,4	0	2	0	€222,22
	Optimum	8,9	0	2	0	€222,22
12	Concurrent Scheduler	1,3	0	2	0	€222,22
	Optimum	12,4	0	2	0	€222,22
13	Concurrent Scheduler	1,6	45	2	1	€251,22
	Optimum	26,5	45	2	1	€251,22
14	Concurrent Scheduler	1,6	45	2	1	€251,22
	Optimum	51,3	45	2	1	€251,22
15	Concurrent Scheduler	1,5	45	2	1	€251,22
	Optimum	345,4	45	2	1	€251,22
16	Concurrent Scheduler	1,5	45	2	1	€251,22
	Optimum	464,6	45	2	1	€251,22
17	Concurrent Scheduler	1,2	45	2	1	€251,22
	Optimum	172,2	45	2	1	€251,22
18	Concurrent Scheduler	1,6	45	2	1	€251,22
	Optimum	454,6	45	2	1	€251,22

Only for the schedule with eight service trips, the cost is different. In Figure 3.13, the schedules for this case can be found. Here, the drawback of the concurrent scheduler becomes visible. The scheduler keeps assigning trips to a bus until the bus is empty. In this case, this results in the use of an extra bus, since the first bus is empty, and there is not sufficient time to charge the bus and start the next service trip. In Figure 3.13b, the optimal solution is given. From this figure it becomes clear that it can be beneficial to start charging before the battery is empty. In this case, it results in using one bus less.



(a) Concurrent Scheduler



(b) Optimal schedule

Figure 3.13: Comparison between Concurrent scheduler and optimal schedule, for 8 service trips

Besides the difference in the value of the objective function of the schedulers, there is also a difference in computation time between the concurrent scheduler and MILP implementation. For the smaller instances, the optimal solution is around four times slower, where for the larger instances it is around 300 times slower. It is expected that this difference increases for higher amount of service trips.

3.5 Conclusion on concurrent scheduler

The concurrent scheduler is implemented and tested on several timetables. Then, the quality of the result is assessed by comparing the concurrent scheduler, the VDL scheduler and the optimal solution. The quality of the result of the concurrent scheduler is on average 12% better than the VDL scheduler.

Besides the benefits to computation time and the quality of the results, the concurrent scheduler also supports more features than the previous solution methods. However, the concurrent scheduler does not guarantee the optimal solution. A drawback of this scheduler is that charging during rush hours is not discouraged or prohibited, resulting in the addition of extra buses during rush hours when other buses are charging.

Because the concurrent scheduler is fast, and the solution it gives is close to or the same as the optimal solution for small problems, the conclusion is drawn that the concurrent scheduler is a good way to obtain a feasible solution quickly.

Chapter 4

Additions to concurrent scheduler

The concurrent scheduler as described and tested in the previous chapter is able to supply a feasible solution quickly. However, the algorithm is not perfect. The additions on the concurrent scheduler as given in this chapter are based on the idea that it is beneficial to charge outside of the rush hours. In this chapter, multiple possible improvements are proposed, whereof some are applied. The methods chosen here are picked because of their ease of implementation and the fact that they do not have a random factor, and are thus repeatable. In addition to the possible improvements, the ability to limit the number of chargers per charging location is added. Then, the quality of these alterations is tested. Finally, the conclusion on the additions is given.

4.1 Rush hour

As described in the previous chapter, it is not wise to charge during rush hours. To be able to prevent this, the rush hours need to be defined first. In this section, this is briefly investigated. In Figure 4.1, the number of simultaneous service trips that are present in timetable seven can be found. From this figure it becomes clear that the morning rush hour ends around 9:00 and the afternoon rush hour starts around 14:00 for this timetable. For this remainder of this chapter these times are used as the rush-hour times. Please note that the magnitude and the extent of the rush hours are timetable dependent.

4.2 Decrease charging costs during non-rush hours

One idea to let the concurrent scheduler reduce charging during rush hours is the lowering of the charging costs during non-peak hours. For the rush hours the time intervals of 7:00h to 9:00h and 14:00h to 18:00h are chosen. The time used to determine the charging costs is the end time of the service trip before that charging session. Since the charging takes 45 minutes and there is some deadhead time, the higher rate for charging starts one hour before the rush hours. The charging costs during the rush hour are €20, twice the rate chosen in the previous solution, outside the rush hours charging is free. In Table 4.1 the results can be found.

The biggest decrease in daily costs are seen for schedule 8, this can be seen in Table 4.2. It turns out this is caused by the fact that sometimes, a bus starts charging a service trip earlier than previously, reducing the total number of buses needed. On average, the method where charging during non-rush hours is free has $0,4\%$ lower costs on these test schedules. This is a small improvement and it is uncertain how this difference develops on other schedules.

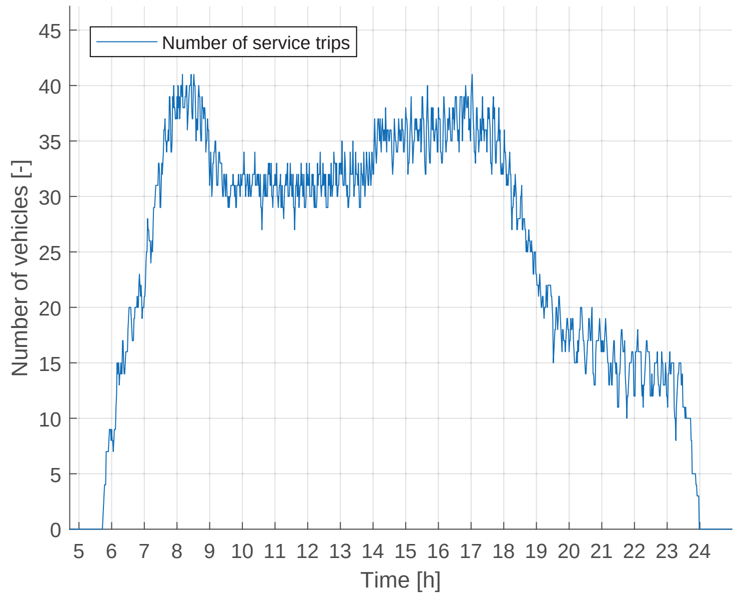


Figure 4.1: Number of service trips that are conducted simultaneously

Table 4.1: Results of concurrent scheduler with increased charging costs during rush hour

Test Schedule number	Computation Time [s]	Cost per day	Number of buses used	Energy used	Distance driven [km]	Number of charging sessions
1	1,19	€971	6	1066	710	2
2	1,22	€971	6	1066	710	2
3	1,28	€966	6	1048	699	2
4	3,37	€2250	11	3443	2296	11
5	3,68	€2055	10	3088	2059	12
6	6,36	€2055	10	3088	2059	12
7	67,08	€11490	58	16969	11313	52
8	73,11	€11159	57	15957	10638	57
9	214,30	€11294	57	15864	10576	73

Table 4.2: Comparison between concurrent scheduler with increased charging costs during rush hour and the standard concurrent scheduler

Test Schedule number	Concurrent Scheduler	Concurrent Scheduler - Variable charging costs	Improvement
1	€971	€971	0.0 %
2	€971	€971	0.0 %
3	€966	€966	0.0 %
4	€2250	€2250	0.0 %
5	€2057	€2055	0.1 %
6	€2110	€2055	2.6 %
7	€11665	€11490	1.5 %
8	€11256	€11159	0.9 %
9	€11136	€11294	-1.4 %

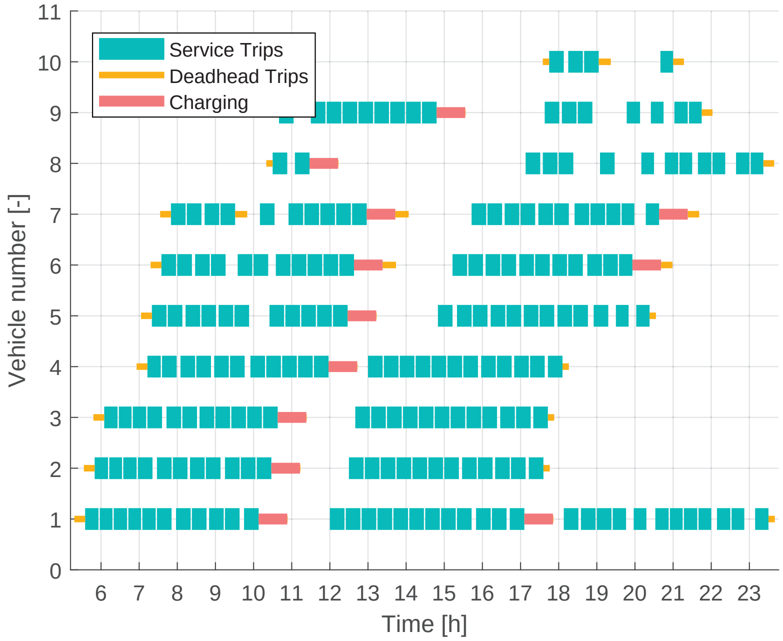


Figure 4.2: Schedule of test timetable 5 with increased charging before the afternoon rush hour

4.3 Higher SoC between rush hours

The next idea is to minimize the charging during the rush hours by ensuring that the buses enter that period with a higher SoC. To achieve this, the decision on which sequence is chosen for the current service trip is altered. No longer the cheapest option is always chosen, but the option where charging occurs is preferred. Because the charging and the deadhead trips combined take approximately an hour, the latest time that more charging should start is chosen to be 13:00h. As a begin time 11:00h is chosen. Note that it is not impossible that charging occurs outside this interval.

When planning a new trip, the end time of the previous trip is assessed. If this end time is within the higher SoC interval as determined earlier, and the SoC is lower than the increased lower bound of the SoC, the CSA considers a charging session. If the CSA would force a charging session every time, a lot of charging sessions would occur. For this section, it is chosen that the CSA lets every 5th planned trip go by a charger within the interval. An example of a resulting schedule can be found in Figure 4.2.

In Figure 4.2 it becomes clear that bus number eight starts charging earlier than previously, removing the need to charge at a later point. In Table 4.3 the results of all the test timetables can be found.

In Table 4.4, the results are compared. The new devised method is on average 0.4% worse than the standard concurrent scheduler. Furthermore, like the alteration where charging is free in between the rush-hours, it is not certain if this method always gives a better solution. Therefore, it is not advised to use this method.

Table 4.3: Results of concurrent scheduler with higher SoC before rush

Test schedule number	Computation time [s]	Cost per day	Number of buses used	Energy used [kWh]	Distance driven [km]	Number of charging sessions
1	1,41	€971	6	1066	710	2
2	1,36	€971	6	1066	710	2
3	1,50	€966	6	1048	699	2
4	4,02	€2.360	12	3436	2291	11
5	3,81	€2.057	10	3099	2066	12
6	5,88	€2.082	10	3154	2103	13
7	62,79	€11.530	58	17046	11364	54
8	60,94	€11.170	55	16232	10821	73
9	148,40	€11.395	57	16132	10755	76

Table 4.4: Comparison between concurrent scheduler with increased SoC before rush hour and the standard concurrent scheduler

Test Schedule number	Concurrent Scheduler	Concurrent Scheduler - Increased SoC	Improvement
1	€971	€971	0.0 %
2	€971	€971	0.0 %
3	€966	€966	0.0 %
4	€2250	€2360	-4.9 %
5	€2057	€2057	0.0 %
6	€2110	€2082	1.3 %
7	€11665	€11530	1.2 %
8	€11256	€11170	0.8 %
9	€11136	€11395	-1.6 %

4.4 Limited number of chargers

As stated before, the reason to use the concurrent scheduler is that it is quick, and that the results are used as an input for the next solution method. However, in the standard concurrent scheduler, the number of chargers at each charging location is assumed to be infinite. When this result is used as a starting point for a solution method where the number of chargers is limited, this method may be starting with an infeasible solution. This is not desirable. A method is developed to limit the number of chargers on a charging location in the concurrent scheduler. The method is devised in collaboration with and implemented by S.J.A Rutten. In this section, the alterations that are made with respect to the standard concurrent scheduler are explained. In addition to this, the results of the concurrent scheduler with limited chargers are given.

First, it is important to set the maximum number of chargers per charging location. The next step is to generate an array which states the number of available chargers per charging location for each time increment. Then, set the values of available chargers for all times to the maximum number of chargers on that location.

The CSA is altered to implement the limited number of chargers per charging location. Steps 1 and 2 of the CSA are not changed. In step 3 it is checked if the charging location that the arc goes to has at minimum one charger available for the duration of a charging session. This is done while taking into account the latest possible time to leave this charging location to start the next service trip on time. If there is no charger available at least once during this period, the sequence is removed. If there are enough chargers available in the interval, the charging session is planned as early as possible. An example of this can be seen in Figure 4.3. If no charging session can be planned, and no buses are available to complete the trips, a new bus is taken from the depot.

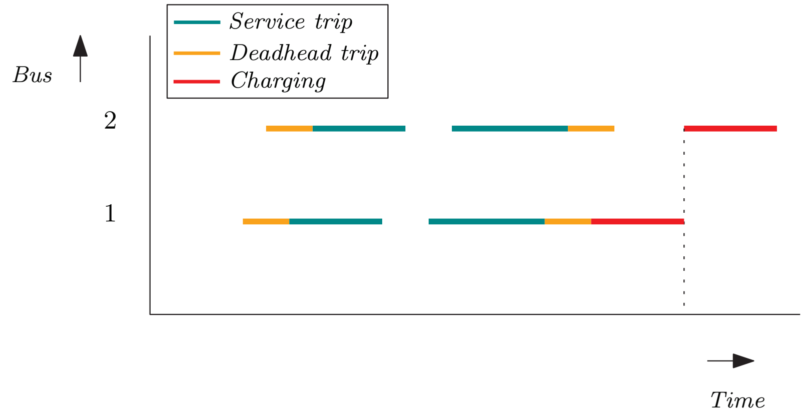


Figure 4.3: Shifting charging session with limited chargers

Table 4.5: Results of concurrent scheduler with limited number of chargers

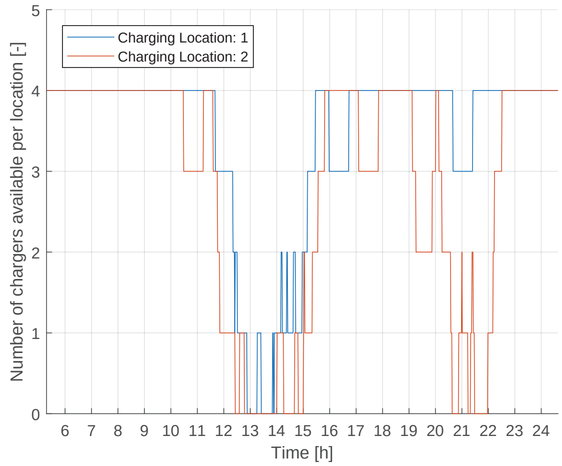
Test Schedule number	Computation Time [s]	Total cost per day	Number of buses used	Total energy used	Total distance driven [km]	Total number of charging sessions	Number of chargers per charging location	Number of chargers used without constraints
1	1,71	€971	6	1066	710	2	1	2
2	1,65	€971	6	1066	710	2	1	2
3	1,74	€966	6	1048	699	2	1	2
4	4,21	€2.134	10	3349	2233	13	2	4
5	5,60	€2.080	10	3109	2073	14	2	3
6	11,13	€2.076	10	3055	2037	15	2	3
7	111,81	€13.391	74	17395	11596	53	4	14
8	78,21	€11.449	55	16528	11019	93	4	12
9	127,81	€11.302	54	16320	10880	95	4	15

The next alteration is in step 4 of the CSA. Here, the charger availability array is updated. With these alterations the number of chargers used per charging location never exceeds the limit. The results of the test timetables with limited number of chargers can be found in Table 4.5.

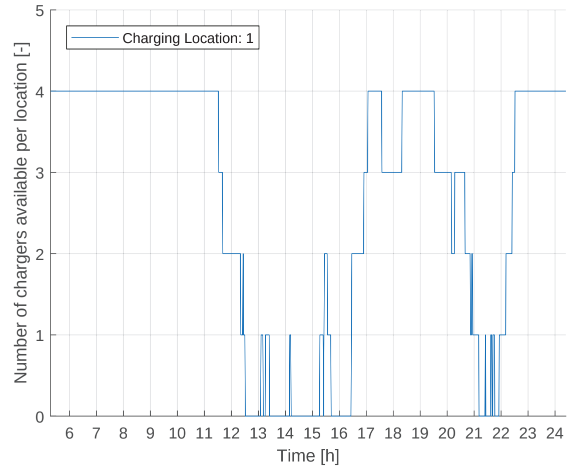
In Figure 4.4a, the number of available chargers over time can be found for test timetable eight. It becomes clear that between 12:30h and 14:00h intermittently all chargers are in use at both charging locations. For test timetable eight, the objective function is 1.7% worse than the solution with unlimited chargers. The number of available chargers for test timetable seven can be found in Figure 4.4b, the objective function for this schedule is 14.8% worse. Note that for both test timetables, the number of chargers available at the charging locations is four. Therefore, the total number of chargers for test timetable seven is more strict than for test timetable eight.

4.5 Results of additions to concurrent scheduler

In this section, the results of the different improvements methods are compared to the standard concurrent scheduler.



(a) Test timetable 8



(b) Test timetable 7

Figure 4.4: Number of available chargers per charging location for different test timetables

Table 4.6: Results of different schedule methods on test time timetables

Test Schedule number	Concurrent Scheduler	Concurrent Scheduler - Variable charging costs	Improvement	Concurrent Scheduler - Higher SoC before rush	Improvement
1	€971	€971	0.0%	€971	0.0%
2	€971	€971	0.0%	€971	0.0%
3	€966	€966	0.0%	€966	0.0%
4	€2.250	€2.250	0.0%	€2.360	-4.9%
5	€2.057	€2.055	0.1%	€2.057	0.0%
6	€2.110	€2.055	2.6%	€2.082	1.3%
7	€11.665	€11.490	1.5%	€11.530	1.2%
8	€11.256	€11.159	0.9%	€11.170	0.8%
9	€11.136	€11.294	-1.4%	€11.395	-2.3%
	Average Improvement		0.4%		-0.4%

From Table 4.6 it becomes clear that adding variable charging costs has an average improvement of 0.4% on the costs for the test timetables. The second proposed alteration, requiring a higher SoC between the rush hours, has a negative effect of 0.4% on the costs. Note that for these test timetables in combination with the chosen battery size the charging in the standard concurrent scheduler the charging mainly does not occur during rush hours. This could be a reason why the suggested improvement methods do not have a significant improvement. Note that each alterations is made on the standard concurrent scheduler. The alterations are not combined and tested.

Besides the options that are conducted here, other methods could be thought of as well. For instance vary the start SoC of the buses or implement random charging session between rush hours.

4.6 Conclusion on additions to the concurrent scheduler

In this chapter two suggested improvements are implemented into the concurrent scheduler algorithm. These suggested alterations to the concurrent scheduler do not have significant improvements. It is therefore not advised to implement these methods. Beside these improvements the concurrent scheduler is expanded to support a limited number of chargers per charging location. This feature is useful because in practice, the number of chargers per charging location is limited. Furthermore, this feature expands the possibility of the concurrent scheduler solution to be used as an initial solution for another solution method.

Chapter 5

Column generation

In this chapter, the column generation technique is explained using an example, where for simplicity the goal is to minimize the number of buses that is used. The column generation technique has been first proposed by Ford and Fulkerson [31]. First, the example problem is given and a classic ILP formulation is given that solves this problem. Next, this ILP is re-formulated to apply column generation. After that, the example problem is solved step-by-step using the column generation technique. Finally, the benefit the column generation technique can provide is explained and the conclusion is given.

5.1 Definition of example problem

In this section, an example timetable is given for which a schedule is made using two methods: a classic ILP formulation and the column generation algorithm. For these schedulers, the goal is to minimize the number of buses. The constraints are, that all the service trips need to be performed and that only compatible service trips are performed by the same bus. It is assumed that buses do not use energy, and thus, charging or energy levels are not taken into account. Furthermore, all the service trips begin and end at the depot.

5.1.1 Timetable of example problem

The example timetable consists of five service trips, with begin times $h^{\text{start}} = [1, 2, 3, 4, 5]^T$ and end times $h^{\text{end}} = [3, 4, 5, 6, 7]^T$. Because the begin and end locations coincide, no travel time is present from any end to any start location. In this timetable, the maximum number of simultaneous service trips is two, giving a lower bound on n^b, the number of buses that are needed. An upper bound is given by the number of service trips in the timetable, in this case five. Furthermore, the minimum time between trips, h^{gap} is set to zero. With the *comp* matrix as defined in (3.1), the compatibility matrix for this example becomes:

$$comp = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{5.1}$$

5.2 Using an ILP to solve the example problem

One possibility to solve the scheduling problem is to use an Integer Linear Program (ILP). A straightforward formulation that can be used is explained in this section. The decision variable in this formulation is a_{tb}, which is 1 if service trip $t \in T$ is performed by bus $b \in B$ and 0 otherwise. The set T consists of all service trips. Note here that B represents the set of available buses, which should always be larger than or equal to the number of buses needed in the schedule, n^b.

Minimize over n^b:

$$n^b, \tag{5.2a}$$

subject to:

$$b a_{tb} \leq n^b \qquad \forall\, t \in T, b \in B, \tag{5.2b}$$

$$a_{t_1 b} + a_{t_2 b} \leq 1 \qquad \forall\, t_2 > t_1;\; comp(t_1, t_2) = 0;\; t_1, t_2 \in T;\; b \in B, \tag{5.2c}$$

$$\sum_{b \in B} a_{tb} = 1 \qquad \forall\, t \in T, \tag{5.2d}$$

$$a_{tb} \in \{0, 1\}, \tag{5.2e}$$

$$n^b \in \mathbb{Z}^+. \tag{5.2f}$$

Constraint (5.2c) states that for each bus, no two service trips that are incompatible are allowed in the solution. The second constraint, (5.2d), states that each service trip must be performed by exactly one bus.

The number of decision variables is $|T||B| + 1$. With an upper bound on $|B|$ of $|T|$, the upper bound on the number of decision variables is $|T|^2 + 1$. When charging is added, or a larger timetable is considered, the number of decision variables is considerably higher. In the research of Monhemius [1], charging and deadhead trips are added to this approach and the vehicle scheduling problem is solved. However, the computation times are impractical.

The model as described above is implemented in MATLAB. The documentation can be found in Appendix B.2. Next, problem (5.2) is solved and the resulting solution is given by:

$$\left[\begin{array}{c} a_{11} \\ a_{21} \\ a_{31} \\ a_{41} \\ a_{51} \\ a_{12} \\ a_{22} \\ a_{32} \\ a_{42} \\ a_{52} \\ \hdashline a_{13} \\ \vdots \\ a_{55} \\ \hdashline n^b \end{array}\right] = \left[\begin{array}{c} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ \hdashline 0 \\ \vdots \\ 0 \\ \hdashline 2 \end{array}\right].$$

The number of used buses is 2, where the first bus performs service trips $1, 3$, and 5 and the second bus performs service trip 2 and 4.

5.3 Reformulated problem

In the previous section, it became clear that it is possible to formulate an ILP to solve a scheduling problem. The decision variable was which bus performed which trip. However, this is not required. In this section, the problem is re-formulated.

The concept of a vehicle task is introduced first. A vehicle task consists of all the service trips that are performed by a bus during a day. The set of all vehicle tasks that exist is denoted by V, wherein each column represents a vehicle task. The goal is to use the fewest buses as possible. This is equivalent to using the fewest vehicle tasks. For the example problem $X_{tv} = V$, where X_{tv} consists of x_{tv} which is 1 if service trip $t \in T$ is performed by vehicle task $v \in V$ and 0 otherwise.

$$V = X_{tv} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Above, the matrix V is given for the example problem. The number of columns is 12 because with the given constraints the number of possible vehicle tasks is 12. An upper bound on the number of vehicle tasks is $2^{|T|} - 1$, which is the number of possible vehicle tasks when all service trips are compatible. Since only feasible vehicle tasks are present in the set V, the compatibility constraint does not need to be added. The decision variable is u_v, which is 1 if vehicle task v is in the solution and 0 otherwise. The objective function is to minimize the number of vehicle tasks used, thereby minimizing the number of buses.

Minimize over u_v:

$$n^b = \sum_{v \in V} u_v, \tag{5.3a}$$

subject to:

$$\sum_{v \in V} x_{tv} u_v = 1 \qquad \forall\, t \in T, \tag{5.3b}$$

$$u_v \in \{0, 1\} \qquad \forall\, v \in V. \tag{5.3c}$$

Equation (5.3b) implies that for all the columns that are selected combined, each service trip must be performed once. Furthermore, the decision variable u_v must be binary. This means that it is not allowed to let half a bus drive a vehicle task. Problem (5.3) is solved using an integer solver and the result can be found below:

$$u_v^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

From this result, it becomes clear that two vehicle tasks are selected and therefore two buses are used. The ninth and the tenth vehicle task are selected from V. This means that the first bus performs service trip $1, 3$ and 5 and the second bus performs service trips 2 and 4, which is the same result as using the formulation in Section 5.2.

This formulation does not seem to be an improvement. One reason is that the number of decision variables in problem (5.3) grows exponentially with the number of service trips, $2^{|T|} - 1$, whereas the number of decision variables in problem (5.2) grows polynomially, $|T|^2 + 1$. The second reason is that, for larger problems, it is not possible to enumerate the vehicle tasks and thus, it is impossible to obtain the set V. Furthermore, even if it was possible to find V, problem (5.3) would have many integer decision variables and thus, would be hard to solve.

The idea behind column generation is, that it is not required to start with an entire set of vehicle tasks V, but to start with a few vehicle tasks and then iteratively find new vehicle tasks. When the iteration phase is completed, fewer vehicle tasks are needed than are present in V. This is possible because in the set V, multiple columns are present which are inefficient. In the example problem, columns 6, 8 and 12 in V are inefficient because the bus is inactive for large portions of the day.

The technique described above is called column generation because the goal is to find new vehicle tasks, which are represented by columns. When the number of decision variables is large and the number of constraints is relatively low in problem (5.3), this technique is particularly beneficial. In the next section the formulations of the steps of column generation are given.

5.4 Formulating column generation

In this section, the formulation of the steps of the column generation algorithm are given. These steps are applied to the example problem of the previous section. A flowchart of the column generation algorithm can be found in Appendix A.1.

5.4.1 Restricted master problem

The first step is to solve a linear relaxation of problem (5.3), which is solved on a subset of V, called V'. The Master Problem (MP) is described in (5.3), and the new formulated, (5.4), is called the Restricted Master Problem (RMP). The RMP has to ensure that each service trip is driven at least once.

Minimize over u_v:

$$n^b = \sum_{v \in V'} u_v, \tag{5.4a}$$

subject to:

$$\sum_{v \in V'} x_{tv} u_v \geq 1 \qquad \forall\ t \in T, \tag{5.4b}$$

$$u_v \geq 0 \qquad \forall\ v \in V'. \tag{5.4c}$$

Here, (5.4b) states that each service trip must be performed at least once and (5.4c) states that non-negative portion of buses can be assigned to a vehicle task.

5.4.2 Dual of restricted master problem

When the dual of the RMP as described in (5.4) is solved, shadow prices are obtained. These can be used to find a new vehicle path. The shadow prices are also calculated when the RMP is solved using the simplex method. Below, the dual of the RMP is given, where π_τ are the decision variables.

Maximize over π_τ:

$$\sum_{\tau \in T} \pi_\tau, \tag{5.5a}$$

subject to:

$$\sum_{\tau \in T} X^T_{tv} \pi_\tau \leq 1 \qquad \forall\, v \in V', \tag{5.5b}$$

$$\pi_\tau \geq 0 \qquad \forall\, \tau \in T. \tag{5.5c}$$

The resulting decision variables π_τ are also called the shadow prices. Each constraint in the RMP has a corresponding shadow price in the dual. These shadow prices can be interpreted as the amount of improvement to the objective of the RMP, when the corresponding constraint is relaxed by one. Thus, when a constraint is inactive, the corresponding shadow price is zero. For column generation, the shadow prices are used to find a new column to add to V'.

5.4.3 Subproblem

The subproblem is used to find a new vehicle task to add to V', with the use of the shadow prices as found in the dual of the RMP. From the formulation it becomes clear that the number of integer decision variables in the subproblem is equal to the number of dual variables, and thus, to the number of constraints in the RMP. This is why the column generation technique is efficient when the number of constraints is relatively low. The new vehicle task j has to improve the solution of the RMP to the biggest extent. When no column can be found that improves the RMP, the set V' is sufficient and no more columns need to be added.

The decision variable in the subproblem is $\delta_{\tau j}$, which is one if service trip $\tau \in T$ is in j and zero otherwise. Furthermore, c^v_j is the costs of the new column j and π_τ the shadow prices as determined in the dual of the RMP.

Minimize over $\delta_{\tau j}$:

$$c^v_j - \sum_{\tau \in T} \pi_\tau \delta_{\tau j}, \tag{5.6a}$$

subject to:

$$\delta_{\tau_1 j} + \delta_{\tau_2 j} \leq 1 \qquad \forall\, comp(\tau_1, \tau_2) = 0;\ \tau_2 > \tau_1;\ \tau_1, \tau_2 \in T, \tag{5.6b}$$

$$\delta_{\tau j} \in \{0, 1\}. \qquad \forall\, \tau \in T. \tag{5.6c}$$

In constraint (5.6b) it is stated that no two service trips are allowed in the new vehicle task j when these service trips are incompatible. The number of constraints this equation describes is the number of zero elements above the diagonal in the compatibility matrix, as described in (3.1). The value of the objective function in the subproblem is also called the reduced cost, since it gives the quantity by which the objective of the RMP could decrease by adding the newly found column. If the reduced cost is negative, the new column is added to the set V' and the dual of the RMP (5.5) is solved again.

5.4.4 Solving to integer solution

When the RMP is solved for the known set of columns V', it is not certain, even unlikely that all the decision variables u_v are integers. This is not an issue when searching for new vehicle tasks. However, when all the required columns are added and an integer solution is desired, this can become an issue. To circumvent this problem, the MP can be applied on the set V'.

Minimize over u_v:

$$n^b = \sum_{v \in V'} u_v, \tag{5.7a}$$

subject to:

$$\sum_{v \in V'} x_{tv} u_v = 1 \qquad \forall\ t \in T, \tag{5.7b}$$

$$v_p \in \{0, 1\} \qquad \forall\ v \in V'. \tag{5.7c}$$

Note here that the number of decision variables in u_v is equal to the number of columns in V'. This means that for larger problems, where possibly more columns are present in V', it is hard to find an integer solution using an integer solver. For now, it is assumed that finding an integer solution is possible.

If using an integer solver is not possible, the RMP can be solved. The solution for u_v can be rounded up to the nearest integer. This way, it is ensured that each service trip is performed. However, it is possible that some service trips are performed more than once.

In the next section the column generation algorithm is applied to the example problem as described in Section 5.1.

5.5 Applying column generation to example problem

In this section, the column generation algorithm as described above is applied to the example problem used in this chapter. First, the subset of V, called V', is defined. For this initial set for V', the RMP needs to be feasible. Furthermore, each vehicle task present in V' must be feasible for a bus. In this example, the identity matrix is chosen as V'. This means that one bus is assigned to each service trip, where that bus does not perform any other service trips during that day. This is an inefficient schedule since each bus only driving a short period of the day. However, the usage of the identity matrix as the initial set V' ensures that the RMP is feasible. Note here that any feasible schedule is allowed as an initial set for V'.

$$V' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

5.5.1 Iteration 1

Using this starting solution for V', the dual of the RMP, problem (5.5) is solved. Here, c^v, the cost of each vehicle task is set to one and $X_{tv} = V'$.

Dual problem Problem (5.5) is applied to the example problem, the resulting optimization problem is stated below:

Maximize:

$$\pi_1 + \pi_2 + \pi_3 + \pi_4 + \pi_5, \tag{5.8a}$$

subject to:

$$0 \leq \pi_\tau \leq 1 \qquad \forall\, \tau \in T \tag{5.8b}$$

The resulting shadow prices π_τ can be found below:

$$\pi_\tau^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Subproblem These shadow prices are then applied to the subproblem as described in (5.6). This be found below:

Minimize:

$$1 - 1\delta_{1j} - 1\delta_{2j} - 1\delta_{3j} - 1\delta_{4j} - 1\delta_{5j}, \tag{5.9a}$$

subject to:

$$\begin{aligned} \delta_{1j} + \delta_{2j} &\leq 1, \\ \delta_{2j} + \delta_{3j} &\leq 1, \\ \delta_{3j} + \delta_{4j} &\leq 1, \\ \delta_{4j} + \delta_{5j} &\leq 1, \end{aligned} \tag{5.9b}$$

$$\delta_{\tau j} \in \{0, 1\}\ \forall\, \tau \in T \tag{5.9c}$$

The resulting proposed new column j is then:

$$\delta_{\tau j} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix},$$

which has a reduced costs of -2 and thus, can be added to the set V'. The next iteration can start by solving the dual problem. The new set V' is given by:

$$V' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

5.5.2 Iteration 2

For this iteration, the same steps as for iteration 1 are performed.

Dual With the newly obtained array V', the dual of the RMP is solved again and the shadow prices π_τ are: $[0, 1, 1, 1, 0]^T$.

Subproblem With these shadow prices, the subproblem is solved and the proposed new column becomes:

$$\delta_{\tau j} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

The reduced costs of the column above is -1, which is negative. Therefore, the column is added to the known set V', which becomes:.

$$V' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

5.5.3 Iteration 3

With this newly obtained array V', the dual is solved again.

Dual The resulting shadow prices can be found below:

$$\pi_\tau^T = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Subproblem The subproblem is solved and the proposed column is the following:

$$\delta_{\tau j} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

The reduced costs of this column is 0. This is not negative and therefore, the proposed column is not added to V'. When the reduced cost is non-negative it is certain that no column exists that can improve the solution of the RMP. For this example, two columns are added to the original five. These seven vehicle tasks are sufficient to solve problem (5.4) to optimality.

5.5.4 Solving to integer solution

When the reduced costs in the subproblem become non-negative, the MP can be solved for V' and the solution is obtained. For the example, the number of decision variables is seven. The solution can be found below:

$$u_v^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

From here, it becomes clear that the sixth and seventh column are chosen. Therefore, one bus drives service trip $1, 3$ and 5 and one bus drives service trip 2 and 4, which means that two buses are used in total and that the solution is the same as the solution of problem (5.2) and problem (5.3).

5.6 Conclusion on column generation

The example problem as described in Section 5.1 is solved using different solution methods. First, the problem is solved using an ILP directly. The number of decision variables for this formulation increases polynomially. Next, the problem is re-formulated. In this reformulation, the number of integer decision variables grows exponentially. The column generation technique is applied on the example problem and it is shown that it is not necessary to add all possible vehicle tasks.

The problem is split up into a global problem, the MP/RMP, and a local problem, the sub-problem, where the global problem ensures that each service trip is driven, and directs the local problem to find a good solution. The local problem does not have to ensure that each service trip is driven, just that the gain on the RMP is maximized and that the new column is feasible for a bus. The main advantage of column generation is illustrated because not all of the possible columns need to be enumerated, but only important columns are added iteratively.

Chapter 6

Application of column generation to electric vehicle scheduling

In this chapter, a model is formulated that is based on the column generation technique, which is used to solve an electric vehicle scheduling problem. The model explained in this chapter is an elaboration of the model explained in Chapter 5. The first extension is that a limit on the number of chargers on a charging location is taken into account. Furthermore, a limit on the power that is drawn from the grid on the charging location is considered. As before, only one depot and one charging location is considered, which is the begin and end location of all service trips. In this chapter, the modeling decisions are explained first. Next, the formulation of each step of the column generation algorithm is given. Then, multiple stopping criteria are discussed. Finally, it is explained how an integer solution is obtained.

6.1 Modeling decisions

An important decision to make is to choose what decisions should be made by the master problem and what decisions by the subproblem. A multitude of possibilities exist.

One possibility is that the subproblem is to find a vehicle task, that can be performed on a single battery charge. Then, the MP/RMP becomes to choose which vehicle tasks are performed by the same bus. The effect is that the MP/RMP has to ensure that enough charging occurs between the vehicle tasks. Furthermore, the begin and end times of the vehicle tasks have to be taken into account.

Another possibility is to discretize time and to construct the subproblem as a Shortest Path Problem (SPP). An example of this can be found in the work of Kooten Niekerk [26] and Posthoorn [18]. If both the time and the SoC are discretized the subproblem becomes a SPP, which can be solved in polynomial time. If the SoC is not discretized, the subproblem becomes an SPP with resource constraints. It is important to note here that while solving a SPP is relatively easy, constructing the SPP while taking the shadow prices into account is non-trivial. In the work as described earlier, the size of the graph was reduced by removing unreachable nodes.

For this research, it is chosen that the definition of the vehicle task is not altered with respect to the previous chapter. Thus, the vehicle task is defined as the combination of tasks a vehicle performs during an entire day. Therefore, the decision which service trip is performed and when charging occurs is present in the subproblem. The effects of this decision are discussed in section 6.2.5

To simplify modeling it is assumed that charging is linear. Furthermore, the time is discretized into $|Z|$ timeblocks, to simplify tracking the number of chargers that are used at the same time.

6.2 Model formulation

In this section, each step of the column generation algorithm is explained. The constraints applied to the MP/RMP represent the restrictions that apply to the complete system. Because of the decision made in the previous section, the global constraints are that each service trip is performed, the number of chargers used at the same time does not exceed the number of available chargers, and the charging power on the charging location is not exceeded. The local constraints are the constraints of a vehicle task. No incompatible trips are performed in the same vehicle task, energy is only added to a vehicle that is connected to a charger. Furthermore, the amount of energy that is added to a bus connected to a charger is within the lower and upper bound. No charging occurs during driving and the energy level in the battery does not exceed the lower and upper bounds. The time is discretized into n^z timeblocks. Each timeblock is denoted by $z \in Z$ where Z is the set of timeblocks. In Appendix A.3, a flowchart can be found of each step performed in the formulation used in this chapter.

6.2.1 Explanation of vehicle task

In the previous section, the choice is made that the subproblem consists of finding a vehicle task that spans the entire day and that time is discretized. With the addition of charging, this means that the vehicle task includes the service trips that the bus performs, the time slots that are used to charge, and how much energy is charged during those time slots.

The vehicle task consists of three parts. The first part, X_{tv} is the same as defined in the previous chapter, where x_{tv} is one if service trip t is performed in vehicle task v. As described in the previous section, the time is discretized into $|Z|$ time blocks. In the new vehicle task, $s_{zv} \in S_{zv}$ is one if charging occurs during time block z in vehicle task v and zero otherwise. Here, S_{zv} is a matrix whereas S is the collection of charging locations. Furthermore, the amount of energy that is added to the bus needs to be determined. This information is stored in E_{zv} where e_{zv} is the amount of energy that is charged during time block z in vehicle task v.

In the previous chapter, it is stated that the identity matrix can be used as an initial set of columns V, since this describes that each bus performs a single service trip. The same choice is made for the model in this chapter. Therefore, the array X_{tv} is set to the identity matrix. The matrices S_{zv} and E_{zv} are zero matrices. This results in a feasible solution for the RMP, since each trip can be performed and no chargers are used.

6.2.2 Master Problem

In this section, problem (5.3) is extended to support the vehicle tasks as described above. As in problem (5.3), the decision variable is u_v, which is one if vehicle task v is used in the solution and zero otherwise. It is no longer assumed that each vehicle task has a cost of one. The parameter c_v^v is introduced that expresses the costs of vehicle task $v \in V$. The objective is to minimize the total cost of the solution. In this formulation, n^s is the number of chargers that is available. The parameter $\epsilon_z^{\text{s',max}}$ indicates the maximum amount of energy that can be delivered by the grid to the charging location during time block z.

Minimize over u_v:

$$\sum_{v \in V} c_v^v u_v, \tag{6.1a}$$

subject to:

$$\sum_{v \in V} x_{tv} u_v = 1 \qquad \forall\, t \in T, \tag{6.1b}$$

$$\sum_{v \in V} s_{zv} u_v \leq n^s \qquad \forall\, z \in Z, \tag{6.1c}$$

$$\sum_{v \in V} e_{zv} u_v \leq \epsilon_z^{\text{s',max}} \qquad \forall\, z \in Z, \tag{6.1d}$$

$$u_v \in \{0, 1\} \qquad \forall\, v \in V. \tag{6.1e}$$

Constraint (6.1b) states that each trip must be performed once. Constraint (6.1c) ensures that the number of simultaneous charging sessions does not exceed the number of available chargers for every time block and the final constraint, (6.1d), dictates that the amount of energy that is added to the buses by all the used vehicle tasks does not exceed the capability of the grid connection in each time block.

6.2.3 Restricted Master Problem

The same alterations as in section 5.4.1 are performed. First, the integrality constraint is relaxed and the master problem is solved for a subset of vehicle tasks, $V' \subset V$. Furthermore, constraint (6.1b) is altered to ensure that each trip is performed at least once.

Minimize over u_v:

$$\sum_{v \in V'} c_v^v u_v, \tag{6.2a}$$

subject to:

$$\sum_{v \in V'} x_{tv} u_v \geq 1 \qquad \forall\, t \in T, \tag{6.2b}$$

$$\sum_{v \in V'} s_{zv} u_v \leq n^s \qquad \forall\, z \in Z, \tag{6.2c}$$

$$\sum_{v \in V'} e_{zv} u_v \leq \epsilon_z^{\text{s',max}} \qquad \forall\, z \in Z, \tag{6.2d}$$

$$u_v \geq 0 \qquad \forall\, v \in V'. \tag{6.2e}$$

The interpretation of the constraints in the RMP is not changed with respect to the MP. This interpretation can be found below problem (6.1).

6.2.4 Dual of restricted master problem

Next, the dual of problem (6.2) is constructed. Dual variables π_τ, θ_ζ and ρ_ζ are introduced. When the dual is solved, the shadow prices are obtained.

Maximize over π_τ, θ_ζ and ρ_ζ:

$$\sum_{t\in T} \pi_\tau - \sum_{\zeta\in Z} \theta_\zeta n^s - \sum_{\zeta\in Z} \rho_\zeta \epsilon^{\text{s,max}}, \tag{6.3a}$$

subject to:

$$\sum_{t,\tau\in T} X^T_{tv}\pi_\tau + \sum_{z,\zeta\in Z} S^T_{zv}\theta_\zeta + \sum_{z,\zeta\in Z} E^T_{zv}\rho_\zeta \le c^v_v, \qquad \forall\, v \in V' \tag{6.3b}$$

$$\pi_\tau \ge 0 \qquad \forall\, \tau \in T, \tag{6.3c}$$

$$\theta_\zeta \ge 0 \qquad \forall\, \zeta \in Z, \tag{6.3d}$$

$$\rho_\zeta \ge 0 \qquad \forall\, \zeta \in Z. \tag{6.3e}$$

Here, the shadow prices π_τ state how expensive it is to perform each service trip τ, θ_ζ gives the costs of exceeding the number of chargers available in time block ζ and ρ_ζ of exceeding the power limit of the charging location in time block ζ.

6.2.5 Subproblem

In this section, the subproblem is given and explained. The goal of the subproblem is to obtain a new vehicle task that improves the solution of the RMP the most. The subproblem uses the shadow prices as given by the dual problem to find the new vehicle task. From the definition of the vehicle task in section 6.2.1, it becomes clear that the number of decision variables in the subproblem is $|T| + 2|Z|$. The decision variables $\delta_{\tau,j}, \sigma_\zeta$ and ϵ_ζ are introduced, where binary decision variable $\delta_{\tau,j}$ is one if service trip $\tau \in T$ is performed by new vehicle task j and zero otherwise. The binary decision variable σ_ζ is one if the bus is using a charger during time block $\zeta \in Z$ and zero otherwise. Finally, ϵ_ζ is the amount of energy that is added to the bus during time block ζ. Thus, there are $|T| + |Z|$ integer decision variables and $|Z|$ continuous decision variables in the subproblem.

The cost of energy is c^e, the minimum energy level in the bus is $e^{\text{b,min}}$, the maximum energy level in the bus is $e^{\text{b,max}}$ and $\epsilon^{\text{s,min}}$ is the minimum amount of energy that has to be charged during a time block if a bus is connected to a charger. Furthermore, e^t_τ is the energy that service trip $\tau \in T$ requires. In the case described earlier, the costs for a unit of energy is fixed. If time of day pricing of energy is required, c^e is altered to c^e_z, which is the cost of energy in timeblock $z \in Z$.

As in problem (5.6), the objective function is the cost of the new vehicle task, c^v_j, minus the gain that can be achieved, depending on the decision variables and the shadow prices. Here, the cost of the new vehicle task is not set to one, but to the price of the bus, c^b, plus the cost of the energy that is charged during the vehicle task. The cost of a vehicle task is calculated as:

$$c^v_j = c^b + \sum_{\zeta\in Z} \epsilon_\zeta c^e. \tag{6.4}$$

With (6.4), the subproblem becomes (6.5). Note that c^b is a constant and can be ommited from the objective function, since the solution is the same. However, in this research c^b is kept in the objective function to be able to use the objective function value as the reduced cost directly.

Minimize over $\delta_{\tau,j}, \sigma_\zeta$ and ϵ_ζ:

$$c^b - \sum_{\tau \in T} \pi_\tau \delta_{\tau,j} + \sum_{\zeta \in Z} \theta_\zeta \sigma_\zeta + \sum_{\zeta \in Z} \epsilon_\zeta (c^e + \rho_\zeta), \tag{6.5a}$$

subject to:

$$\delta_{\tau_1,j} + \delta_{\tau_2,j} \leq 1 \qquad \forall \ \text{comp}(\tau_1, \tau_2) = 0; \tau_2 > \tau_1; \tau_1, \tau_2 \ \in T, \tag{6.5b}$$

$$\delta_{\tau,j} + \sum_{\zeta = h_\tau^{\text{start,z}}}^{h_\tau^{\text{end,z}}} \sigma_\zeta \leq 1 \qquad \forall \ \tau \in T, \tag{6.5c}$$

$$-\sigma_\zeta M + \epsilon_\zeta \leq 0 \qquad \forall \ \zeta \in Z, \tag{6.5d}$$

$$\epsilon^{\text{s,min}} \sigma_\zeta - \epsilon_\zeta \leq 0 \qquad \forall \ \zeta \in Z, \tag{6.5e}$$

$$\sum_{1}^{\tau} \delta_{\tau,j} e_\tau^t - \sum_{\zeta=1}^{h_\tau^{\text{end,z}}} \epsilon_\zeta \leq e^{\text{b,max}} - e^{\text{b,min}} \qquad \forall \ \tau \in T, \tag{6.5f}$$

$$\sum_{\zeta=1}^{h_\tau^{\text{start,z}}-1} \epsilon_\zeta - \sum_{1}^{\tau-1} \delta_{\tau,j} e_\tau^t \leq 0 \qquad \forall \ \tau \in T, \tag{6.5g}$$

$$\delta_{\tau,j} \in \{0, 1\} \qquad \forall \ \tau \in T, \tag{6.5h}$$

$$\sigma_\zeta \in \{0, 1\} \qquad \forall \ \zeta \in Z, \tag{6.5i}$$

$$0 \leq \epsilon_\zeta \leq \min\{\epsilon^{\text{s,max}}, \epsilon^{\text{s',max}}, \epsilon^{\text{b,max}}\} \qquad \forall \ \zeta \in Z. \tag{6.5j}$$

The functions $h_\tau^{\text{start,z}}$ and $h_\tau^{\text{end,z}}$ are explained first. These functions state in which time block $\zeta \in Z$ service trip τ begins and ends. In Figure 6.1, an example can be found with two service trips. In this example, there are five time blocks and the time domain spans from 400 to 450. It can be seen that service trip 1 starts during the first time block and ends during the second. Thus, $h_{\tau_1}^{\text{start,z}}$ is 1 and $h_{\tau_1}^{\text{end,z}}$ is 2.

The first constraint states that only compatible trips are allowed in the new column. The second constraint ensures that a bus driving a service trip cannot be connected to a charger. In constraint (6.5d), the big-M method is used to ensure that energy can only be added when the bus is connected to a charger. Here, M is a sufficiently large number. Equation (6.5d) is only correct if M is larger than the maximum value of ϵ_ζ. Thus, $M > \min\{\epsilon^{\text{s,max}}, \epsilon^{\text{s',max}}, \epsilon^{\text{b,max}}\}$, where $\epsilon^{\text{s,max}}$ is the maximum amount of energy a charger can deliver during a time period, $\epsilon^{\text{s',max}}$ is the maximum amount of energy a charging location can deliver and $\epsilon^{\text{b,max}}$ is the maximum amount of energy a bus can charge during a time period. In this research, $M = 1000$ is used.

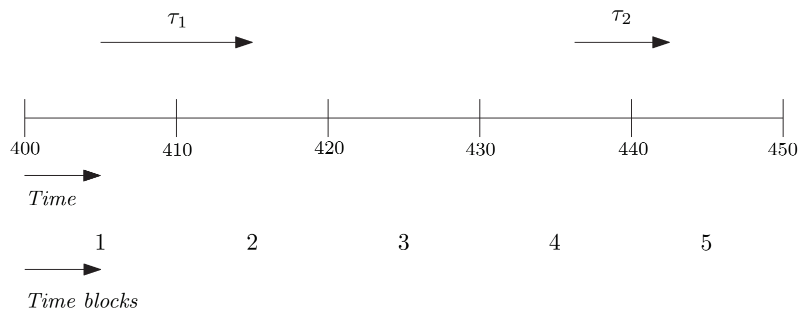


Figure 6.1: Example for determining overlap time blocks and service trips

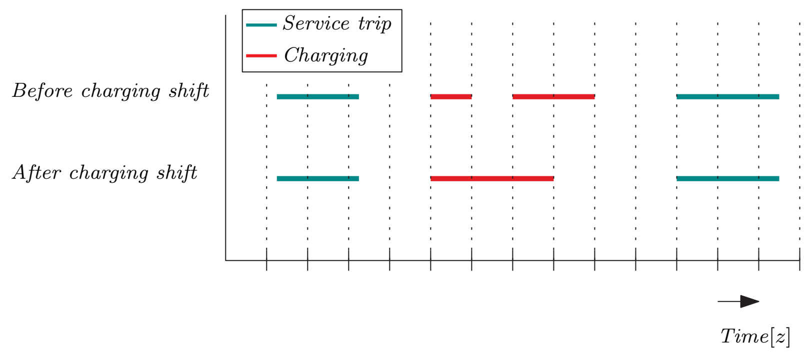


Figure 6.2: Comparison charging before and after shifting charge sessions

Constraint (6.5e) states that if a bus is connected to a charger, a minimum amount of energy is added to the bus. To ensure that the energy level of the bus does not go below the lower bound, constraint (6.5f) is added. Here, the sum of the energy of all the trips performed up to and including the current service trip minus the energy charged has to be above the minimum energy level. Constraint (6.5g) states that the energy level of the bus cannot exceed the upper bound, by ensuring that the energy that is added to the bus is always less than the consumed energy.

Even though the goal is to find a new column that improves the solution of the RMP the most, it is not required to solve the subproblem to optimality for each iteration. As long as the reduced cost is negative, the column can improve the solution of the RMP. However, to prove that no column exists that improves the solution of the RMP, the reduced costs needs to be non-negative when the subproblem is solved to optimality.

In the formulation of the subproblem, a bus is not prohibited to connect and disconnect from a charger multiple times in between service trips. A constraint can be formulated that prevents multiple charging sessions in between service trips. In this research, an other solution approach is used. Here, charging sessions in between service trips are shifted to be consecutive after the first charging session in between service trips. An example of the difference before and after shifting can be found in Figure 6.2. Note here that this has an impact on the reduced costs of the proposed new vehicle task.

6.2.6 Stop criteria

In the previous chapter, the column generation algorithm was halted when the reduced cost became non-negative. When this is the case, it is certain that no vehicle task exists that can improve the solution of the RMP. However, for larger problems, the number of columns that

Table 6.1: Stop criteria

Stop criterion	Meaning
1	No column exists that improves the result of the RMP
2	Improvement on the objective value of RMP is too low over numerous iterations to continue
3	Maximum number of iterations reached
4	Maximum computation time reached

need to be added until the reduced cost is non-negative can be large. A well known effect of the column generation technique is the tailing-off effect. More information about the tailing off effect can be found in the work of Lübbecke [32]. This means that the greatest improvement on the RMP is obtained in the first iterations, where later iterations have a lower benefit to the objective function of the RMP. It can therefore be said that if it is not required to solve to optimality it can be useful to stop iterating before the reduced cost is non-negative.

In Table 6.1, the stopping criteria used in this research are given. The first stop criterion is that the reduced cost is non-negative. Another possibility is to stop iterating when the solution of the RMP has not improved above a set amount over a finite number of iterations. Other possibilities are to set a limit on the number of iterations or computation time. It is important to note that when any stop criterion is used other than the first, it is no longer guaranteed the set of vehicle tasks is sufficient to find the globally optimal solution.

6.2.7 Solving to an integer solution

The column generation algorithm iterates between the RMP, dual, and the subproblem to find more vehicle tasks to add to V'. As explained in the previous section, at some point the algorithm is stopped. At that point, the set V' is known and the search for an integer solution for u_v can start. If the set V' is small enough, an integer solver can be used to find the vehicle tasks that are used. This is caused by the fact that the number of integer decision variables is equal to the number of vehicle tasks in V'. If the set V' consists of many vehicle tasks, the computation time when an integer solver is used can be long.

To circumvent this problem, an other method can be used to find an integer solution. In the previous chapter, the RMP was solved and any decision variables that were larger than zero were rounded up to one. This resulted in possibly driving service trips more than once. However, with a limit on the number of chargers and the limit on the grid capacity, this is no longer possible. For example, if either constraint (6.2c) or constraint (6.2d) is active when the RMP is solved and the non-integer decision variables are rounded up to one, at least one constraint is violated, resulting in infeasibility. In this research, a simple method is proposed, based on rounding.

In Figure 6.3, the proposed rounding algorithm is shown by a flowchart. First, the RMP is solved on set V' that has been obtained with the column generation algorithm. The solution to u_v could be, and probably will be, non-integer. However, if some decision variables are one, equality constraints are added that fix these decision variables to one. Then, the algorithm repeats. Another option is that no decision variable, that has not been fixed to one before, is one. Then, the decision variable closest to one is set to one. If a decision variable is rounded to one, it is possible to start the column generation algorithm again to add a couple of new vehicle tasks. Using the method as described above, it it possible that an integer solution is obtained using only a problem with continuous decision variables.

The first important note here is that not adding any additional columns might be possible,

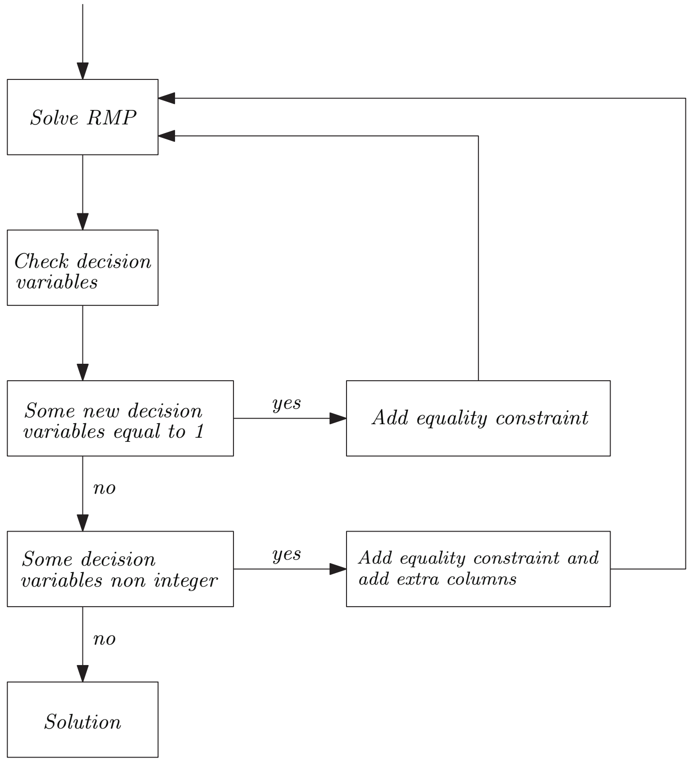


Figure 6.3: Flowchart of performed steps to find integer solution using linear solver

but the quality of the solution can suffer. Ideally, columns are added until the reduced cost is non-negative. However, this can result in high computation times. The second note is that the method described above has a large drawback, namely that is not guaranteed that a feasible integer solution is obtained. More sophisticated methods to find an integer solution exist, whereof some can be found in the work of Kooten Niekerk [26] and Pepin [14]. Furthermore, once a decision variable is rounded, it is no longer certain that the solution is globally optimal.

6.3 Results of model

In this section, the test timetables as described in section 3.3 are scheduled using the model based on column generation as described by (6.1)-(6.5). From now on, this is referred to as the column generation model. First, the values for the parameters that are used are explained. Next, the results of the schedules for the test timetables made by the column generation model are presented. Then, the quality of the solution is compared with the concurrent scheduler heuristic. After that, the effect of increasing the number of iterations and the decrease in quality caused by using the rounding algorithm is discussed. Furthermore, it is investigated if using an other initial set of vehicle tasks can improve the results. Finally, the conclusion and recommendations are given.

6.3.1 Results column generation

The column generation model is used to schedule electric vehicles for the timetables as described in section 3.3. To be able to solve the problem, some parameters need to be known. These are similar as described in section 3.1. Since the model does not support multiple locations, it is

Table 6.2: Results from column generation on test timetables, with unlimited number of chargers

	Concurrent Scheduler			Column Generation					
	Computation time [s]	Costs	#Buses	Computation time [s]	Number of iterations	Stop Criterion	Mean trips	Costs	#Buses
Timetable 1	1.85	€628	5	4.10	22	1	1,000	€474	4
Timetable 4	4.54	€1294	9	873.75	200	3	1.074	€1292	9
Timetable 7	57.63	€7007	52	-	-	-	-	-	-
Timetable 10	1.78	€255	2	4.46	49	1	1.231	€240	2

assumed that all the service trips described in the timetables start and end at the depot location. Furthermore, all the available chargers are located at the depot. The minimum time between trips h^{gap} is set to 1 [min]. In addition to this, a battery capacity of 216 [kWh] is chosen whereof 80% is available. All the vehicles use 1.5 [kWh/km], regardless of the operating conditions. The depreciation cost of a bus is 111.11 [€/day]. The price of energy is set at 0.20 [€/kWh]. The stop criteria as described in section 6.2.6 are used. The second stop criterion is that the RMP should improve more than 1% over 200 iterations. Thus, the value of the RMP should be less than 99% of the value of the RMP 200 iterations earlier. The maximum number of iterations is set to 250 and the maximum computation time is set to two hours. Please note that these stop criteria are applied to the iterative part of the column generation algorithm and do not include the search for an integer solution. As an initial solution, each service trip is assigned a unique bus, where the cost for each of these vehicle tasks is set to 1000. For timetable 1, the number of time steps is set to 50, for all the other timetables 100 time steps are used.

As explained in section 6.2.5, it is not always required to solve the subproblem to optimality. To reduce the computation time, a time limit of 60 seconds is set to the computation time of the subproblem. Please note that this time has to be sufficient to find a feasible solution, with a negative reduced cost, for the subproblem. Finally, the number of service trips in timetable 10 is set to 13.

The column generation model is implemented in MATLAB and the documentation can be found in Appendix B.3.2. Next, the main results of the column generation model are presented. First, the results are given when the number of chargers is unlimited. For these results, an integer solver is used to obtain an integer solution.

From Table 6.2, it can be seen that when the column generation model is used to schedule the trips, the resulting costs is lower than the costs using the concurrent scheduler for test timetable one, four and ten. For test timetable one this is because fewer buses are used. For test timetable four and ten his is caused by the fact that the resulting costs are the costs of the buses plus the costs of the energy. The column generation model can choose how much energy is added, where the concurrent scheduler always charges fully. This means that where the column generation can end the day at the minimum SoC, the concurrent scheduler can have a higher SoC. Furthermore, it becomes clear that test timetable seven is not computable. This is caused by the number of integer decision variables in the subproblem, which is equal to the number of service trips $|T|$ plus the number of time steps $|Z|$. For this instance, the number of integer decision variables is $1097 + 100 = 1197$, which is too large for the integer solver used. In addition to this, the computation time is longer for the column generation than the concurrent scheduler. Finally, it becomes clear that for test timetable 4 and 10 the mean number of service trips performed is larger than one. This means that some service trips are present in more than one vehicle task that is chosen. In practice, these trips would not be performed multiple times.

Next, the column generation model is used to solve the test timetables where the number of chargers is limited. For test timetable 1, 4 and 10 the number of chargers is set to one, and for test timetable 7 it is set to four. In Table 6.3 the results are given when the number of chargers

Table 6.3: Results from column generation on test timetables, with limited number of chargers

	Concurrent Scheduler			Column Generation					
	Computation time [s]	Costs	#Buses	Computation time [s]	Number of iterations	Stop Criterion	Mean trips	Costs	#Buses
Timetable 1	1.88	€715	6	4.38	20	1	1.143	€575	5
Timetable 4	4.38	€1.294	9	1712.30	200	3	1.857	€9465	26
Timetable 7	81.92	€7.902	64	-	-	-	-	-	-
Timetable 10	1.68	€255	2	4.75	47	1	1.231	€240	2

(a) Unlimited number of chargers **(b)** Limited number of chargers

Figure 6.4: Gantt charts of test timetable 4 with unlimited and limited number of chargers

is limited. From this table it becomes clear that the resulting costs is lower when the column generation model is used than when the concurrent scheduler is used for timetable one and ten. The reason is the same as for the situation with an unlimited number of chargers. For timetable 4, the case with a limited number of chargers has a better result than the concurrent scheduler, while for the case with a limited number of chargers the result is worse, almost by an order of magnitude.

From Figure 6.4b it becomes clear that for the case with unlimited chargers each bus drives multiple service trips, where for the case with a limiting amount of chargers some buses only drive a single service trip. These vehicle tasks are the initial vehicle tasks, with high costs. Since each of these vehicle tasks have a cost of 1000, the resulting cost is high. The question is then, how do these bad vehicle tasks end up in the solution. One reason could be that not enough iterations of the column generation algorithm are completed. This can cause that some service trips are not present in newly generated vehicle tasks, forcing the solver to pick an initial column to ensure that each service trip is performed. In the next section, the effect of increasing the number of iterations is briefly investigated.

6.3.2 Increasing the number of iterations

In this subsection, the effect of increasing the number of iterations for the column generation algorithm is investigated. As stated before, it is beneficial to generate a higher amount of columns. This is because each iteration can improve the solution of the RMP. It is important to note that the RMP is a linear problem, whereas the MP is an integer linear program. It is not guaranteed that an improvement on the RMP also means that the MP is improved. Test

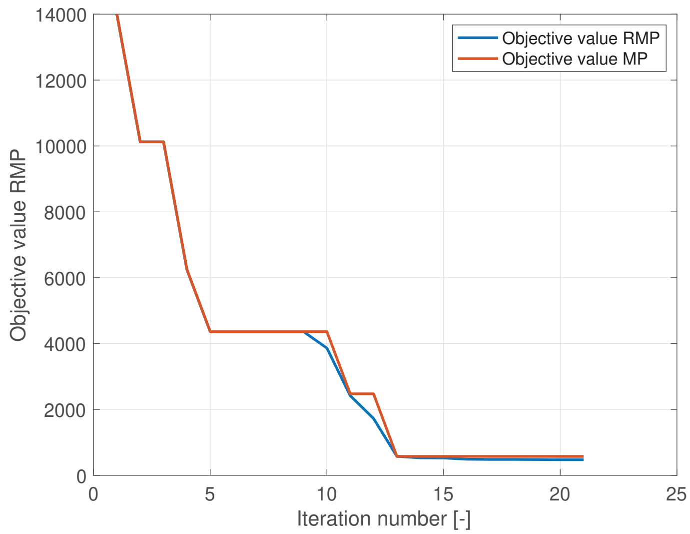


Figure 6.5: Comparison objective value of RMP and MP for multiple iterations

Table 6.4: Comparison of results on timetable four with with different number of iterations

Computation time [s]	Number of iterations	Stop criterion	Mean trips	Costs	#Buses
1055.70	100	3	1.803	€ 11359	27
2262.80	250	3	1.8571	€ 9465	26

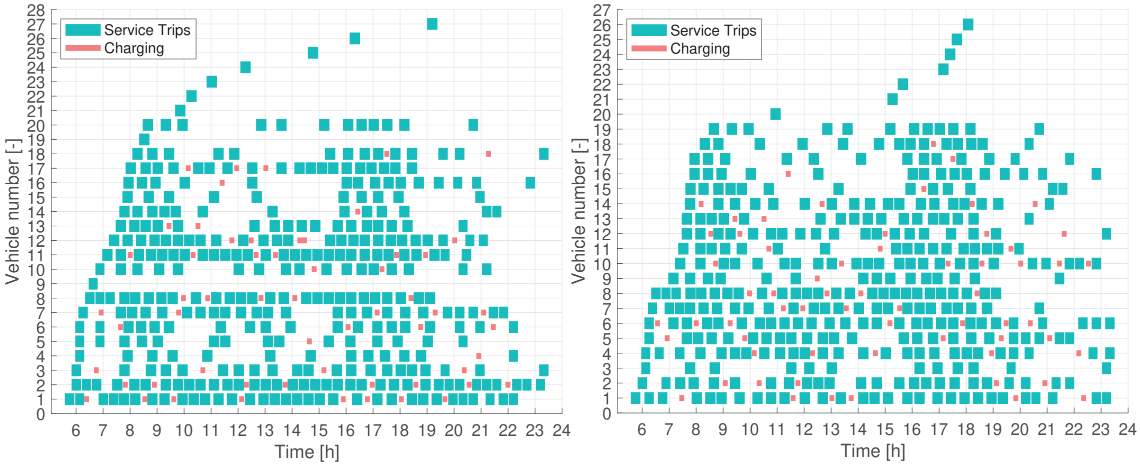
timetable 1 is used as an example first, where in Figure 6.5 the objective function of the RMP is shown for each iteration. Furthermore, for each iteration, the objective function of the MP is given as well, when the MP is solved for the set V'.

From Figure 6.5, it can be seen that when for each iteration the RMP is solved, the solution improves. For this example, the MP is solved on the known columns in each iteration too. As is expected, the solution of the MP is always higher or equal to the solution of the RMP, since the decision variables have to be integer. Furthermore, it becomes clear that when the solution of the RMP improves, the solution of the MP does not necessarily improve. Next, the column generation algorithm is used to schedule test timetable four where the maximum number of iterations is set to 100 and 250 respectively.

From Table 6.4 it can be seen that indeed, increasing the number of iterations can be beneficial. It is important to note that in both solutions, original columns with disproportionately high costs, are present. This can be seen in Figure 6.6. This could indicate that, also for the case with more iterations, more columns should be added. However, note that when more columns are added, the problem becomes harder to solve using an integer solver. After a certain point, one has to resort to an other method to obtain an integer solution.

6.3.3 Effects of the rounding algorithm

When the search for columns has been completed, the solution can be calculated. However, when many columns are present in V', an integer solver can no longer be used to obtain an integer solution. In this research, using a linear solver in combination with rounding has been proposed. To not let the rounding algorithm benefit from having more vehicle tasks than the integer solver, the number of extra columns that are generated if rounding is applied is set to zero. As described in section 6.2.7, it is not certain that a feasible solution is obtained. However,



(a) Test timetable four with 100 iterations **(b)** Test timetable four with 250 iterations

Figure 6.6: Gantt charts for test timetable four with different number of iterations

in the tests as described in this research a feasible solution is always obtained. In this section, using an integer solver and the rounding algorithm to find an integer solution is compared. The test timetable used here is test timetable four, where the number of chargers is one. The limit on the number of iterations is set to 100. In Table 6.5 the results can be found.

Table 6.5: Comparison between using rounding to find integers and using an integer solver for test timetable four

Solution method	Computation time [s]	Number of iterations	Stop criterion	Mean trips	Costs	#Buses
Integer solver	1058.50	100	3	1.803	€ 11359	27
Rounding	968.77	100	3	1.7241	€ 13394	30

From this table it becomes clear that for this case, the rounding algorithm provides a worse solution than the case where an integer solver is used.

Next, test timetable one is scheduled using both the solution methods, where both the situation with a limited number and unlimited amount of chargers is scheduled. In Table 6.6 the results are given.

Table 6.6: Comparison between using rounding to find integers and using an integer solver for test timetable one

Solution method	Number of chargers	Computation time [s]	Number of iterations	Stop criterion	Mean trips	Costs	#Buses
Integer solver	4 chargers	3,39	22	1	1,0000	€ 474	4
Rounding	4 chargers	3,44	22	1	1,0000	€ 474	4
Integer solver	1 charger	3,67	20	1	1,1429	€ 575	5
Rounding	1 charger	3,70	20	1	1,4286	€ 699	6

From this table it becomes clear that for test timetable one, when the number of chargers is not limiting, the result is identical. The linear solver does not give an all integer solution in the first iteration of the rounding algorithm. However, the same vehicle tasks are used as in the solution of the integer solver. When the number of chargers is limiting, the rounding algorithm and the integer solver give a different solution. It can be seen that the rounding algorithm uses six vehicle tasks, and thus buses, where the integer solver uses five buses. As a reminder, the concurrent scheduler algorithm also uses six buses in this case.

6.3.4 Warm start

Up to this point, the initial set V' is constructed by assigning a unique bus to each service trip. This means that the number of vehicle tasks when the column generation algorithm starts is equal to the number of service trips, n^t. In the formulation as described in section 6.2, no columns are removed from V' and thus, the number of integer decision variables when an integer solution is desired is at least n^t.

In this section, it is investigated if starting with a different set of vehicle tasks V' is beneficial. Since the initial set has to be feasible for the RMP, not all sets of vehicle tasks are sufficient as an initial set. A heuristic can be used to provide an initial set. Using a better initial set might be beneficial because of two reasons.

The first reason is that a heuristic most probably assigns buses more efficiently than using each bus for just one trip. This means that fewer buses, and thus vehicle tasks, are present in the initial set V'. This increases the number of vehicle tasks that can be generated by the column generation algorithm, before an integer solver can no longer be used to find an integer solution, possibly increasing the quality of the solution.

The second reason that using a warm start can be beneficial, is the possibility that higher quality vehicle tasks are generated by the column generation algorithm. If the identity matrix is used, the first iterations have a large impact on the value of the objective function of the RMP. However, this does not mean that the generated vehicle tasks are good vehicle tasks, just that a large improvement is made on the bad initial solution. The first vehicle tasks generated by the column generation algorithm are tasks where service trips are combined that can be driven by a single vehicle. It is possible that when a warm start is used fewer iterations of the column generation algorithm are needed. Thereby improving the computation time. Note that finding an initial feasible solution does have a computational costs itself.

In this section, the test timetables are solved using the column generation algorithm with the identity matrix and with the solution of the concurrent scheduler as described in section 3 as an initial set of vehicle tasks. To ease using the concurrent scheduler as an initial solution, it is assumed that buses do not consume energy and thus, do not need charging. This assumption can be dropped, when the charging timeblocks as described in section 6.2 are taken into account in the concurrent scheduler. In the column generation algorithm equality constraints are added to fix the decision variables in the subproblem, σ_ζ and ϵ_ζ, to zero. With these alterations, the column generation algorithm is used to schedule the test timetables. An integer solver is used to obtain an integer solution.

In Table 6.7, the results are given. For each test timetable a cold and a warm start is considered. When the identity matrix is used as an initial solution, it is referred to as a cold start. On the other hand, if the solution of the concurrent scheduler is used as the initial solution, it is called a warm start. Furthermore, the computation time of the results with a warm start is the computation time of the concurrent scheduler combined with the computation time of the column generation algorithm. It can be seen that for test timetable 7, results are available where for the case where charging was possible results were not obtained. This is at least partly caused by the reduction in necessary decision variables. The computation time of test timetable 7 is higher than the other test timetables, since the number of service trips is higher, resulting in higher computation time of the subproblem.

Table 6.7: Results of test timetables solved by the column generation algorithm with warm and cold starts

Test timetable	Start	Total computation time [s]	Number of iterations	Stop criterion	Mean trips	Costs	#Buses
1	Cold	2,06	10	1	1,0714	€333,34	3
1	Warm	3,767	6	1	1	€333,34	3
4	Cold	17,14	200	3	1,0739	€888,89	8
4	Warm	13,17	200	3	1	€777,78	7
7	Cold	2048,70	200	3	1,4416	€6666,67	60
7	Warm	776,79	120	1	1,0009	€4999,99	45
10	Cold	1,66	1	1	1	€111,11	1
10	Warm	3,16	0	1	1	€111,11	1

Table 6.8: Comparison of results on timetable four with with increased charging costs between 11:00 and 15:00

Energy price	Computation time [s]	Number of iterations	Stop criterion	Mean trips	Costs	#Buses
Normal pricing	429.74	100	3	1.074	€1292	9
Higher pricing	423.15	100	3	1.09	€1475	10

For test timetable one, the number of iterations performed is reduced to six when the concurrent scheduler is used to provide an initial solution. However, the integer solution does not contain any of the newly generated vehicle tasks. The same is the case in test timetable ten, where no new column is generated if a warm start is used. When a cold start is used for test timetable ten, the globally optimal solution is obtained in one iteration.

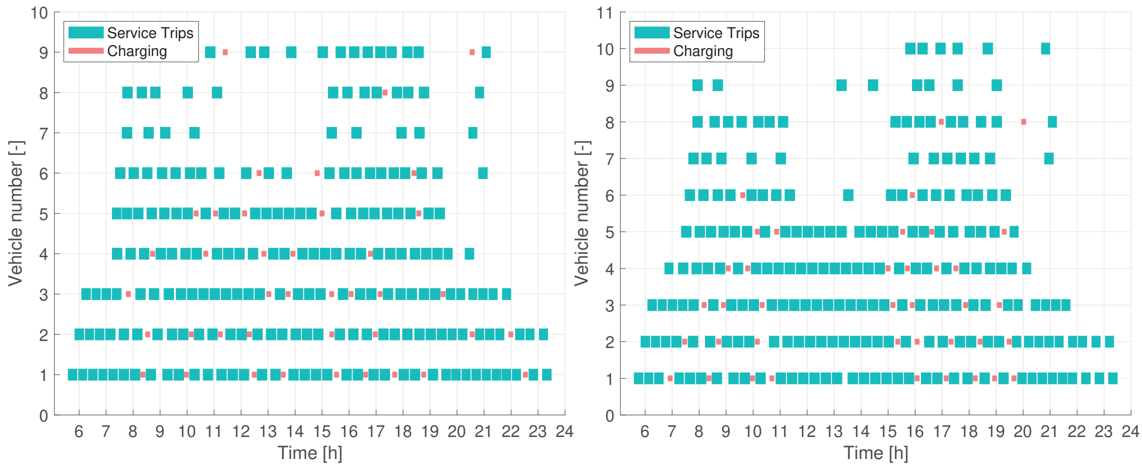
The most interesting results are the results of test timetable four and seven. Here, it becomes clear that when a warm start is applied, both the quality of the result and the computation time can be improved by using a warm start. However, the costs are the same as the costs of the initial solution provided by the concurrent scheduler. This could be caused by the simplification of the problem, without deadhead trips and energy consumption. It is uncertain how this difference develops if charging and energy consumption is included. With the results as shown above it becomes clear that a warm start is a promising research direction.

6.3.5 Time of day pricing

In the formulation given in section 6.2, adding time of day pricing is briefly explained. The cost of energy is altered from c^e to c^e_z, to provide the cost of energy for each timeblock $z \in Z$. In this section, as an example, test timetable four is solved where the price of energy is set 100 times higher between 11:00 and 15:00. In Table 6.8 the results can be found. From Figure 6.7 it becomes clear that when the charging price is increased for a certain period, less charging occurs in that period.

6.4 Modeling recommendations

The model as described in section 6.2 does have some drawbacks. One drawback is that, at least for the first iterations, the value of the objective function of the subproblem is not dependent on the time when charging occurs. Therefore, a multitude of combination of the decision variables result in the same objective value, increasing the computation time. Since no charging occurs



(a) Gantt chart when price is constant **(b)** Gantt chart when price is increased temporarily

Figure 6.7: Gantt charts for test timetable four with different charging costs

in the initial solution, no constraints in the subproblem regarding charging are active. Thus, all shadow prices associated with these constraints are zero. Therefore, it does not matter in which timeblocks charging occurs, as long as enough charging occurs between service trips and charging is not during service trips. To solve this problem, a small penalty can be added to the charging costs in the subproblem. For example, linearly increasing in time to ensure charging starts at the earliest possible moment. Note that this factor can be small, since the only goal is to make the model well defined.

The main drawback of the current implementation is the number of integer decision variables in the subproblem, which is $n^t + n^z$. Larger timetables are therefore harder to compute using this formulation. To reduce the size of the subproblem, the problem can be reformulated. If the subproblem is to provide the tasks that are performed on a single charge, the charging can be transferred to the MP/RMP. This eliminates the need to discretize in time, reducing the number of integer decision variables in the subproblem. The MP/RMP has to ensure that each trip is performed, that enough charging occurs between vehicle tasks and that the charging limits are not exceeded. Furthermore, the RMP/MP needs to decide which vehicle tasks are performed by the same vehicle. In this proposed formulation, non-linear charging and charger limits can be implemented in a similar manner as in the work of Monhemius [1]

A different solution method can be used as well. In the work of Kooten Niekerk [26], the subproblem is defined as a SPP. The subproblem is obtained by constructing a graph, where both time and SoC are discretized. Once the graph is obtained, the problem reduces to a SPP, which is solvable in polynomial time. If the SoC is not discretized, the subproblem can still be solved using a graph. The problem then becomes a shortest path problem with resource constraints. The difficulty of this method is constructing the graph. Kooten Niekerk reduced the size of the graph by combining nodes. Note that the size of the graph might become intractable quickly. Furthermore, it is important to keep track of the reduction of the graph, to apply the shadow prices correctly. Because of the difficulty constructing the graph, and applying the shadow prices correctly, it is not advised to use this method.

6.5 Conclusion and recommendations column generation

In this chapter, the modeling decisions are explained and the extended model is given. In addition to this, stop criteria are given and a method to find an integer solution is explained. After that, the results of the column generation model are given. Finally, the benefit of using a heuristic as an initial solution is discussed.

From the results it becomes clear that the column generation technique is a useful technique to solve larger integer problems. The schedules provided by the column generation model have lower costs than the schedules made by the concurrent scheduler heuristic. However, the computation time is longer. Most importantly, using the formulation as provided in this chapter, the column generation model is not able to schedule the largest test timetable. With formulation, it seems promising that these larger problems can be solved with the column generation algorithm as well. Furthermore, this reformulation can add extra features, like deadhead trips, non-linear charging and the implementation of multiple charging locations.

Furthermore, it is investigated if increasing the number of iterations is beneficial to the quality of the solution. For a single test scenario conducted in this research it is shown that it is beneficial to increase the number of iterations, as long as an integer solver can be used to find an integer solution. This is because it is shown that the rounding algorithm proposed in this research reduces the quality of the solution. Therefore, it is not advised to use the rounding algorithm.

In tests performed in this chapter, it became clear that using a heuristic to obtain the initial set of columns is useful. This warm start reduced the necessary iterations or, for the same number of iterations, reduced the costs of the solution. In addition to this, the computation time is reduced when larger problems are assessed. Because of these reasons, it is advised to use a heuristic to obtain an initial set of columns instead of assigning a unique bus to each service trip.

Finally, some recommendations are given. The main recommendation is to divide the MP and subproblem differently to reduce the size of the subproblem. Hereto, the definition of the columns needs to be altered. The advise given is to redefine the subproblem to find a vehicle task that can be driven on a single charge instead of finding a task for a bus for an entire day. The decision on which vehicle tasks are assigned to the same bus and how much charging should take place between these vehicle tasks is solved by the MP/RMP. This reformulation reduces the number of decision variables in the subproblem. Furthermore, the need to discretize time to take into account the charger usage is eliminated, further decreasing the number of integer decision variables.

Chapter 7

Conclusions and recommendations

In this thesis, two different scheduling methods for electric vehicles are constructed and applied. To find a feasible solution quickly, a concurrent scheduler heuristic is used in this research. This scheduler is implemented and compared with the scheduler previously developed by VDL. The concurrent scheduler gives, on average, 12% lower costs than the VDL scheduler, while supporting more features. It is advised to use the concurrent scheduler when a feasible solution is needed quickly, or as an initial solution for another solution method. Furthermore, the concurrent scheduler is expanded to support a limit on the number of chargers, increasing the usability of this heuristic to provide an initial solution.

However, the concurrent scheduler is not perfect. The main disadvantage is that charging during rush-hours is not discouraged. Another disadvantage is that the concurrent scheduler does not consider charging earlier than required, while this could reduce the number of simultaneous charging sessions. To reduce these effects, two methods are devised. Both of these methods are tested, where the effect on the quality of the result is low. Thus, is is not advised to implement these additions. In addition to this, the concurrent scheduler is expanded to limit the number of chargers on a charging location. This increases the practical employability of the concurrent scheduler. Both as a scheduler and as a provider of an initial solution for another solution method. The second solution method applied in this research is based on a column generation algorithm. A simple example is used to explain the column generation algorithm. Next, this example is used to show the reduction of integer decision variables. After that, the model is expanded to support electric vehicles, where the subproblem remains to find a vehicle task for a bus for an entire day. In addition to this, stop criteria are defined and a method is devised to obtain an integer solution. This method is based on rounding. The most important note is that the rounding algorithm is not guaranteed to provide a feasible solution. In addition to this, results show that when the rounding algorithm is used, the quality of the solution is lower than when an integer solver is used. When possible, it is advised to use an integer solver to find an integer solution instead of the proposed rounding algorithm.

The results where an integer solver is used, generally have a lower cost than the concurrent scheduler heuristic. An important condition to this results is that sufficient number of columns are generated. The column generation technique is useful to find a high quality solution. More features can be added to the column generation model. In this research, an example of time of day pricing of energy is given.

Considering the reduction in decision variables provided by column generation, it is advised to continue to use this technique to solve the vehicle scheduling problem. In the formulation given in this research, the size of the subproblem is limiting the computational efficiency of the column generation model. Therefore, it is advised to reformulate the column generation model, to reduce the size of the subproblem. Besides improving performance, this reformulation can include features like deadhead trips, multiple charging locations and non-linear charging. In addition to this, removing unnecessary vehicle tasks from the set of known vehicle tasks can improve the performance further.

Concluding, the column generation technique is a promising direction for future research. It is expected that with some reformulation the electric vehicle scheduling problem is solvable within reasonable time for the largest real life time tables.

Bibliography

[1] M. A. M. Monhemius, "Solving an Electric Vehicle Scheduling Problem using Mixed Integer Linear Programming," tech. rep., Eindhoven University of Technology, Department Mechanical Engineering, Dynamics and Control Research Group. Report number: DC 2019.047, Eindhoven, 2019.

[2] J. D. Adler, *Routing and Scheduling of Electric and Alternative-Fuel Vehicles*. PhD thesis, Arizona State University, 2014.

[3] L. Bodin, D. Rosenfield, and A. Kydes, "UCOST: a micro approach to a transportation planning problem," *Journal of Urban Analysis*, vol. 5, no. 1, pp. 47–69, 1978.

[4] S. R. Adheesh, M. Shravanth Vasisht, and S. K. Ramasesha, "Air-pollution and economics: Diesel bus versus electric bus," *Current Science*, vol. 110, no. 5, pp. 858–862, 2016.

[5] M. Mahmoud, R. Garnett, M. Ferguson, and P. Kanaroglou, "Electric buses: A review of alternative powertrains," *Renewable and Sustainable Energy Reviews*, vol. 62, pp. 673–684, 2016.

[6] "Staatscourant," *Ministerie van Economische Zaken*, no. 68651, p. 68, 2016.

[7] J. K. Lenstra and A. H. G. Rinnooy Kan, "Complexity of Vehicle Routing and Scheduling Problems," *Networks: And International Journal*, vol. 11, no. 2, pp. 221–227, 1981.

[8] O. Sassi and A. Oulamara, "Electric vehicle scheduling and optimal charging problem: complexity, exact and heuristic approaches," *International Journal of Production Research*, vol. 55, no. 2, pp. 519–535, 2017.

[9] O. J. Ibarra-Rojas, F. Delgado, R. Giesen, and J. C. Muñoz, "Planning, operation, and control of bus transport systems: A literature review," *Transportation Research Part B*, vol. 77, pp. 38–75, 2015.

[10] S. Bunte and N. Kliewer, "An overview on vehicle scheduling models," *Public Transport*, vol. 1, no. 4, pp. 299–317, 2009.

[11] J. D. Adler and P. B. Mirchandani, "The Vehicle Scheduling Problem for Fleets with Alternative-Fuel Vehicles," *Transportation Science*, vol. 51, no. 2, pp. 441–456, 2016.

[12] J.-Q. Li, "Transit Bus Scheduling with Limited Energy," *Transportation Science*, vol. 48, no. 4, pp. 521–539, 2014.

[13] P. C. Guedes and D. Borenstein, "Column generation based heuristic framework for the multiple-depot vehicle type scheduling problem," *Computers and Industrial Engineering*, vol. 90, pp. 361–370, 2015.

[14] A. S. Pepin, G. Desaulniers, A. Hertz, and D. Huisman, "A comparison of five heuristics for the multiple depot vehicle scheduling problem," *Journal of Scheduling*, vol. 12, no. 1, pp. 17–30, 2009.

[15] T. E. Morton and D. W. Pentico, *Heuristic Scheduling Systems*. New York, NY, USA: John Wiley & Sons, Inc., 1993.

[16] M. Gendreau, J. Y. Potvin, O. Bräysy, G. Hasle, and A. Løkketangen, "Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography," tech. rep., Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, Montréal, Québec, 2008.

[17] M. L. Pinedo and X. Chao, *Operation Scheduling: with applications in manufacturing and services*. Irwin/Mcgraw-Hill, 1999.

[18] C. Posthoorn, "Vehicle Scheduling of Electric City Buses: A Column Generation Approach," tech. rep., Delft University of Technology, Electrical Engineering, Mathematics and Computer Science Faculty, Department of Applied Mathematics, Delft, 2016.

[19] M. Wei, W. Jin, W. Fu, and X. N. Hao, "Improved ant colony algorithm for multi-depot bus scheduling problem with route time constraints," *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, pp. 4050–4053, 2010.

[20] H. Wang and J. Shen, "Heuristic approaches for solving transit vehicle scheduling problem with route and fueling time constraints," *Applied Mathematics and Computation*, vol. 190, no. 2, pp. 1237–1249, 2007.

[21] X. Xu, Z. Ye, J. Li, and C. Wang, "Solving a Large-Scale Multi-Depot Vehicle Scheduling Problem in Urban Bus Systems," *Mathematical Problems in Engineering*, vol. 2018, pp. 1–13, 2018.

[22] M. Schneider, A. Stenger, and D. Goeke, "The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations," *Transportation Science*, vol. 48, no. 4, pp. 500–520, 2014.

[23] N. Kliewer, T. Mellouli, and L. Suhl, "A time-space network based exact optimization model for multi-depot bus scheduling," *European Journal of Operational Research*, vol. 175, no. 3, pp. 1616–1627, 2006.

[24] J. Reuer, L. Wolbeck, and N. Kliewer, "The electric vehicle scheduling problem: A Study on time-space network based and heuristic solution approaches," *Proceedings of the 13th Conference on Advanced Systems in Public Transport*, no. July 2015, 2015.

[25] C. C. Lu, S. Yan, and Y. W. Huang, "Optimal scheduling of a taxi fleet with mixed electric and gasoline vehicles to service advance reservations," *Transportation Research Part C: Emerging Technologies*, vol. 93, no. June, pp. 479–500, 2018.

[26] M. van Kooten Niekerk, *Optimizing for Reliable and Sustainable Public Transport*. PhD thesis, Utrecht University, 2018.

[27] G. Desaulniers, J. Desrosiers, and M. M. Solomon, *Column Generation*. New York, NY, USA: Springer Science + Business Media Inc. 233 Spring Street, 2005.

[28] A. Löbel, "Vehicle Scheduling in Public Transit and Lagrangean Pricing," *Management Science*, vol. 44, no. 12-part-1, pp. 1637–1649, 1998.

[29] M. Huang and J. Q. Li, "The shortest path problems in battery-electric vehicle dispatching with battery Renewal," *Sustainability (Switzerland)*, vol. 8, no. 607, pp. 1–17, 2016.

[30] B. Golden, S. Raghavan, and E. Wasil, *The Vehicle Routing Problem: Latest Advances and New Challenges*. New York, NY, USA: Springer Science + Business Media Inc. 233 Spring Street, 2008.

[31] L. R. Ford and D. R. Fulkerson, "A Suggested Computation for Maximal Multi-Commodity Network Flows," *Management Science*, vol. 5, no. 1, pp. 97–101, 1958.

[32] M. E. Lübbecke and J. Desrosiers, "Selected Topics in Column Generation," *Operations Research*, vol. 53, no. 6, pp. 1007–1023, 2005.

Appendix A

Column generation algorithm flowcharts and structures

A.1 Flowchart of simplified model

In this section, the steps in column generation are visualized using a flowchart, Figure A.1. The first step is to formulate the Master Problem (MP). Global constraints are part of the MP. An example of a global constraint is the constraint that each service trip needs to be performed. The second step is to formulate the Restricted Master Problem by relaxing the integrality constraint in the MP and to solve for V', a subset of V. The third step is to solve the dual of the RMP, wherefrom the shadow prices for each constraint in the RMP is extracted. The fourth step is to find a new column/vehicle task using the subproblem. The shadow prices from the dual are an input to the subproblem. In the subproblem the local constraints are added. An example of a local constraint is the constraint that no incompatible service trips are allowed in the new vehicle task. After step four, the reduced cost is calculated. If the reduced cost is negative, the new column is added to V', and the algorithm is repeated from step three. The fifth and final step is to solve the MP for set V'. If the set V' is too large, the RMP can be solved and the decision variables are rounded up to integers.

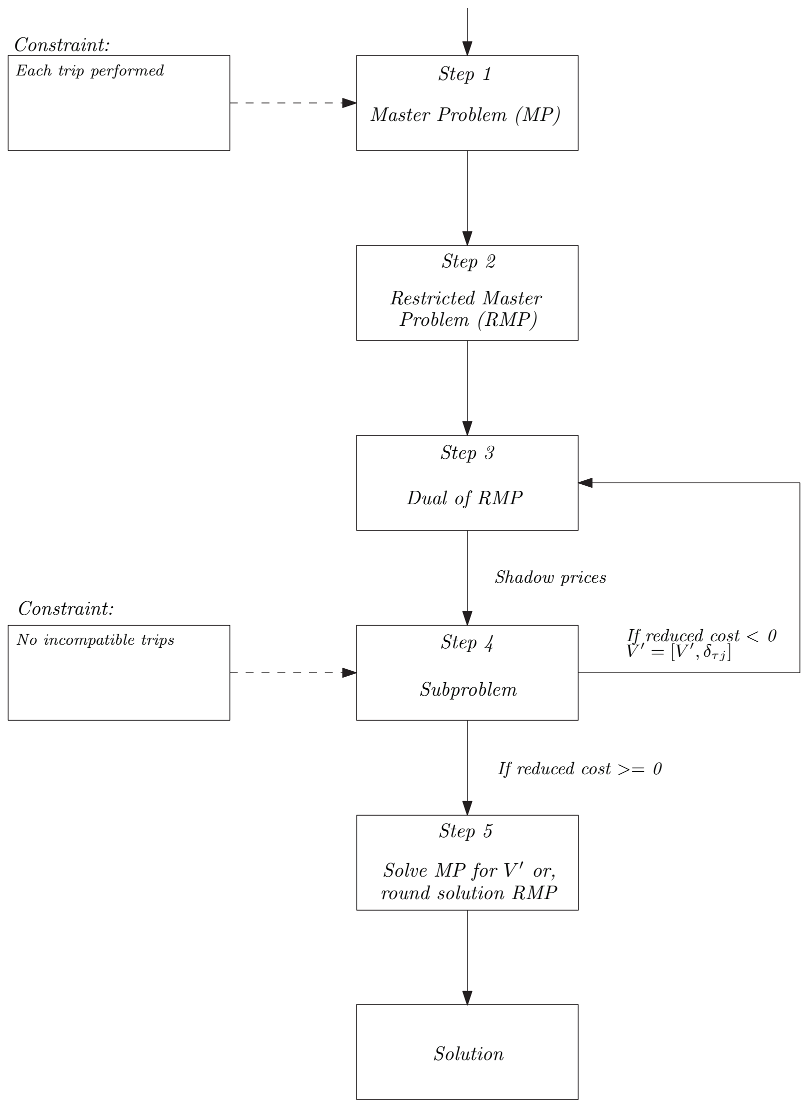


Figure A.1: Flowchart of steps in column generation algorithm

A.2 Structure of extended column

In Figure A.2 the structure of the extended columns can be found. Here, X_{tv} is an array with integer values, where x_{tv} is one if trip $t \in T$ is performed by vehicle task $v \in V$ and zero otherwise. The values in the integer array S_{zv} are ones if charging occurs during timeblock $z \in \zeta$ in vehicle task $v \in V$ and zero otherwise. The third part of the extended column is the E_{zv} array. Here, the amount of energy is given that is charged during timeblock $z \in Z$ in vehicle task $v \in V$.

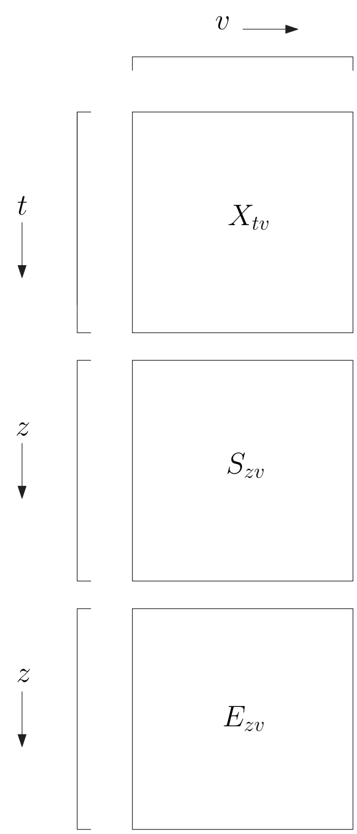


Figure A.2: Structure of extended column

A.3 Flowchart of model extended to support electric vehicles

This flowchart is similar as described in the previous section. Here, extra constraints are added to support the usage of electric vehicles and charging. Most importantly, if the RMP is used to obtain an integer solution, a non-integer variable is fixed and possibly more columns are added. This rounding algorithm is discussed in detail in section 6.2.7.

Figure A.3: Flowchart of steps in column generation algorithm with support for electric vehicles

Appendix B

Documentation of schedulers

B.1 Documentation of concurrent scheduler

In this document the input, the code and the output of the concurrent scheduler is explained. The main steps of the concurrent scheduler can be found in the report. Here the code is explained that uses these steps.

Input

Excel file

As an input for the time table and location information an excel file is used. The format is as follows: In the first sheet, called Time_Table, the service trips are given. In the table below an example can be found. Here the distance has to be given in meters.

From	Start	End	To	Dist
ehvbst	08:00:00	08:30:00	ehvapt	40000
ehvbst	08:15:00	08:45:00	ehvapt	40000
ehvapt	08:40:00	09:10:00	ehvbst	40000
ehvapt	08:55:00	09:25:00	ehvbst	40000
ehvbst	09:00:00	09:30:00	ehvapt	40000
ehvbst	09:15:00	09:45:00	ehvapt	40000

In the second sheet, called All_locations. The location names and the location numbers are mapped. Here all unique start and end locations for service trips are listed first, whereafter all the depots, and the charging locations. An example can be found in the table below.

Name	Number
ehvapt	1
ehvbst	2
Depot_1	3
Fuel_1	4

In the third sheet, called d the distances between all locations have to be given. Again, for distance the unit is meters. Here it is important to note that Depot_1 and Fuel_1 are at the same location. Furthermore, the distance is from the location in the first column toward the location in the other columns. An example can be found below.

	ehapt	ehvbst	Depot_1	Fuel_1
ehapt	0	11083	5833	5833
ehvbst	11083	0	9916	9916
Depot_1	5833	9916	0	0
Fuel_1	5833	9916	0	0

The next sheet, called *t* is similar to the sheet *d*. This sheet represents the travel time between the different locations. This travel time is a fraction of the day, thus 1 hour travel time is represented as 1/24.

	ehapt	ehvbst	Depot_1	Fuel_1
ehapt	0,0000	0,0132	0,0069	0,0069
ehvbst	0,0132	0,0000	0,0118	0,0118
Depot_1	0,0069	0,0118	0,0000	0,0000
Fuel_1	0,0069	0,0118	0,0000	0,0000

The next sheet called Depot states all the depot locations and their names.

Name
Depot_1

The last sheet called Charging gives the names of all the charging locations. Furthermore, an extra column can be added with the maximum number of chargers available at that location.

Name
Fuel_1

MATLAB

The first section in MATLAB is designated to the input of variables, and the input of the excel file as described earlier. In the following code the variables are stated:

```
%% Input

%Trips
h_gap           = 1;     %Minimal amount of time between trips

%Charging
c_e = 0.20;      %Cost of energy [euro/kWh]
h_s = 45;        %Charging time (time in that a vehicle can be charged fully)
c_s = 10;        %Cost of visiting a charging station

%Buses
c_b         = 500000/(15*300);  %Unit price of the considered bus type
e_b_max     = 216*0.8;          %Usable energy capacity for each vehicle [kWh]
e_w         = 1.5;              %energy usage [kWh/km]
c_w         = 0.1;              %Variable costs of bus [euro/km] (excl energy)

```

```
%Time-table
filename    = 'test_schedule1.xlsx';

time_table  = readtable(filename,'Sheet','Time_Table');
L           = readtable(filename,'Sheet','All_Locations');
w           = xlsread(filename,'d');
h           = xlsread(filename,'t');
D_string    = readtable(filename,'Sheet','Depot');
S_string    = readtable(filename,'Sheet','Charging');
```

Code

Pre-calculations

In the following code the time inputs are converted from fractions of the day to minutes since the start of the day. Furthermore, the number of service trips that are happening simultaneously are calculated, giving the lower bound on the number of buses. In addition to this, the location names are converted to their location number. This is done to ease the referencing and looping over locations later.

```
%Change model time to minutes
h_start = round(h_start*24*60);
h_end   = round(h_end*24*60);
h       = round(h*24*60);

%Calculate number of concurrent service trips
for i = 1:n_t
    begin_time = h_start(i);
    temp = find(begin_time > h_start & h_end >= begin_time);
    n_sim_service_trips(i) = length(temp)+1;
end
min_buses = max(n_sim_service_trips);

%Convert location name of begin and end location to location number
for i = 1:n_t
    begin_loc = l_start_cell{i};
    end_loc   = l_end_cell{i};

    [l_start(i),~]  = find(strcmp(begin_loc,L.Name) == 1);
    [l_end(i),~]  = find(strcmp(end_loc,L.Name) == 1);
end
```

To be able to map the trips between locations as a deadhead trip all the different deadhead arcs are numbered. This can be found below. Here the start of the deadhead arcs is a high number, this number has to be higher than the number of service trips. The start location of each deadhead arc is i, and the end locations is j.

```
n_l                 = height(L);
arc_number          = zeros(n_l,n_l);
deadhead_arc_number = 100001;
for i = 1:n_l
    for j = 1:n_l
        arc_number(i,j) = deadhead_arc_number;
        deadhead_arc_number = deadhead_arc_number+1;
    end
end
```

In the information in the excel file the cost and energy consumption of deadhead trips is not given. In the following code this is calculated.

```
e_deadhead   = zeros(n_l);
c_deadhead   = zeros(n_l);

for i = 1:n_l
    for j = 1:n_l
        if ne(i,j)
            e_deadhead(i,j) = (w(i,j)/1000)*e_w;
            c_deadhead(i,j) = (w(i,j)/1000)*c_w+e_deadhead(i,j)*c_e;
        end
    end
end
```

This model adds the cost of charging, the charging time and the cost of the bus to the outgoing arcs of the charging stations and the depot respectively. In the code below this calculation is given.

```
%Add price of bus and charging time
for i = 1:n_l
    for j = 1:n_l
        if any(ismember(S,i))  %If deadhead trip is from charging location
            h(i,j) = h(i,j)+h_s;
            c_deadhead(i,j) = c_deadhead(i,j)+c_s;
        elseif any(ismember(D,i))       %If deadhead trip is from depot
            c_deadhead(i,j) = c_deadhead(i,j)+c_b;
        end
    end
end
```

Next, the time and time-energy compatibility arrays need to be calculated. This is done in the following code:

```
%% Check compatible trips
comp    = zeros(n_t);
comps   = zeros(n_t,n_t,n_s);

% Determine if trips are time compatible
for i = 1:n_t
    for j = i+1:n_t
        if h_end(i)+h(l_end(i),l_start(j))+h_gap <= h_start(j)
            comp(i,j) = 1;
        end
    end
end

% Determine if trips are energy compatible
for s = 1:n_s
    for i = 1:n_t
        for j = i+1:n_t
            if h_end(i)+h(l_end(i),S(s))+h(S(s),l_start(j))+h_gap <= h_start(j)
                comps(i,j,s) = 1;
            end
        end
    end
end
```

CSA

In this section the different steps of the CSA is given in code. First, the different labels have to be made. A label is a set of possible arc sequences. The labels L_1,...,L_4 correspond to steps 1,...,4 of the CSA. The steps 1 to 4 are iterated for all service trips, the currently assessed service trips is i.

```
%% CSA.1 Creating labels
% generate labels for arc associated with trip. Take union from all
% starting depots and from starting depots via charging stations.

L_1         = cell(1,n_t);
L_2         = cell(1,n_t);
L_3         = cell(1,n_t);
L_3_temp    = cell(1,n_t);
L_3_feas    = cell(1,n_t);
L_4         = cell(1,n_t);
```

CSA step 1 The first step of the CSA is to obtain all the sequences that lead up to and include the current service trip i. For the first trip this is both the direct way from the depots to the start of the service trip and via a charging station. For any trip other than the first trip, the same sequences are added. Tn addition to this the last locations of the already planned buses are taken and sequences from these locations toward the current service trip are made. These are only added if the service trips are *comp* or *comps*.

```
for i = 1:n_t
    progress = i/n_t;
    waitbar((progress*0.8)+0.1)
    %1.a if first trip:a
    if i == 1
        l = 1;
        n = 1;
        for k = 1:n_d

            %Directly from depot to service trip
            %No self loop allowed
            if D(k) ~= l_start(i)
                %Sequence up to trip
                L_1{i}{l,n} = arc_number(D(k),l_start(i));
            end
            l = l+1;

            %From depot to service trip via charging location
            for s = 1:n_s
                if D(k) ~= S(s)                         %No self loop allowed
                    L_1{i}{l,n}     = arc_number(D(k),S(s));
                    L_1{i}{l,n+1}   = arc_number(S(s),l_start(i));
                    l = l+1;
                end
            end
        end

    %1.b if not first trip:b
    else
        L_1{i}{1,1} = [];                %Generate the next step in L_1

        %Add previous solution to new start solution
```

```
        %Time compatible
        [n_rows_cur_sol,~] = size(current_solution);
        for k = 1:n_rows_cur_sol    %For all current sequences

            %Find last arc
            %Check number of steps in current solution sequences
            n_stepsk    = sum(~cellfun(@isempty,current_solution(k,:)),2);
            last_arc    = current_solution{k,n_stepsk};

            %If last trip is service trip (If last trip is not a service trip,
            %then is toward a charger --> comps)
            %Last service trip in current solution
            if any(ismember(T,last_arc))
                previous_service_trip = last_arc;
                %If previous service trip and new service trip are compatible
                if comp(previous_service_trip,i) == 1
                    if sum(~cellfun(@isempty,L_1{i})) == 0
                        n_rows = 0;
                    else
                        [n_rows, ~] = size(L_1{i});
                    end
                    %Number of steps of solution for k
                    n_stepsk    = sum(~cellfun(@isempty,...
                        current_solution(k,:)),2);
                    for n = 1:n_stepsk
                        %Add previous solution to new possible solution
                        L_1{i}(n_rows+1,n) = current_solution(k,n);
                    end
                end
            end
        end

        %Time-energy compatible
        [n_rows_cur_sol,~] = size(current_solution);
        for k = 1:n_rows_cur_sol    %For all current sequences
            temp = true;
            %Check number of steps in current solution sequences
            n_stepsk = sum(~cellfun(@isempty,current_solution(k,:)),2);
            last_arc = current_solution{k,n_stepsk};
            if any(ismember(T,last_arc))
                last_loc = l_end(last_arc);
            else
                [~,last_loc] = find(arc_number == last_arc);
            end

            %If last arc is a service trip
            if any(ismember(T,last_arc))
                for s = 1:n_s
                    %If previous service trip and new service trip are compatible
                    if comps(last_arc,i,s) == 1

                        if sum(~cellfun(@isempty,L_1{i})) == 0
                            n_rows = 0;
                        else
                            [n_rows, ~] = size(L_1{i});
                        end

                        %Add previous solution to new possible solution
                        for n = 1:n_stepsk
                            L_1{i}(n_rows+1,n) = current_solution(k,n);
                        end
                        %Add arc toward charging station
                        L_1{i}{n_rows+1,n_stepsk+1} = arc_number(last_loc,S(s));
```

```
                    end
                end

            %If last arc is not a service trip
            else

                %Check last service trip
                for l = n_stepsk:-1:1
                    arc = current_solution{k,l};
                    %Last service trip in current solution
                    if any(ismember(T,arc)) && temp == true
                        previous_service_trip = arc;
                        temp = false;
                    end
                end

                for s = 1:n_s
                    %If previous service trip and new service trip are compatible
                    if comps(previous_service_trip,i,s) == 1 && last_loc == S(s)
                        if sum(~cellfun(@isempty,L_1{i})) == 0
                            n_rows = 0;
                        else
                            [n_rows, ~] = size(L_1{i});
                        end
                        %Number of steps of solution for k
                        n_stepsk    = sum(~cellfun(@isempty,...
                            current_solution(k,:)),2);
                        for n = 1:n_stepsk
                            %Add previous solution to new possible solution
                            L_1{i}(n_rows+1,n) = current_solution(k,n);
                        end
                    end
                end
            end
        end

    %Add deadhead arc towards start location of service trip i
    if sum(~cellfun(@isempty,L_1{i})) == 0
        n_rows = 0;
    else
        [n_rows, ~] = size(L_1{i});
    end
    n_steps         = sum(~cellfun(@isempty,L_1{i}),2);

    %Determine arc towards service trip
    for k = 1:n_rows
        n = n_steps(k);
        if n == 0
            break
        end

        last_arc = L_1{i}{k,n};
        if any(ismember(T,last_arc))        %If last arc is a service trip
            l_end_prev = l_end(last_arc);
        else                                %If last arc is not a service trip
            [~,l_end_prev] = find(arc_number == last_arc);
        end

        %If begin location of next service trip is not equal to
        %end location of previous arc
        if l_start(i) ~= l_end_prev
            arc = arc_number(l_end_prev,l_start(i));
            L_1{i}{k,n_steps(k)+1} = arc;
```

```
        end
    end

        %Generate labels if a new bus is taken from the depot
        for l = 1:n_d
            if sum(~cellfun(@isempty,L_1{i})) == 0
                n_rows = 0;
            else
                [n_rows, ~] = size(L_1{i});
            end
            %Directly from depot to service trip
            if D(l) ~= l_start(i)                           %No self loop allowed
                %Sequence up to trip
                L_1{i}{n_rows+1,1} = arc_number(D(l),l_start(i));
            end
            %From depot to service trip via charging location
            for s = 1:n_s
                if D(l) ~= S(s)                             %No self loop allowed
                    L_1{i}{n_rows+1+s,1}    = arc_number(D(l),S(s));
                    L_1{i}{n_rows+1+s,2}    = arc_number(S(s),l_start(i));
                end
            end
        end
    end

    %Add service trip to sequence
    [n_rows, ~]     = size(L_1{i});
    n_steps         = sum(~cellfun(@isempty,L_1{i}),2);
    for k = 1:n_rows
        L_1{i}{k,n_steps(k)+1} = i;
    end

    L_2{i} = L_1{i};
```

CSA step 2 The goal of step 2 is to remove sequences that are generated in step 1 that are energy infeasible. To simplify the code, the energy infeasibility calculation is implemented as a function.

```
   %% CSA.2 Remove dominated and energy infeasible labels

    %Remove energy infeasible sequences
    [energy_level_L1, energy_infeasible, energy_used_tot, time_used_tot, ...
        cost_tot, dist_tot] = energy_feasibility(L_2{i},T,S,arc_number,...
    e_deadhead,e_t,c_deadhead,c_t,w,w_t,h,l_start,l_end,h_start,h_end,e_b_max);
    L_2{i}(find(energy_infeasible ~= 0),:) = [];
```

In the next section the energy_feasibility function is given. This function does not only give a statement on which sequence is energy infeasible, but also the cost, energy used and distance driven. This function is used multiple times in the script, it is only explained here. It is important to note that for the energy level, it sets the energy level to full when traveling toward a charging station. It is also checked if the bus can still reach the charging station with the energy remaining.

```
function [energy_level, energy_infeasible, energy_used_tot, ...
    time_used_tot, cost_tot, dist_tot] = energy_feasibility(L,T,...
    S,arc_number,f,f_trip,c,c_trip,d,d_trip,t,b,e,bt,et,w)
%Check if solution is energy feasible
```

```
[n_rows,~]        = size(L);
n_steps           = sum(~cellfun(@isempty,L),2);

energy_used_arc      = zeros(n_rows,max(n_steps));
time_used_arc     = zeros(n_rows,max(n_steps));
dist_arc          = zeros(n_rows,max(n_steps));
cost_arc          = zeros(n_rows,max(n_steps));

energy_level      = zeros(n_rows,max(n_steps));
energy_infeasible = zeros(n_rows,1);
energy_used_tot = zeros(n_rows,1);
time_used_tot     = zeros(n_rows,1);
dist_tot          = zeros(n_rows,1);
cost_tot          = zeros(n_rows,1);

for k = 1:n_rows
    %Check which sequences are not feasible with regard to energy

    for l = 1:n_steps(k)                    %For all possible sequences
        arc                  = L{k,l};      %Check which arc is traversed

        if any(ismember(T,arc))                         %If trip is a service trip
            %energy usage of service trip i
            energy_used_arc(k,l)   = f_trip(arc);
            %Time used to fullfill service trip i
            time_used_arc(k,l)   = et(arc)-bt(arc);
            %Costs associated with service trip i
            cost_arc(k,l)        = c_trip(arc);
            %Driven distance by service trip i
            dist_arc(k,l)        = d_trip(arc);

        else                                %If trip is a deadhead trip
            %Check what are the start and end locations
            [b_loc, e_loc]   = find(arc_number == arc);
            %Check the energy consumption over that arc
            energy_used_arc(k,l)   = f(b_loc,e_loc);
            time_used_arc(k,l)   = t(b_loc,e_loc);   %Check time usage over arc
            cost_arc(k,l)        = c(b_loc,e_loc);   %Check cost of arc
            dist_arc(k,l)        = d(b_loc,e_loc);   %Check distance of arc
        end

        %Energy level of bus on arcs towards service trip
        %If arc is towards charge location, new charge level is w
        if l~= 1 && any(ismember(S,e_loc)) ...
                && energy_used_arc(k,l) < energy_level(k,l-1)
            energy_level(k,l) = w;
            %If it is first arc in sequence, then new charge level is full
            %charge level minus energy used.
        elseif l == 1
            energy_level(k,l) = w-energy_used_arc(k,l);

        else %New charge level is old charge level minus energy used.
            energy_level(k,l) = energy_level(k,l-1)-energy_used_arc(k,l);
        end
    end

    %Determine total
    %Determine total energy used
    energy_used_tot(k)    = sum(energy_used_arc(k,:));
    %Determine total time used
    time_used_tot(k)    = sum(time_used_arc(k,:));
    %Determine total cost
    cost_tot(k)         = sum(cost_arc(k,:));
```

```
    %Determine total distance driven
    dist_tot(k)        = sum(dist_arc(k,:));
end
for k = 1:n_rows
    if any(energy_level(k,:) < 0)
        energy_infeasible(k) = 1;
    end
end
end
```

CSA step 3 In this step, all the arcs are added from the last location toward the different charging locations.

```
  %% CSA. 3 Generate labels associated with visiting each charging station
    %following the service trip
    %disp('Step 3: Generating possible sequences to charge station
    %after service trip')
    n_steps         = sum(~cellfun(@isempty,L_2{i}),2); %Number of arcs traveled
    %Count number of current feasible sequences
    [n_rows, ~]     = size(L_2{i});
    %Pre-define new solution cell-array
    L_3{i}          = cell(n_rows,min(n_steps));

    for k = 1:n_rows
        n       = n_steps(k);   %Number of arcs including service trip

        for s = 1:n_s+1

            if s == 1           %If not towards charging station, only old label
                for l = 1:n
                    L_3{i}{(n_s+1)*(k-1)+s,l} = L_2{i}{k,l};        %Old label
                end
            else                %If towards charging station after service trip
                %Arc number from end location service trip to charge station
                arc = arc_number(l_end(i),S(s-1));
                for l = 1:n+1
                    if l == n+1
                        %Add arc towards charge station
                        L_3{i}{(n_s+1)*(k-1)+s,l} = arc;
                    else
                        L_3{i}{(n_s+1)*(k-1)+s,l} = L_2{i}{k,l};    %Old label
                    end
                end
            end
        end
    end
```

CSA step 4 Step 4 of the CSA is the most comprehensive step. The goal of this step is to check if the bus can still reach its home depot. Removing the infeasible sequences. The next goal is to determine which sequence does add the lowest added costs to the total solution. Then, that solution is chosen and added to the current_solution.

The first step is done by adding the arcs toward the home depot to the end of the sequences of step 3. The sequences that cannot return to their home depot are deleted from both L3 and L4. This way, L3 represents the sequences that can still reach the home depot after the service trip.

Next, the cost of the previous solution is calculated, the cost of the new possible sequences is calculated and it is checked what the added costs are. The lowest added costs is chosen and the new part of that sequence is added to the end of the current solution.

The next step is:

```
%% CSA. 4 Check if buses can still reach home depot. Choose cheapest
    %solution and save this solution.
    % Check if schedules associated with service trip or at the charge station
    % can reach the end depot given remaining energy.

    %----------------------------%
    %Step 4_1: Check if home depot can be reached
    L_4{i}          = L_3{i};

    %Add trip that travels to starting depot of sequence
    n_steps     = sum(~cellfun(@isempty,L_4{i}),2);
    %Count number of current feasible sequences
    [n_rows, ~] = size(L_4{i});

    for k = 1:n_rows
        arc_first= L_4{i}{k,1};
        arc_last = L_4{i}{k,n_steps(k)};

        %Find start location
        if any(ismember(T,arc_first))           %If first trip is a service trip
            b_loc = l_start(arc_first);         %Find start location
        else
            [b_loc, ~] = find(arc_number == arc_first);     %Find start location
        end

        %Find end location
        %If last trip is a service trip
        if any(ismember(T,arc_last))
            e_loc = l_end(i);                               %Find end location
        else
            [~, e_loc] = find(arc_number == arc_last);      %Find end location
        end

        %Add arc to starting depot
        arc_to_depot = arc_number(e_loc,b_loc);
        L_4{i}{k,n_steps(k)+1} = arc_to_depot;
    end

    %Check if labels can reach starting depot with range remaining
    [energy_level_L4, energy_infeasible_L4, energy_used_tot, time_used_tot,...
        cost_tot, dist_tot] = energy_feasibility(L_4{i},T,S,arc_number,...
        e_deadhead,e_t,c_deadhead,c_t,w,w_t,h,l_start,l_end,...
        h_start,h_end,e_b_max);


    %L_4 is array with possible sequences, including arc toward home depot
    %L_3_feas is the same as L_4, but without the arcs toward home depot
    L_4{i}(find(energy_infeasible_L4 == 1),:) = [];
    L_3_temp{i} = L_3{i};
    L_3_temp{i}(find(energy_infeasible_L4 == 1),:) = [];
    L_3_feas{i} = L_3_temp{i};

    %----------------------------%
    %Step 4_2: Determine the best solution for trip i
    %Determine costs of all sequences that are able to perform
    %service trip i (L_3_feas
```

```
    %Determine number of rows
    [n_rows, ~]              = size(L_3_feas{i});
    %Determine number of traversed arcs for each row
    n_steps                  = sum(~cellfun(@isempty,L_3_feas{i}),2);
    cost_total_L3_feas       = zeros(n_rows,1);
    cost_arc_L3_feas         = zeros(n_rows,max(n_steps));
    added_costs              = zeros(n_rows,1);
    clear loc_prev_trip

    %Determine costs of L3_feas possible sequences
    for k = 1:n_rows
        for l = 1:n_steps(k)

            arc = L_3_feas{i}{k,l};
            if any(ismember(T,arc))
                cost_arc_L3_feas(k,l) = c_t(arc);
            else
                [b_loc,e_loc] = find(arc_number == arc);
                cost_arc_L3_feas(k,l) = c_deadhead(b_loc,e_loc);
            end
        end
        cost_total_L3_feas(k) = sum(cost_arc_L3_feas(k,:));

        cost_L3_feas_save{i}        = cost_arc_L3_feas;
        cost_total_L3_feas_save{i}  = cost_total_L3_feas;
    end

    %Find location minimal added costs of performing service trip i
    if i == 1
        added_costs = cost_total_L3_feas;
    else

        %Determine the cost of the sequences up to the first
        %deadhead arc to perform service trip i

        clear n_steps
        %Determine number of rows
        [n_rows, ~]                  = size(L_3_feas{i});
        %Determine number of traversed arcs for each row
        n_steps                      = sum(~cellfun(@isempty,L_3_feas{i}),2);
        cost_total_L3_feas_prev      = zeros(n_rows,1);
        cost_arc_L3_feas_prev        = zeros(n_rows,max(n_steps));

        for k = 1:n_rows
            %If the possible sequence has an earlier service trip, before
            %traveling to the current service trip
            if any(ismember(setdiff(T,i),[L_3_feas{i}{k,1:n_steps(k)}]))
                for l = n_steps(k):-1:1
                    arc = L_3_feas{i}{k,l};
                    if arc < i
                        %Save the step number which is the previous service trip.
                        loc_prev_trip(k) = l;
                        break
                    end
                end

                for l = 1:loc_prev_trip(k)
                    arc = L_3_feas{i}{k,l};
                    if any(ismember(T,arc))
                        cost_arc_L3_feas_prev(k,l) = c_t(arc);
                    else
                        [b_loc,e_loc] = find(arc_number == arc);
                        cost_arc_L3_feas_prev(k,l) = c_deadhead(b_loc,e_loc);
```

```
                    end
                end
                cost_total_L3_feas_prev(k) = sum(cost_arc_L3_feas_prev(k,:));
                added_costs(k) = cost_total_L3_feas(k)-...
                    cost_total_L3_feas_prev(k);

                cost_total_L3_feas_prev_save{i} = cost_arc_L3_feas_prev;
            else
                %If the possible sequence does not have an earlier service trip
                added_costs(k) = cost_total_L3_feas(k);
            end
        end
    end

    added_costs_save{i} = added_costs;

    [loc_seq_best,~]     = find(added_costs == min(added_costs));
    [n_sol, ~]           = size(loc_seq_best);
    if n_sol > 1
        loc_seq_best     = loc_seq_best(1,1);
    end
    loc_seq_best_save{i} = loc_seq_best;
    %Determine number of traversed arcs for each row
    n_steps              = sum(~cellfun(@isempty,L_3_feas{i}),2);

    for n = 1:n_steps(loc_seq_best)
        best_seq_sol{i,n} = L_3_feas{i}{loc_seq_best,n};
    end
    clear added_costs loc_last_service_trip

    %---------------------------%
    %Step 4_3: Save best sequence to the current_solution
    %Check if beginning of best sequence is already part of another sequence,
    %if so: Add solution to current_solution

    %If trip is the first service trip
    if i == 1
        for n = 1:n_steps(loc_seq_best)
            current_solution{i,n} = best_seq_sol{i,n};
        end

    %If trip is not the first service trip
    else
        after_service_trip = false;

        %Check if part of new best solution is already an earlier trip
        [~, n_steps] = size(best_seq_sol(i,:));
        for n = 1:n_steps
            arc = best_seq_sol{i,n};
            if any(ismember(T,arc)) && arc ~= i
                %The new best solution is done after an earlier service trip
                after_service_trip = true;
            end
        end

        %If new solution is after an earlier service trip
        if after_service_trip == true
            n_steps = sum(~cellfun(@isempty,best_seq_sol(i,:)),2);
            %Check from which arc the new arcs start
            for n = n_steps:-1:1
                if any(ismember(T,best_seq_sol{i,n})) && best_seq_sol{i,n} ~= i
                    loc_last_service_trip = n;
                end
```

```
            end

            %Save the previous solution that has to be found in current_solution
            previous_trip = best_seq_sol(i,1:loc_last_service_trip);

            %Find location of previous trips in current_solution
            [n_rows,~]      = size(current_solution);
            n_stepsk        = sum(~cellfun(@isempty,previous_trip),2);
            for k = 1:n_rows
                check_same = zeros(1,n_stepsk);
                for n = 1:n_stepsk
                    if current_solution{k,n} == previous_trip{1,n}
                        check_same(1,n) = 1;
                    else
                        check_same(1,n) = 0;
                    end
                end

                if all(check_same)  %If all values in check_same are 1
                    location_sol = k;
                    [~, n_steps_sol] = size(best_seq_sol(i,:));
                    current_solution(location_sol,1:n_steps_sol)...
                        = best_seq_sol(i,:);
                    break
                end
            end
        else                        %If new solution takes a new bus
            [n_rows,~]      = size(current_solution);
            %Determine number of traversed arcs for each row
            n_steps         = sum(~cellfun(@isempty,best_seq_sol(i,:)));
            for n = 1:n_steps
                current_solution{n_rows+1,n} = best_seq_sol{i,n};
            end
        end
    end
```

Finishing code

Here the arcs are added from the last location of the bus towards the home depot of that bus, since it was checked earlier that this was energy feasible this is not checked again here.

```
%Add deadhead arcs toward home station after all the service trips
[n_rows,~]  = size(current_solution);
n_steps     = sum(~cellfun(@isempty,current_solution),2);
for k = 1:n_rows
    n = n_steps(k);
    last_arc = current_solution{k,n};
    first_arc = current_solution{k,1};

    if any(ismember(T,first_arc))
        b_loc = l_start(first_arc);
    else
        [b_loc, ~] = find(arc_number == first_arc);
    end

    if any(ismember(T,last_arc))
        e_loc = l_end(last_arc);
    else
        [~,e_loc] = find(arc_number == last_arc);
    end
```

```

    %Add deadhead arc towards home depot to solution
    current_solution{k,n+1} = arc_number(e_loc,b_loc);

end
```

Checking solution

It is important to know that the solution that is calculated is also feasible. In terms of energy but also in terms of time and location. These checks are performed in the code below.

```
%Check if solution is energy feasible
[energy_level_solution, energy_infeasible_solution, energy_used_tot_solution,...
    time_used_tot_solution, cost_tot_solution, dist_tot_solution] = ...
    energy_feasibility(current_solution,T,S,arc_number,e_deadhead,e_t,...
    c_deadhead,c_t,w,w_t,h,l_start,l_end,h_start,h_end,e_b_max);

if any(energy_infeasible_solution)
    disp('ERROR: Solution is energy infeasible')
else
    disp('- Solution is energy feasible')
end

%Check if solution completes all service trips a single time
total_occurrences = zeros(n_t,1);
for i = 1:n_t
    % Find OccurrencesIn Each Cell
    nr_in_cell = cellfun(@(x) find(x==i), current_solution, 'Uni',0);
    % Output Total Occurrences In All Cells
    total_occurrences(i) = numel([nr_in_cell{:}]);
end
if all(total_occurrences)
    disp('- Every service trip is performed exactly once')
end

%Check if bus ends at the depot where it started
[n_rows,~]  = size(current_solution);
n_steps     = sum(~cellfun(@isempty,current_solution),2);
for k = 1:n_rows
    n = n_steps(k);
    last_arc = current_solution{k,n};
    first_arc = current_solution{k,1};
    if any(ismember(T,first_arc))
        b_loc = l_start(first_arc);
    else
        [b_loc, ~] = find(arc_number == first_arc);
    end

    if any(ismember(T,last_arc))
        e_loc = l_end(last_arc);
    else
        [~,e_loc] = find(arc_number == last_arc);
    end

    if e_loc == b_loc
        home_depot(k) = 1;
    end
end

if any(home_depot) %if all values in home_depot are one
```

```
    disp('- All buses return to their start depot')
else
    disp('- ERROR: Not all buses return to their start depot')
end


%Check if every next trip starts from the end location of the previous trip
startendcheck_save      = cell(1,n_t);
startendcheck_array     = zeros(1,n_t);
for k = 1:n_rows
    startendloc         = zeros(n_steps(k),2);
    startendcheck_temp  = zeros(n_steps(k)-1,1);

    for n = 1:n_steps(k)
        %Check which arc is traversed
        arc                 = current_solution{k,n};
        if any(ismember(T,arc))                 %If trip is a service trip
            b_loc           = l_start(arc);     %Begin location of trip
            e_loc           = l_end(arc);       %End location of trip
        else
            [b_loc,e_loc]   = find(arc_number == arc);
        end

        startendloc(n,1) = b_loc;
        startendloc(n,2) = e_loc;
        if n ~= 1
            %If previous end location is the same as the current start location
            if startendloc(n-1,2) == startendloc(n,1)
                startendcheck_temp(n-1,1) = 1;
            end

        end
    end
    startendcheck_save{k} = startendcheck_temp;

    if any(startendcheck_save{k})
        startendcheck_array(k) = 1;
    end
end

if any(startendcheck_array)
    disp('- All start/end locations are correct')
else
    disp('- ERROR: Buses move between locations without deadhead arc')
end
```

Output

Numbers

```
disp('=======================================')
disp(' ')
disp('Results:')

%Determine total CPU time
computation_time = toc;
A = ['- Computation time:         ',num2str(computation_time),' [s]'];
disp(A)

%Determine total costs
total_cost_solution = round(sum(cost_tot_solution));
total_cost_disp = addComma(total_cost_solution);
A = ['- Total cost:               ',char(8364),total_cost_disp,'.-'];
disp(A)

%Determine number of buses used
[n_buses,~] = size(current_solution);
A = ['- Number of buses used:     ', num2str(n_buses)];
disp(A)

%Determine number of extra buses used due to deadhead trips and charging
% A = ['- Extra buses due to deadhead+charging:   ', num2str(n_buses-min_buses)];
% disp(A)

%Display Bus Assignment
% disp(' ')
% disp('Bus Assignment :')
Bus_assignment = current_solution;
% disp(Bus_assignment)

%Display energy levels
% disp('energy level after each movement')
% disp(energy_level_solution)
total_energy_used = sum(energy_used_tot_solution);
A = ['- Total energy used:   ', num2str(round(total_energy_used)), ' [kWh]'];
disp(A)

%Display kilometers driven
%Per bus
%Total
total_distance_driven = sum(dist_tot_solution);
A = ['- Total distance driven:  ', num2str(round(total_distance_driven/1000)),...
    ' [km]'];
disp(A)


%Display number of charging sessions
to_charger                   = zeros(n_rows,max(n_steps));
n_charging_sessions_bus      = zeros(n_rows,1);
n_charging_sessions_charger = zeros(n_s,1);
for k = 1:n_rows
    for n = 1:n_steps(k)
        arc_towards = Bus_assignment{k,n};
        [~,e_loc] = find(arc_number == arc_towards);
        if any(ismember(S,e_loc))
            [~,charger_number] = find(S == e_loc);
            n_charging_sessions_charger(charger_number) = ...
```

```
                n_charging_sessions_charger(charger_number)+1;
            to_charger(k,n) = 1;
        end
    end
    n_charging_sessions_bus(k) = sum(to_charger(k,:));
end
%Per bus

%Total
total_number_charging_sessions = sum(n_charging_sessions_bus);
A = ['- Total number of charging sessions:      ',...
    num2str(total_number_charging_sessions)];
disp(A)
```

Figures

```
%% Figures
%Make Gantt chart
[n_rows,~]  = size(Bus_assignment);
n_steps     = sum(~cellfun(@isempty,Bus_assignment),2);
service_trip_times  = zeros(n_t,4);
first_service_trip  = zeros(n_rows,1);
deadhead_times      = zeros(1,1,n_rows);
n_deadhead          = zeros(n_rows,1);
charging_times      = zeros(1,1,1);
n_charging          = zeros(n_rows,1);


%Determine start and end times of different types of trips in sequence

%Service trips
for k = 1:n_rows
    temp = true;
    for n = 1:n_steps(k)
        trip_number = Bus_assignment{k,n};
        if any(ismember(T,trip_number)) %If trip is a service trip
            service_trip_times(trip_number,1) = trip_number;
            service_trip_times(trip_number,2) = k;
            service_trip_times(trip_number,3) = h_start(trip_number);
            service_trip_times(trip_number,4) = h_end(trip_number);

            if temp == true
                first_service_trip(k) = n;
                temp = false;
            end
        end
    end
end

%Deadhead + charging trips
for k = 1:n_rows

    for n = first_service_trip(k)-1:-1:1
        trip_number     = Bus_assignment{k,n};
        if any(ismember(T,trip_number)) %If current trip is a service trip
        else                            %If current trip is a deadhead trip
            [b_loc,e_loc]   = find(arc_number == trip_number);
            next_trip       = Bus_assignment{k,n+1};
            n_deadhead(k)   = n_deadhead(k)+1;

```

```
            if any(ismember(S,b_loc))   %If from charging location
                n_charging(k) = n_charging(k)+1;

                if any(ismember(T,next_trip))       %Next trip is a service trip
                    deadhead_times(n_deadhead(k),1,k) = trip_number;
                    deadhead_times(n_deadhead(k),2,k) = k;
                    deadhead_times(n_deadhead(k),3,k) = ...
                        h_start(next_trip)-h(b_loc,e_loc);
                    deadhead_times(n_deadhead(k),4,k) = h_start(next_trip);

                    charging_times(n_charging(k),1,k) = trip_number;
                    charging_times(n_charging(k),2,k) = k;
                    charging_times(n_charging(k),3,k) = ...
                        h_start(next_trip)-h(b_loc,e_loc);
                    charging_times(n_charging(k),4,k) = ...
                        h_start(next_trip)-h(b_loc,e_loc)+h_s;

                else                            %Next trip is not a service trip
                end

            else                        %If not from charging location
                if any(ismember(T,next_trip))   %Next trip is a service trip
                    deadhead_times(n_deadhead(k),1,k) = trip_number;
                    deadhead_times(n_deadhead(k),2,k) = k;
                    deadhead_times(n_deadhead(k),3,k) = ...
                        h_start(next_trip)-h(b_loc,e_loc);
                    deadhead_times(n_deadhead(k),4,k) = h_start(next_trip);
                else                    %Next trip is not a service trip
                    deadhead_times(n_deadhead(k),1,k) = trip_number;
                    deadhead_times(n_deadhead(k),2,k) = k;
                    deadhead_times(n_deadhead(k),3,k) = ...
                        deadhead_times(n_deadhead(k)-1,3,k)-h(b_loc,e_loc);
                    deadhead_times(n_deadhead(k),4,k) = ...
                        deadhead_times(n_deadhead(k)-1,3,k);

                end
            end
        end
    end

    for n = first_service_trip(k)+1:n_steps(k)
        trip_number     = Bus_assignment{k,n};

        if any(ismember(T,trip_number)) %If current trip is a service trip
        else                            %If current trip is a deadhead trip
            [b_loc,e_loc]   = find(arc_number == trip_number);
            previous_trip   = Bus_assignment{k,n-1};
            n_deadhead(k)   = n_deadhead(k)+1;

            if any(ismember(S,b_loc))   %If from charging location
                n_charging(k) = n_charging(k)+1;

                %If previous trip is a service trip
                if any(ismember(T,previous_trip))
                    %Never occurs
                else                    %If previous trip is not a service trip
                    deadhead_times(n_deadhead(k),1,k) = trip_number;
                    deadhead_times(n_deadhead(k),2,k) = k;
                    deadhead_times(n_deadhead(k),3,k) = ...
                        deadhead_times(n_deadhead(k)-1,4,k)+h_s;
                    deadhead_times(n_deadhead(k),4,k) = ...
                        deadhead_times(n_deadhead(k)-1,4,k)+h(b_loc,e_loc);

```

```
                    charging_times(n_charging(k),1,k) = trip_number;
                    charging_times(n_charging(k),2,k) = k;
                    charging_times(n_charging(k),3,k) = ...
                        deadhead_times(n_deadhead(k)-1,4,k);
                    charging_times(n_charging(k),4,k) = ...
                        deadhead_times(n_deadhead(k)-1,4,k)+h_s;
                end

            else                        %If not from charging location
                %If previous trip is a service trip
                if any(ismember(T,previous_trip))
                    deadhead_times(n_deadhead(k),1,k) = trip_number;
                    deadhead_times(n_deadhead(k),2,k) = k;
                    deadhead_times(n_deadhead(k),3,k) = ...
                        h_end(previous_trip);
                    deadhead_times(n_deadhead(k),4,k) = ...
                        h_end(previous_trip)+h(b_loc,e_loc);
                else                    %If previous trip is not a service trip
                    deadhead_times(n_deadhead(k),1,k) = trip_number;
                    deadhead_times(n_deadhead(k),2,k) = k;
                    deadhead_times(n_deadhead(k),3,k) = ...
                        deadhead_times(n_deadhead(k)-1,4,k);
                    deadhead_times(n_deadhead(k),4,k) = ...
                        deadhead_times(n_deadhead(k)-1,4,k)+h(b_loc,e_loc);
                end
            end
        end
    end
end

%Plotting
figure
hold on
grid on
xmax = max([max(max(max(deadhead_times(:,3:end,:))))/60+0.1,...
    max(max(max(service_trip_times(:,3:end,:))))/60+0.1,...
    max(max(max(charging_times(:,3:end,:))))/60+0.1]);
xmin = min(nonzeros(deadhead_times(:,3:end,:)))/60-0.1;
axis([xmin xmax 0 n_rows+1])
xlabel('Time [h]')
ylabel('Vehicle number [-]')

for k = 1:n_rows        %Deadhead trips
    for n = 1:n_deadhead(k)
        deadhead_trip_plot = plot([deadhead_times(n,3,k)/60 ...
            deadhead_times(n,4,k)/60],[deadhead_times(n,2,k) ...
            deadhead_times(n,2,k)],'-','Color',[1 0.7 0],'LineWidth',3);
    end
end

for k = 1:n_rows        %Charging
    for n = 1:n_charging(k)
        charging_plot = plot([charging_times(n,3,k)/60 ...
            charging_times(n,4,k)/60],[charging_times(n,2,k)...
            charging_times(n,2,k)],'-','Color',[1 0.5 0.5],'LineWidth',5);
    end
end

for k = 1:n_t       %Service trips
    service_trip_plot = plot([service_trip_times(k,3)/60 ...
        service_trip_times(k,4)/60],[service_trip_times(k,2) ...
        service_trip_times(k,2)],'-','Color',[0 0.75 0.75],'LineWidth',10);
    text(service_trip_times(k,3)/60,service_trip_times(k,2),...
```

```
        num2str(k),'FontSize',8);
end


xticks([floor(xmin):1:round(xmax)])
yticks([0:1:n_buses+1])
legend([service_trip_plot,deadhead_trip_plot,charging_plot],...
    'Service Trips','Deadhead Trips','Charging','Location','northwest')
```

Most important variables

arc_number The variable arc_number stands for the deadhead arc number. Here all the locations, service trip start and end location, depots and charging locations are interconnected with deadhead arcs. Each arc has a unique number, the two arcs between two locations therefore have a different number. The direction is from the first column to the other column.

current_solution In this cell-array the current solution is stored. Here each row stands for a bus, and the arcs in that row stands for the sequence that that bus performs. Here are both the deadhead arcs and service trips present. One can see that charging occurs when two deadhead arcs follow each-other. When the computation of each service trip scheduling is completed this solution is added to the current_solution. Therefore, if the calculation is stopped earlier than the total amount of trips the current_solution stores the solution up to that point.

L_x This cell array stores the labels of step x of the CSA. Here it is important to note that each row of arc sequences stand for a possible way to perform service trip i. From all these possible sequences one is chosen by the CSA and added to the current_solution. For each service trip the possible arc sequences are saved. $L\{i\}\{k,n\}$, i is the service trip, k the row number and n the step number.

B.2 Direct ILP

In this section, the code that is used in section 5.2 is given and explained. First, the input parameters are given, in this case, the example timetable. Next, the compatibility array is determined. Furthermore, the number of incompatible trips combinations are calculated.

```
%Script to determine a schedule for a timetable with only electric buses.
%Technique used: Direct ILP
%Filename: Direct_ILP.m
%JWM Wijnheijmer - November 2019 - VDL-ETS
clc; close all;clear all

%% Input
T       = [1,2,3,4,5];      %Set of service trips
n_t     = size(T,2);        %Count number of service trips
h_start = [1,2,3,4,5];      %Begin times of service trips
h_end   = [3,4,5,6,7];      %End times of service trips
c_b     = 1;                %Price of a bus
B       = T;                %Set of buses is equal to set of service trips
n_b_min = size(B,2);        %Determining lower bound on number of buses

%Set options for integer solver
options_intlinprog = optimoptions('intlinprog','Display','none');

%% Process inputs

%Determining comp array
comp = zeros(n_t);
for i = 1:n_t
    for j = i+1:n_t
        if h_start(j) >= h_end(i)
            comp(i,j) = 1;
        end
    end
end

%Determining number of incompatible trip combinations
for i = 1:n_t
    n_incomp(i) = sum(find(comp(i,i+1:n_t) == 0));
end
n_incomp_trips = sum(n_incomp);
```

Next, the arrays are constructed that are used by the integer solver. Firstly, the constraint is added that ensures that no incompatible trips are performed by a bus. Secondly, the constraint that is necessary for the objective function is added. Lastly, the equality constraint that each service trip needs to be performed once is added. Next, the arrays are combined and the lower and upper bounds on the decision variables are added.

```
%% Set up optimization problem
n_decision_var = n_t*n_b_min+1;      %Number of decision variables

%Objective function
f = zeros(n_decision_var,1);
f(n_decision_var) = c_b;

%%INEQUALITY CONSTRAINTS
%No incompatible trips together
A1 = zeros(n_incomp_trips*n_b_min,n_decision_var);
```

```
m = 1;
for b = 0:n_b_min-1
    for i = 1:n_t
        for j = i+1:n_t
            if comp(i,j) == 0
                A1(m,b*n_t+i) = 1;
                A1(m,b*n_t+j) = 1;
                m = m+1;
            end
        end
    end
end
b1 = ones(n_incomp_trips*n_b_min,1);

%Minimize number of used buses
A2 = zeros(n_t*n_b_min,n_t*n_b_min+1);
m = 1;
for b = 0:n_b_min-1
    for t = 1:n_t
        A2(m,b*n_t+t) = b+1;
        A2(m,n_decision_var) = -1;
        m = m+1;
    end
end
b2 = zeros(n_t*n_b_min,1);

%%EQUALITY CONSTRAINTS
%Each trip performed once
A3 = zeros(n_t,n_decision_var);
for t = 1:n_t
    for b = 0:n_b_min-1
        A3(t,b*n_t+t) = 1;
    end
end
b3  = ones(n_t,1);

A   = [A1;A2];
b   = [b1;b2];
Aeq = A3;
beq = b3;
lb  = zeros(n_decision_var,1);
ub  = ones(n_decision_var,1);
ub(n_decision_var) = Inf;
```

In the end, the optimization problem is solved and the results are displayed.

```
%% Solving optimization problem
[solution,Obj_val,~,~] = intlinprog(f,1:n_b_min*n_t,A,b,Aeq,beq,...
    lb,ub,options_intlinprog);

%% Display results
solution        = round(solution)
n_buses_used    = solution(end)
total_cost      = Obj_val

tasks_performed = zeros(n_t,n_b_min);
for b = 1:n_b_min
    tasks_performed(1:n_t,b) = solution((b-1)*n_t+1:(b-1)*n_t+n_t);
end

tasks_performed = round(tasks_performed)
```

B.3 Documentation of column generation

B.3.1 Column generation without energy usage and charging

In this section, the MATLAB code is given and explained that is used to solve the example problem in section 5.3. This script can be used to solve the problem using three methods. The first method is solving the MP. In that case, the set with all the available vehicle paths P needs to be given. The second method is to use the identity matrix as an initial solution and add vehicle tasks iteratively using column generation. The third option is to use column generation for a given initial set of vehicle tasks.

First, the input parameters are given. In this case, the timetable that has to be scheduled. Then, the initial set of columns has to be given.

```
%Script to determine a schedule for a timetable with only electric buses.
%Technique used: Column generation
%Filename: CG_basic.m
%JWM Wijnheijmer - November 2019 - VDL-ETS
clc; clear all; close all

%% Parameters

%Set used method
method = 2;      %1 if all vehicle tasks are known
                 %2 to use identity matrix as initial set of vehicle tasks
                 %3 to define initial set of vehicle tasks manually

%Service trips
T                = [1 2 3 4 5];      %Set of service trips
h_start          = [1 2 3 4 5];      %Begin times of service trips
h_end            = [3 4 5 6 7];      %End times of service trips
n_t              = size(T,2);        %Number of service trips
c_b              = 1;                %Price of a bus
maxiter          = 10;               %Maximum number of iterations
M                = 100;

if method == 1
    %(1) Give all existing vehicle tasks
    V   = [1 0 0 0 0 1 1 1 1 0 0 0;
           0 1 0 0 0 0 0 0 0 1 1 0;
           0 0 1 0 0 1 0 0 1 0 0 1;
           0 0 0 1 0 0 1 0 0 1 0 0;
           0 0 0 0 1 0 0 1 1 0 1 1];
    Xzp     = V;
    V_star  = V;
    C_v     = c_b*ones(1,size(V_star,2));

elseif method == 2
    %(2) Use identity matrix as initial solution
    V_star  = eye(n_t);
    C_v     = c_b*ones(1,n_t);              %Cost of vehicle tasks

else
    %(3) State initial set of vehicle tasks
    V_man    = [1 0 0;
                0 1 0;
                1 0 1;
                0 1 0;
                0 0 1];
```

```

    C_v     = c_b*ones(1,size(V_man,2));    %Cost of vehicle tasks
    V_star  = V_man;
end

%Set options for solvers
options_intlinprog  = optimoptions('intlinprog','Display','none');
options_linprog     = optimoptions('linprog','Display','none');
```

The next step is to find the array *comp*.

```
%% Process inputs
% Determine if trips are time compatible
comp    = zeros(n_t);
for i = 1:n_t
    for j = i+1:n_t
        if h_end(i) <= h_start(j)
            comp(i,j) = 1;
        end
    end
end
```

The next step is to perform the column generation algorithm. This algorithm is repeated until the reduced costs are no longer non-negative, or the maximum number of iterations has been reached.

First, the arrays that state the constraints of the subproblem are formulated. In this example the only constraint in the subproblem is that no service trips that are incompatible are allowed in the same vehicle task. Next the shadow prices are determined. Please note that this script does not use the dual of the RMP, but obtains the shadow prices directly from the RMP. Next, the subproblem is solved using the newly found shadow prices, and the reduced cost is calculated. If the reduced costs is negative, the new vehicle task is added to V' and the algorithm is started again.

```
%% Iteravely find new columns
%Make constraint that ensures no time incompatible trips will be in vehicle task
%Transform comp matrix to constraint matrix
A1_sub = zeros(1,n_t);
k = 0;
for i = 1:n_t
    for j = i+1:n_t
        if comp(i,j) == 0
            if k == 0
                n_rows_A_sub = 0;
                A1_sub(1,i) = 1;
                A1_sub(1,j) = 1;
            else
                n_rows_A_sub = size(A1_sub,1);
                A1_sub(n_rows_A_sub+1,i) = 1;
                A1_sub(n_rows_A_sub+1,j) = 1;
            end
            k = k+1;
        end
    end
end
b1_sub = ones(size(A1_sub,1),1);
clear k

```

```
reduced_costs_save = zeros(maxiter,1);

for i = 1:maxiter

    disp('----------------------------------------')
    disp_txt = ['Iteration number: ', num2str(i)];
    disp(disp_txt)
    disp(' ')
    n_v = size(V_star,2);   %Number of vehicle tasks

    %% RMP: Find shadow prices
    %In this example, the shadow prices are obtained directly from the RMP
    %when this is solved using the simplex method. You can also define the dual
    %of the RMP, and solve that. The resulting decision variables are then the
    %shadow prices.

    f_RMP   = C_v;
    A_RMP   = -V_star(1:end,:);
    b_RMP   = -ones(n_t,1);
    Aeq_RMP = [];
    beq_RMP = [];
    lb_RMP  = zeros(n_v,1);
    ub_RMP  = ones(n_v,1);

    [u_v, Obj_val, exitflag, ~, shadows_prices] = linprog(f_RMP,A_RMP,b_RMP,...
        Aeq_RMP,beq_RMP,lb_RMP,ub_RMP,options_linprog);

    disp_txt = ['Solution RMP'];
    disp(disp_txt)
    disp(u_v)

    %Get shadow prices
    pi_tau              = shadows_prices.ineqlin(1:n_t);
    pi_t_tau_save{i}    = pi_tau';

    disp_txt = ['Shadow prices'];
    disp(disp_txt)
    disp(pi_tau)

    %% SUBPROBLEM: Make new vehicle task

    f_sub   = [-pi_tau'];
    A_sub   = A1_sub;
    b_sub   = b1_sub;
    Aeq_sub = [];
    beq_sub = [];
    lb_sub  = zeros(size(f_sub,2),1);
    ub_sub  = ones(size(f_sub,2),1);

    [new_column, fval, exitflag, output] = intlinprog(f_sub,1:size(f_sub,2),...
        A_sub,b_sub,Aeq_sub,beq_sub,lb_sub,ub_sub,options_intlinprog);
    new_column(1:n_t) = round(new_column(1:n_t));
    new_column_save{i} = new_column;
    fval_sub_save(i) = fval;

    disp_txt = ['New column'];
    disp(disp_txt)
    disp(new_column)

    %Compute reduced costs
    disp_txt = ['Reduced costs'];
    disp(disp_txt)
    reduced_costs = c_b+fval;
```

```
    disp(reduced_costs)
    reduced_costs_save(i) = -reduced_costs;

    %If new column will improve the RMP, add the column to V_star.
    %Otherwise, stop iterating
    if i ~= maxiter && reduced_costs < 0
        V_star  = [V_star,   new_column];
        C_v     = [C_v,  c_b];

        disp_txt = ...
            ['Reduced costs are negative, new column is added to known columns'];
        disp(disp_txt)
        disp(' ')

        disp_txt = ['Known set of vehicle tasks'];
        disp(disp_txt)
        disp(V_star)


    else
        disp_txt = ['Reduced costs are non-negative'];
        disp(disp_txt)
        disp_txt = ['No column can be found that will improve the basis'];
        disp(disp_txt)
        break
    end
end
```

If the reduced cost is non-negative, the MP is solved for the set V' and the results are displayed.

```
%% Find final solution
disp('============================================')
disp_txt = ['Start with finding final solution from basis'];
disp(disp_txt)

f_end   = C_v;
A_end   = -V_star(1:n_t,:);      %Each service trip is performed at least once
b_end   = -ones(n_t,1);
Aeq_end = [];
beq_end = [];
lb_end  = zeros(n_v,1);
ub_end  = ones(n_v,1);
[u_v, total_costs, ~, ~] = intlinprog(f_end,1:n_v,A_end,b_end,Aeq_end,...
    beq_end,lb_end,ub_end,options_intlinprog);

disp_txt = ['Vehicle tasks used in solution'];
disp(disp_txt)
tasks_used = find(u_v == 1);
disp(tasks_used)
disp_txt = ['Cost of solution'];
disp(disp_txt)
disp(total_costs)
```

B.3.2 Column generation with energy usage and charging

The first step is to import the timetable used and the settings that are needed to perform the column generation algorithm.

```
%Script to determine a schedule for a timetable with only electric buses.
%Technique used: Column Generation
%Filename: CG.m
%J.W.M. Wijnheijmer - november 2019 - VDL-ETS
clear all;close all;clc
tic
%% Input

%Time-table
test_schedule       = ['1'];
filename            = ['test_schedule',test_schedule,'_column.xlsx'];

MP_integer_end      = 0;    %(1) Inlinprog should be used to solve
                            %to integer number of vehicle tasks
                            %(0) Linprog in combination with rounding
                            %is used to solve to integers
use_eq_RMP          = 0;    %(1) Use equality constraint in RMP to give
                            %shadow prices.
                            %(0) Use inequality constraint in RMP to give
                            %shadow prices
n_s                 = 1;    %Number of chargers [-]

use_intlin_RMP      = 0;    %(1) Also use intlinprog in RMP, to find integer
                            %optimal objective value of RMP
                            %(0) Only use linprog in RMP, to find current
                            %objective value

%Options ultimate solution
save_results        = 1;    %(1) Save figures and workspace variables
                            %(0) Do not save figures and workspace variables
one_bus_per_trip    = 0;    %(1) Alters the end MP/RMP demands that each trip
                            %is performed exactly once
                            %(0) Each trip has to be performed at least once
%Set stop criteria
max_computation_time    = 7200; %Maximum computation time [s]
maxiter                 = 400;  %Maximum number of iterations [-]
n_min_improvement       = 200;  %Number iterations that is looked back to
min_improvement         = 0.99; %Fraction from which the improvement is to low

%Figure properties
fontsize_figure         = 14;
linewidth_figure        = 2;

%If connect charging sessions between service trips to make them
%continuous (1 is on, 0 is off)
connect_charging_blocks = 1;

%Give solver options
options_linprog     = optimoptions('linprog','Display','none');
options_intlinprog  = optimoptions('intlinprog','Display','none');

maxtime_first_iterations    = 60;
maxtime_later_iterations    = 30;
n_iterations_switch         = 10;

h_gap                       = 1;        %Minimal gap between trips [min]
```

```

%%COLUMN GENERATION
n_z                           = 50;       %Number of time-steps[-]
min_improv_reduc_costs        = -1e-3;    %Value for reduced costs from where a new
                                          %Column will not be added any more [euro]
M                             = 1e3;      %Large M
error_integer                 = 0.0001;   %Error that can be on integer, while
                                          %still considering them as integers [-]
delete_initial_vehicle_tasks = 0;         %If the initial vehicle tasks will be
                                          %deleted from set of vehicle tasks [-]
extra_columns_after_rounding = 0;         %Number of extra columns that are
                                          %generated to ensure that obtaining an
                                          %integer solution [-]
value_rounding_no_extra       = 0.99;     %If vp is higher than given value, no
                                          %extra columns will be added [-]

%%BUS
%Maximum charging power bus (In optimal condition)
P_b_max               = 230;              %[kW]
P_b_min               = 10;               %Minimum charging power bus[kW]
min_SoC               = 0.10;             %Minimum SoC [%]
max_SoC               = 0.90;             %Maximum SoC [%]
%Maximum energy capacity of the battery pack [kWh]
battery_capacity      = 216;
e_b_min               = battery_capacity*min_SoC;
e_b_max               = battery_capacity*max_SoC;
c_b                   = 500000/(15*300);  %Unit price of the considered bus type,
                                          %costs per day of operation (fixed costs)
e_w                   = 1.5;              %Energy usage [kWh/km]
c_w                   = 0.1;              %Variable costs of bus [euro/km]

%%CHARGING
max_charge_power_charger      = P_b_max;  %Maximum power of charger [kW]
max_power_charge_loc          = 1000;     %[kW]
max_chargers_connected_bus    = 1;        %Maximum amount of chargers a bus can
                                          %be connected to
c_e                           = 0.2;      %Price of energy in low energy price
                                          %time [euro/kWh]
c_e_high                      = 0.2;      %Price of energy of high energy prive
                                          %time [euro/kWh]
price_high_start              = 11*60;    %Start time of high energy price [min]
price_high_end                = 15*60;    %End time of high energy price [min]
price_energy                  = c_e*ones(n_z,1);
```

The next step is to read the data and to process the inputs.

```
%% Process inputs
time_table        = readtable(filename,'Sheet','TimeTable');
locations_all     = readtable(filename,'Sheet','All_Locations');
h                 = xlsread(filename,'t');

l_start_cell      = time_table.From;      %Begin location, stored as cell array
h_start_cell      = time_table.Start;     %Begin time
h_end_cell        = time_table.End;       %End time
l_end_cell        = time_table.To;        %End location, stored as cell array
w_t               = time_table.Dist;      %Distance of service trip

n_t           = length(l_start_cell);     %Number of service trips
l_start       = zeros(n_t,1);             %Allocate begin locations
l_end         = zeros(n_t,1);             %Allocate end locations
T             = linspace(1,n_t,n_t)';     %Trip number
e_t           = (w_t/1000)*e_w;           %Fuel usage of service trips
```

```
c_t         = (w_t/1000)*c_w+e_t*c_e;   %Cost of performing service trip

%Transformation of input, if is not according to format
if iscell(h_start_cell)
    [n_rows, ~] = size(h_start_cell);
    h_start = zeros(n_rows,1);
    h_end = zeros(n_rows,1);
    for i = 1:n_rows
        h_start(i) = str2num(h_start_cell{i,1});
        h_end(i) = str2num(h_end_cell{i,1});
    end
else
    h_start = h_start_cell;
    h_end   = h_end_cell;
end

%Change model time to minutes
h_start = round(h_start*24*60);
h_end   = round(h_end*24*60);
h       = round(h*24*60);
h_start_schedule = min(h_start);
h_end_schedule   = max(h_end)+1;

%Convert location name of begin and end location to location number
for i = 1:n_t
    b = l_start_cell{i};
    e = l_end_cell{i};

    [l_start(i),~]  = find(strcmp(b,locations_all.Name) == 1);
    [l_end(i),~]  = find(strcmp(e,locations_all.Name) == 1);
end

%Clear unnecessary variables
clear b_cell e_cell all_loc end_loc bt_cell et_cell i locations b e

% Determine if trips are time compatible
comp    = zeros(n_t);
for i = 1:n_t
    for j = i+1:n_t
        if h_end(i)+h(l_end(i),l_start(j))+h_gap <= h_start(j)
            comp(i,j) = 1;
        end
    end
end

%Begin and end times of time blocks
time_step           = (h_end_schedule-h_start_schedule)/n_z;        %[min]
time_block          = h_start_schedule:time_step:h_end_schedule;    %[-]
time_block_start    = time_block(1:end-1);                          %[-]
time_block_end      = time_block(2:end);                            %[-]
time_blocks         = [time_block_start',time_block_end'];          %[-]

%Transform power input to energy per time-step
max_energy_charging_charger     = ((time_step*60)*max_charge_power_charger*...
                                    1000)/3.6e6;
max_energy_charging_bus         = ((time_step*60)*P_b_max*1000)/3.6e6;
min_energy_charging_bus         = ((time_step*60)*P_b_min*1000)/3.6e6;
max_energy_charging_location    = ((time_step*60)*max_power_charge_loc*1000)...
                                    /3.6e6;

%Give initial vehicle tasks
V_star  = eye(n_t);         %Each service trip a new bus
Szv     = zeros(n_z,n_t);   %No charging occurs (no charging sessions)
```

```
Ezv     = zeros(n_z,n_t);   %No charging occurs (no charging power)
V_star  = [V_star;Szv;Ezv]; %Update vehicle task array

%Give costs of initial vehicle tasks
Cv    = M*ones(1,n_t);     %Give a high value to ensure they are not
                           %present in the final solution

%Calculate charging costs for each time block
price_high_start_block = find(time_blocks(:,1)<= price_high_start &...
    price_high_start <= time_blocks(:,2));
price_high_end_block   = find(time_blocks(:,1)<= price_high_end & ...
    price_high_end <= time_blocks(:,2));
price_energy(price_high_start_block:price_high_end_block) = c_e_high;
```

Once these steps are performed, the column generation problem can be set up. First, the lower and upper bounds of the variables in the subproblem are determined. Next, the arrays that describe the constraints are constructed.

```
%% Set-up subproblem
%Bounds on decision variables
%1.If trip is driven
lb1_sub = zeros(n_t,1);
ub1_sub = ones(n_t,1);

%2.If is being charged
lb2_sub = zeros(n_z,1);
ub2_sub = max_chargers_connected_bus*ones(n_z,1);

%3.How much is being charged
lb3_sub = zeros(n_z,1);
ub3_sub = min([max_energy_charging_bus,max_energy_charging_charger*...
    max_chargers_connected_bus,max_energy_charging_location])*ones(n_z,1);

lb_sub= [lb1_sub;lb2_sub;lb3_sub];
ub_sub= [ub1_sub;ub2_sub;ub3_sub];

%Make constraint that ensures no time incompatible trips will be in vehicle task
%Transform comp matrix to constraint matrix
A1_sub = zeros(1,n_t+n_z+n_z);
k = 0;
for i = 1:n_t
    for j = i+1:n_t
        if comp(i,j) == 0
            if k == 0
                n_rows_A_sub = 0;
                A1_sub(1,i) = 1;
                A1_sub(1,j) = 1;
            else
                n_rows_A_sub = size(A1_sub,1);
                A1_sub(n_rows_A_sub+1,i) = 1;
                A1_sub(n_rows_A_sub+1,j) = 1;
            end
            k = k+1;
        end
    end
end
b1_sub = ones(size(A1_sub,1),1);
clear k

%Make constraint array that will ensure that service trips do not overlap
%with charging sessions
```

```
%Determine in which time blocks a ST has overlap
h_startz = zeros(1,n_t);
h_endz   = zeros(1,n_t);
for i = 1:n_t
    h_startz(i) = find(time_blocks(:,1) <= h_start(i) & ...
        time_blocks(:,2) > h_start(i));
    h_endz(i) = find(time_blocks(:,1) <= h_end(i) & ...
        time_blocks(:,2) > h_end(i));
end

%Ensure that charging cannot in timeblock where ST has overlap
A2_sub = zeros(n_t,n_t+n_z+n_z);
for i = 1:n_t
    A2_sub(i,i) = 1;
    A2_sub(i,n_t+h_startz(i):n_t+h_endz(i)) = 1;
end
b2_sub = ones(size(A2_sub,1),1);

%Ensure that energy is only added to a bus when it is charging
A3_sub = zeros(n_z,n_t+n_z+n_z);
A3_sub(1:n_z,n_t+1:n_t+n_z) = -M*eye(n_z);
A3_sub(1:n_z,n_t+n_z+1:end) = eye(n_z);
b3_sub  = zeros(n_z,1);

%Ensure that when bus is charging, at least a minimal amount of power is added
A4_sub = zeros(n_z,n_t+n_z+n_z);
A4_sub(1:n_z,n_t+1:n_t+n_z) = min_energy_charging_bus*eye(n_z);
A4_sub(1:n_z,n_t+n_z+1:end) = -eye(n_z);
b4_sub = zeros(n_z,1);

%Make constraint that ensures energy level of bus never goes below
%minimal energy level
A5_sub = zeros(n_t,n_t+n_z+n_z);
for i = 1:n_t
    A5_sub(i,1:i) = e_t(1:i);
    A5_sub(i,n_t+n_z+1:n_t+n_z+h_endz(i)) = -ones(1,h_endz(i));
end
A5_sub;
b5_sub = (e_b_max-e_b_min)*ones(n_t,1);

%Make constraint that ensures energy level of bus never goes above
%maximum energy level
A6_sub = zeros(n_t,n_t+n_z+n_z);
for i = 1:n_t
    if i > 1
        A6_sub(i,1:i-1) = -e_t(1:i-1);
    end

    if h_startz(i)-1 == 0
    else
        A6_sub(i,n_t+n_z+1:n_t+n_z+h_startz(i)-1) = ones(1,h_startz(i)-1);
    end
end
b6_sub = zeros(n_t,1);

%Construct inequality constraint of subproblem array
A_sub   = [A1_sub;A2_sub;A3_sub;A4_sub;A5_sub;A6_sub];
b_sub   = [b1_sub;b2_sub;b3_sub;b4_sub;b5_sub;b6_sub];
Aeq_sub = [];
beq_sub = [];
```

Then, the iterative search for new columns can start. In this application, the RMP is used directly to obtain the shadow prices. Thus, the dual of the RMP is not constructed here. Furthermore, two options to obtain these shadow prices are present. One is the option as used in this research, where constraint (6.2b) is an inequality constraint. The other option is to use an equality constraint. When an equality constraint is used, the shadow price for π_τ can be negative.

```
%% Iteratively find new vehicle tasks
Obj_val_RMP_save_all = [];
cut_due_to_time          = zeros(2,1);
reduced_costs_save       = zeros(2,1);
Obj_val_RMP_lin_save     = zeros(2,1);

for i = 1:maxiter

    disp('-----------------------------------------')
    A = ['Iteration number: ', num2str(i)];
    disp(A)

    %Delete initial vehicle tasks from set of vehicle tasks,
    %since they will not be used in the final solution anyway
    if i == 2*n_t && delete_initial_vehicle_tasks == 1
        V_star(:,1:n_t) = [];
        Cv(1:n_t) = [];
    end

    %% RMP, find shadow prices of constraints
    n_v = size(V_star,2);


    if use_eq_RMP == 1
        %Set up RMP
        f_RMP    = Cv;
        A1_RMP   = V_star(n_t+1:n_t+n_z,:);
        b1_RMP   = n_s*ones(n_z,1);
        A2_RMP   = V_star(n_t+n_z+1:n_t+2*n_z,:);
        b2_RMP   = max_energy_charging_location*ones(n_z,1);
        A_RMP    = [A1_RMP;A2_RMP];
        b_RMP    = [b1_RMP;b2_RMP];
        Aeq_RMP  = V_star(1:n_t,:);
        beq_RMP  = ones(n_t,1);
        lb_RMP   = zeros(n_v,1);
        ub_RMP   = ones(n_v,1);

    else
        %Set up RMP
        f_RMP    = Cv;
        A1_RMP   = -V_star(1:n_t,:);
        b1_RMP   = -ones(n_t,1);
        A2_RMP   = V_star(n_t+1:n_t+n_z,:);
        b2_RMP   = n_s*ones(n_z,1);
        A3_RMP   = V_star(n_t+n_z+1:n_t+2*n_z,:);
        b3_RMP   = max_energy_charging_location*ones(n_z,1);
        A_RMP    = [A1_RMP;A2_RMP;A3_RMP];
        b_RMP    = [b1_RMP;b2_RMP;b3_RMP];

        Aeq_RMP  = [];
        beq_RMP  = [];
        lb_RMP   = zeros(n_v,1);
        ub_RMP   = ones(n_v,1);
    end

```

```
    [uv,Obj_val_RMP_lin,~,~,lambda_RMP] = linprog(f_RMP,A_RMP,b_RMP,Aeq_RMP,...
        beq_RMP,lb_RMP,ub_RMP,options_linprog);
    Obj_val_RMP_lin_save(i) = Obj_val_RMP_lin;  %Save for plot
    time_Obj_val_RMP_lin_save(i) = toc;
    uv_lin_save{i} = uv;

    if use_intlin_RMP == 1
        options_intlinprog_RMP = ...
            optimoptions('intlinprog','LPPreprocess', 'none');
        [uv,Obj_val_RMP_intlin,~,~,] = intlinprog(f_RMP,1:n_v,A_RMP,b_RMP,...
            Aeq_RMP,beq_RMP,lb_RMP,ub_RMP,options_intlinprog_RMP);
        Obj_val_RMP_intlin_save(i) = Obj_val_RMP_intlin;
        time_Obj_val_RMP_intlin_save(i) = toc;
        vp_intlin_save{i} = uv;
    end

    %Check if objective function is improving enough. If not, stop adding columns
    if i> n_min_improvement
        if Obj_val_RMP_lin > Obj_val_RMP_lin_save(i-n_min_improvement)*...
                min_improvement
   disp('Improvement on objective function is to low, adding columns is stopped')
            stop_criterion = 2;
            break
        end
    end

    %Check to confirm that objective function of RMP is monotonically decreasing
    if i > 1
       if Obj_val_RMP_lin > Obj_val_RMP_lin_save(i-1) + 1e-6
           Obj_val_RMP_lin_save(i-1)
           Obj_val_RMP_lin
           disp('ERROR: Objective value of RMP is not monotonically decreasing!')
           return
       end
    end

    %Display results
    disp_txt = ['Objective value RMP: ',num2str(Obj_val_RMP_lin)];
    disp(disp_txt)

    %Extract shadows prices from solution of RMP
    if use_eq_RMP == 1
        pi_tau      = lambda_RMP.eqlin(1:n_t);
        theta_zeta  = lambda_RMP.ineqlin(1:n_z);
        rho_zeta    = lambda_RMP.ineqlin(n_z+1:end);
    else
        pi_tau      = -lambda_RMP.ineqlin(1:n_t);
        theta_zeta  = lambda_RMP.ineqlin(n_t+1:n_t+n_z);
        rho_zeta    = lambda_RMP.ineqlin(n_t+n_z+1:end);
    end

    %% Subproblem: Find new column

    %Set intlinprog options dependent on iteration number. A limit is set on
    %calculation time, it is not required that the subproblem is solved to
    %optimality, just that reduced costs are negative. However, closer to optimal
    %yields a better improvement for the basis of the RMP. To prove optimality of
    %the basis, the last iteration should be solved to optimality with
    %non-negative reduced costs
    if i <= n_iterations_switch
        options_subproblem  = optimoptions('intlinprog','Maxtime',...
            maxtime_first_iterations,'Display','none');
    else
```

```
        options_subproblem  = optimoptions('intlinprog','Maxtime',...
            maxtime_later_iterations,'Display','none');
    end

    %Update objective function of subproblem with the new shadow prices
    f_sub = [pi_tau',theta_zeta',rho_zeta'+price_energy'];  %Objective function
    [new_column,obj_val_sub,exitflag,~] = intlinprog(f_sub,1:n_t+n_z,A_sub,...
        b_sub,Aeq_sub,beq_sub,lb_sub,ub_sub,options_subproblem);

    %Check if the solution of the subproblem was solved to optimality, or was
    %stopped due to time and the current feasible solution is saved
    if exitflag == 2
        cut_due_to_time(i) = 1;
    else
        cut_due_to_time(i) = 0;
    end

    %Shift charging session so that if charging occurs between service trips,
    %all the charging occurs consecutively. Charging sessions are shifted toward
    %end of first charging session between the service trips
    if connect_charging_blocks == 1
        if any(new_column(n_t+1:n_t+n_z) > 0)
            for k = 1:n_t
                %If service trip i is performed in the new column
                if new_column(k,1) > 1-error_integer && new_column(k,1)...
                        < 1+error_integer

                    for l = k+1:n_t
                        if new_column(l,1)...
                           > 1-error_integer && new_column(l,1) < 1+error_integer
                            %Find service trip that is performed after i

                            %Check which timeblocks are in between these ST'
                            %First block where charging can occur between k and l
                            begin_charging_domain = h_endz(k)+1;
                            %Last block where charging can occur between k and l
                            end_charging_domain = h_startz(l)-1;

                            %Count how many charging blocks are used in the
                            %charging domain
                            n_charging_blocks_used = ...
                                nnz(new_column(n_t+begin_charging_domain:n_t+...
                                end_charging_domain));

                            if n_charging_blocks_used == 0
                                %If no charging occurs in the domain nothing has
                                %to be done
                            elseif n_charging_blocks_used > 1
                                %If just one charging block is used, then nothing
                                %has to be shifted

                                %Flush all charging sessions to directly after
                                %first charging block

                                %Find used charging blocks between service trips
                                charging_indices = find(new_column(n_t+...
                                    begin_charging_domain:...
                                    n_t+end_charging_domain) ~= 0);
                                charging_indices_temp = (n_t+...
                                    begin_charging_domain-1)*...
                                    ones(1,n_charging_blocks_used);
                                charging_indices = charging_indices_temp'+...
                                    charging_indices;
```

```

                                %Number of chargers used
                                %Determine values old location
                                new_charging_column = ...
                                    new_column(charging_indices);
                                %Overwrite to new location
                                new_column(charging_indices) = ...
                                    zeros(n_charging_blocks_used,1);
                                new_column(charging_indices(1):...
                                    charging_indices(1)+...
                                    n_charging_blocks_used-1) = ...
                                    new_charging_column;

                                %Amount of energy charged
                                %Determine values old location
                                charging_indices = charging_indices+n_z;
                                new_charging_column = ...
                                    new_column(charging_indices);
                                %Overwrite to new location
                                new_column(charging_indices) = ...
                                    zeros(n_charging_blocks_used,1);
                                new_column(charging_indices(1):...
                                    charging_indices(1)+...
                                    n_charging_blocks_used-1) = ...
                                    new_charging_column;
         disp('Charging blocks shifted to have one charging session between ST')
                            end
                            break

                        elseif l == n_t %No service trip is performed after ST i
                        end
                    end
                end
            end
        end
    end

    %Calculate the costs of the new column, only bus price and energy costs are
    %taken into account
    cost_new_column = c_b + sum(new_column(n_t+n_z+1:end)'*price_energy);
    %Calculate the reduced costs, if the reduced costs are non-negative,
    %the basis of known columns is proven to be optimal.
    reduced_costs   = c_b + obj_val_sub;

    if toc > max_computation_time
        stop_criterion = 4;
        disp('Maximum computation time reached, finding new columns is stopped,')
        disp('Continuing to find integer solution')
        break
    end

    if isempty(new_column) == 1
        %Display error if
        disp('ERROR')
        disp('No new column could be generated, infeasible subproblem')
        return
    elseif i ~= maxiter && reduced_costs < min_improv_reduc_costs
        n_iter_needed = i;
        new_column(1:n_t+n_z) = round(new_column(1:n_t+n_z));
        %New column is added to set of known columns
        V_star      = [V_star,new_column];
        %Costs of new column is added to known costs
        Cv          = [Cv,cost_new_column];
```

```
        disp_txt    = ['Reduced costs: ', num2str(reduced_costs)];
        disp(disp_txt)                          %Display results
        reduced_costs_save(i) = reduced_costs;  %Save reduced costs for plot
    elseif i == maxiter
        disp('Maximum number of iterations reached')
        stop_criterion = 3;
        n_iter_needed = i;
        new_column(1:n_t+n_z) = round(new_column(1:n_t+n_z));
        %New column is added to set of known columns
        V_star      = [V_star,new_column];
        %Costs of new column is added to known costs
        Cv          = [Cv,cost_new_column];
        disp_txt    = ['Reduced costs: ', num2str(reduced_costs)];
        disp(disp_txt)                          %Display results
        reduced_costs_save(i) = reduced_costs;  %Save reduced costs for plot
    elseif reduced_costs >= min_improv_reduc_costs
        disp_txt    = ['Reduced costs: ', num2str(reduced_costs)];
        disp(disp_txt)                          %Display results
        disp('No column exists that will improve the basis of the RMP')
        stop_criterion = 1;
        break
    elseif toc > max_computation_time
        stop_criterion = 4;
        break
    end
end
```

Once the columns are generated, the search for an integer solution can start. Again, two options are possible here. The first option is to use an integer solver. The second option is to use the rounding algorithm as described in section 6.2.7.

```
%% Get to integer solution
disp('========================================================')
disp('Starting with finding integer solution')
n_v = size(V_star,2);

searching_for_integer_started   = 1;     %Search for integer solution has started

%Solve to integers. Two methods possible. Direct method using intlinprog to
%solve MP, or by rounding highest integer, add a few more columns and iterate
%until all decision variables are integer.

%%USE INTLINPROG DIRECTLY
%If MP is solved to obtain integer solution, use intlinprog directly
if MP_integer_end == 1
    f_end   = Cv;
    lb_end  = zeros(n_v,1);
    ub_end  = ones(n_v,1);

    if one_bus_per_trip == 1
        A1_end  = V_star(n_t+1:n_t+n_z,:);
        b1_end  = n_s*ones(n_z,1);
        A2_end  = V_star(n_t+n_z+1:n_t+2*n_z,:);
        b2_end  = max_energy_charging_location*ones(n_z,1);
        A_end   = [A1_end;A2_end];
        b_end   = [b1_end;b2_end];
        Aeq_end = V_star(1:n_t,:);
        beq_end = ones(n_t,1);
    else
        A1_end  = -V_star(1:n_t,:);
        b1_end  = -ones(n_t,1);
```

```
        A2_end  = V_star(n_t+1:n_t+n_z,:);
        b2_end  = n_s*ones(n_z,1);
        A3_end  = V_star(n_t+n_z+1:n_t+2*n_z,:);
        b3_end  = max_energy_charging_location*ones(n_z,1);
        A_end   = [A1_end;A2_end;A3_end];
        b_end   = [b1_end;b2_end;b3_end];
        Aeq_end = [];
        beq_end = [];
    end

    [uv,obj_val_end,~,~] = intlinprog(f_end,1:n_v,A_end,b_end,Aeq_end,...
        beq_end,lb_end,ub_end);
    %Round the solution to integers (intlinprog does not always give exact
    %integer solution)
    uv = round(uv);

else
    %%USE ROUNDING TO GET TO INTEGER
    maxiter_end             = extra_columns_after_rounding;     %Reset maxiter
    n_iter_extra_needed     = 0;
    %Track how many extra iterations are made
    n_columns_extra         = 0;
    %Track how many extra columns are added
    integer_fixed           = zeros(n_v,1);
    %Track which columns are fixed to 1
    rounding                = zeros(1,2);
    %Track from which value in vp the columns are rounded to 1
    Obj_val_RMP_extra_save  = [];
    %Track the opbjective value of the RMP
    vp_extra_save           = [];
    %Save the intermediate solutions
    rounding_in_progress    = 1;
    %Indicator that the rounding process is active
    i = 0;


    f_end   = Cv;
    lb_end  = zeros(n_v,1);
    ub_end  = ones(n_v,1);

    if one_bus_per_trip == 1
        A1_end  = V_star(n_t+1:n_t+n_z,:);
        b1_end  = n_s*ones(n_z,1);
        A2_end  = V_star(n_t+n_z+1:n_t+2*n_z,:);
        b2_end  = max_energy_charging_location*ones(n_z,1);
        A_end   = [A1_end;A2_end];
        b_end   = [b1_end;b2_end];
        Aeq_end = V_star(1:n_t,:);
        beq_end = ones(n_t,1);
    else
        A1_end  = -V_star(1:n_t,:);
        b1_end  = -ones(n_t,1);
        A2_end  = V_star(n_t+1:n_t+n_z,:);
        b2_end  = n_s*ones(n_z,1);
        A3_end  = V_star(n_t+n_z+1:n_t+2*n_z,:);
        b3_end  = max_energy_charging_location*ones(n_z,1);
        A_end   = [A1_end;A2_end;A3_end];
        b_end   = [b1_end;b2_end;b3_end];
        Aeq_end = [];
        beq_end = [];
    end


```

```
    % Solve RMP to find values for decision variables
    [uv,obj_val_end,~,~,lambda_end] = linprog(f_end,A_end,b_end,Aeq_end,...
        beq_end,lb_end,ub_end,options_linprog);


    while rounding_in_progress == 1
        clear integer
        extra_column_counter = 0;
        i = i+1;

        %Find location of decision variables that are integer
        new_integer = find((uv-1 < error_integer & uv-1 > -error_integer) &...
            integer_fixed ~= 1);
        n_integer   = size(new_integer,1);

        %If the entire solution is integer, stop with algorithm
        if all(ismember(uv,[1,0])) == 1
            disp('All resulting decision variables are integer')
            rounding_in_progress = 0;
            break

        %For the decision variables that are integer, add constraint to fix
        %them to that value. This way they do not change any more in the future.
        elseif isempty(new_integer) == 0
            new_Aeq = zeros(n_integer,n_v);
            disp('One or multiple decision variables are 1.')
            disp('Adding constraint to fix them to one.')

            for j = 1:n_integer
                if integer_fixed(new_integer(j)) ~= 1
                    new_Aeq(j,new_integer(j)) = 1;
                    disp_txt = ['Variables: ', num2str(new_integer(j)),...
                        ' fixed to 1'];
                    disp(disp_txt)
                    integer_fixed(new_integer(j)) = 1;
                end
            end
            new_beq = ones(n_integer,1);

        %%If no integers are found, round the nearest value to 1 to 1
        %(no first columns) and re-iterate
        else
            vp_temp                 = zeros(n_v,1);
            loc_to_assess           = find(integer_fixed == 0);
            vp_temp(loc_to_assess)  = uv(loc_to_assess);
            error                   = abs(vp_temp-1);
            error(1:n_t)            = M;    %Prevent dummy columns to be chosen
            [loc_close_integer,~]   = find(error == min(error) &...
                integer_fixed ~= 1);
            loc_close_integer       = loc_close_integer(end);

            %Round up highest value to 1
            if integer_fixed(loc_close_integer) == 0
                disp('Not all resulting decision variables are integer')
                disp_txt = ['Decision variable: ', num2str(loc_close_integer),...
                    ' is rounded from: ',num2str(uv(loc_close_integer)),' to 1'];
                disp(disp_txt)

                integer_fixed(loc_close_integer) = 1;
                new_Aeq = zeros(1,n_v);
                new_Aeq(1,loc_close_integer) = 1;
                rounding(i,1) = loc_close_integer;
                rounding(i,2) = uv(loc_close_integer);
```

```
                new_beq = 1;
            end

            %% Add more columns to V_start. Code is omitted here.

            integer_fixed = [integer_fixed;zeros(extra_column_counter,1)];

        end

        %%Solve RMP with new columns
        n_v                 = size(V_star,2);

        %Add constraint that the chosen decision variables are 1

        f_end   = Cv;
        lb_end  = zeros(n_v,1);
        ub_end  = ones(n_v,1);

        if one_bus_per_trip == 1
            A1_end  = V_star(n_t+1:n_t+n_z,:);
            b1_end  = n_s*ones(n_z,1);
            A2_end  = V_star(n_t+n_z+1:n_t+2*n_z,:);
            b2_end  = max_energy_charging_location*ones(n_z,1);
            A_end   = [A1_end;A2_end];
            b_end   = [b1_end;b2_end];
            Aeq_end = V_star(1:n_t,:);
            beq_end = ones(n_t,1);
        else
            A1_end  = -V_star(1:n_t,:);
            b1_end  = -ones(n_t,1);
            A2_end  = V_star(n_t+1:n_t+n_z,:);
            b2_end  = n_s*ones(n_z,1);
            A3_end  = V_star(n_t+n_z+1:n_t+2*n_z,:);
            b3_end  = max_energy_charging_location*ones(n_z,1);
            A_end   = [A1_end;A2_end;A3_end];
            b_end   = [b1_end;b2_end;b3_end];
            Aeq_end = [];
            beq_end = [];
        end

        %Add constraint for integers that are fixed
        n_integers_fixed_total = sum(integer_fixed);
        loc_integer_fixed = find(integer_fixed == 1);
        Aeq_fixed = zeros(n_integers_fixed_total,n_v);
        for k = 1:n_integers_fixed_total
            Aeq_fixed(k,loc_integer_fixed(k)) = 1;
        end
        beq_fixed = ones(n_integers_fixed_total,1);
        Aeq_end = [Aeq_end;Aeq_fixed];
        beq_end = [beq_end;beq_fixed];


        [uv,obj_val_end,exitflag_rounding,~,lambda_end] = linprog(f_end,A_end,...
            b_end,Aeq_end,beq_end,lb_end,ub_end,options_linprog);

        if exitflag_rounding == -2 || exitflag_rounding == -5
            disp('ERROR: After rounding some solutions for vp,')
            disp('no feasible solution is possible any more, more columns')
            disp('should be addded at rounding stage')
            return
        elseif exitflag_rounding ~= 1
            disp('ERROR: Some error in rounding to integer solution')
        end
```

```
    end
end
```

The final step is to analyze the results and to plot the figures. Also, the most important variables are saved for future reference.

```
%% Analyse, save and plot results

tasks_used = V_star(:,find(uv == 1));
buses_used = size(tasks_used,2);

%Sort tasks_used on starting time
for i = 1:buses_used
    for j = 1:n_t
        if tasks_used(j,i) ~= 0
            first_trip(i) = j;
            break
        end
    end
end

[~,old_loc] = sort(first_trip);

for i = 1:buses_used
    tasks_used_new(:,i) = tasks_used(:,old_loc(i));
end


clear first_trip old_loc
tasks_used          = tasks_used_new
combined_solution   = sum(tasks_used,2)
computation_time    = toc
total_cost          = sum(obj_val_end)
mean_trips          = mean(combined_solution(1:n_t,:))
buses_used
stop_criterion
n_iter_needed

%%PLOTTING SOC OF ALL USED VEHICLE TASKS
n_v = size(tasks_used,2);
for i = 1:n_v
    for j = 1:n_z

        %Remove energy of performing service trip at first timeblock of service trip
        %Service trips performed(or started) up to time step j
        temp2 = max(find(h_startz <= j));
         %Service trips performed in i up to and including j
        [temp3,~] = find(tasks_used(1:temp2,i) > 1-error_integer & ...
            tasks_used(1:temp2,i) <1+error_integer);
        energy_used(j+1,i) = sum(e_t(temp3));
        energy_charged(j+1,i) = sum(tasks_used(n_t+n_z+1:n_t+n_z+j,i));
        energy_level(j+1,i) = e_b_max-energy_used(j+1,i)+energy_charged(j+1,i);

    end
    energy_level(1,i) = e_b_max;
end

%Go from energy level to SoC
SoC_level = ((energy_level)/battery_capacity)*100;
figure
for i = 1:n_v
```

```
    plot(time_block'/60,SoC_level(:,i),'LineWidth',linewidth_figure)
    grid on
    hold on
end
set(gca,'FontSize',fontsize_figure)
xlabel('Time [h]','fontname','Helvetica Neue')
ylabel('SoC of bus [%]','fontname','Helvetica Neue')
axis([(min(time_block-30)/60) (max(time_block+30)/60) 0 100])
xticks(floor((min(time_block-30)/60)):1:ceil((max(time_block+30)/60)))
if save_results == 1
    filename = ['CG_csoc_test',test_schedule,'_steps_',num2str(n_z),...
        '_chargers_',num2str(n_s),'more'];
    print(filename,'-dpng')
    print(filename,'-dpdf')
    print(filename,'-depsc')
    savefig(filename)
end

%%PLOTTING NUMBER OF CHARGERS AVAILABLE
n_chargers_used = combined_solution(n_t+1:n_t+n_z);
n_chargers_available(1:n_z) = n_s-n_chargers_used;

for i = 1:n_z
    temp(i*2) = n_chargers_available(i);
end
for i = 2:2:2*n_z
    n_chargers_available(i-1) = temp(i);
    n_chargers_available(i) = temp(i);
end
clear temp

time_plotting = [];
for i = 1:size(time_blocks,1)
    time_plotting = [time_plotting, time_blocks(i,:)];
end

figure
plot(time_plotting/60,n_chargers_available,'LineWidth',linewidth_figure)
set(gca,'FontSize',fontsize_figure)
grid on
xlabel('Time [h]','fontname','Helvetica Neue')
ylabel('Number of chargers available [-]','fontname','Helvetica Neue')
xticks(floor((min(time_block-30)/60)):1:ceil((max(time_block+30)/60)))
yticks(0:1:n_s+1)
axis([(min(time_block-30)/60) (max(time_block+30)/60) -.5 n_s+.5])
legend('Number of chargers available')
if save_results == 1
    filename = ['CG_chargers_test',test_schedule,'_steps_',num2str(n_z),...
        '_chargers_',num2str(n_s),'more'];
    print(filename,'-dpng')
    print(filename,'-dpdf')
    print(filename,'-depsc')
    savefig(filename)
end

%%PLOTTING POWER USAGE OF THE GRID
grid_power = combined_solution(n_t+n_z+1:n_t+2*n_z)*(3.6e3/(60*time_step));
for i = 1:n_z
    temp(i*2) = grid_power(i);
end
for i = 2:2:2*n_z
    grid_power_plot(i-1) = temp(i);
    grid_power_plot(i) = temp(i);
```

```
end
clear temp

figure
plot(time_plotting/60,grid_power_plot,'LineWidth',linewidth_figure)
set(gca,'FontSize',fontsize_figure)
grid on
xlabel('Time [h]','fontname','Helvetica Neue')
ylabel('Power delivered by grid [kW]','fontname','Helvetica Neue')
xticks(floor((min(time_block-30)/60)):1:ceil((max(time_block+30)/60)))
axis([(min(time_block-30)/60) (max(time_block+30)/60) -10 max(grid_power)+50])
legend('Power delivered by grid [kW]')
if save_results == 1
    filename = ['CG_gridpower_test',test_schedule,'_steps_',num2str(n_z),...
        '_chargers_',num2str(n_s),'more'];
    print(filename,'-dpng')
    print(filename,'-dpdf')
    print(filename,'-depsc')
    savefig(filename)
end


%%PLOTTING GANTT CHART
figure
set(gca,'FontSize',fontsize_figure)
hold on
grid on
xlabel('Time [h]','fontname','Helvetica Neue')
ylabel('Vehicle number [-]','fontname','Helvetica Neue')
axis([min(h_start)/60-0.5 max(h_end)/60+0.5 0 buses_used+1])
yticks(0:1:buses_used+1)
xticks(floor(min(h_start)/60-0.5):1:(max(h_end)/60+0.5))
%Plotting service trips
for i = 1:buses_used
    for k = 1:n_t       %Service trips
        if tasks_used(k,i)-1 <= error_integer && tasks_used(k,i)-1 >= ...
                -error_integer
            begin_time_trip = h_start(k)/60;
            end_time_trip = h_end(k)/60;
            service_trip_plot = plot([begin_time_trip end_time_trip],[i i],...
            '-','Color',[0 0.75 0.75],'LineWidth',10);
        end
    end
end
%Plotting charging sessions
for i = 1:buses_used
    for k = n_t+1:n_t+n_z       %Charging
        if tasks_used(k,i) > 0.2
            begin_time_charge = time_block_start(k-n_t)/60;
            end_time_charge = time_block_end(k-n_t)/60;
            charging_plot = plot([begin_time_charge end_time_charge],[i i],...
                '-','Color',[1 0.5 0.5],'LineWidth',5);
        end
    end
end

if exist('deadhead_trip_plot') && exist('charging_plot')
    legend([service_trip_plot,charging_plot,deadhead_trip_plot],...
        'Service Trips', 'Charging','Deadhead Trips','Location','northwest')
elseif exist('charging_plot')
    legend([service_trip_plot,charging_plot],'Service Trips','Charging',...
        'Location','northwest')
else
```

```
    legend([service_trip_plot],'Service Trips','Location','northwest')
end
if save_results == 1
    filename = ['CG_Gantt_test',test_schedule,'_steps_',num2str(n_z),...
        '_chargers_',num2str(n_s)];
    print(filename,'-dpng')
    print(filename,'-dpdf')
    print(filename,'-depsc')
    savefig(filename)
end

%PLOT REDUCED COSTS
reduced_costs_save = reduced_costs_save(1:n_iter_needed-1);
x = 1:n_iter_needed-1;
figure
plot(x,-reduced_costs_save,'LineWidth',linewidth_figure)
set(gca,'FontSize',fontsize_figure)
xlabel('Iteration number [-]','fontname','Helvetica Neue')
ylabel('Reduced costs','fontname','Helvetica Neue')
legend('Reduced costs')
grid on
if save_results == 1
    filename = ['CG_reduced_costs_test',test_schedule,'_steps_',num2str(n_z),...
        '_chargers_',num2str(n_s)];
    print(filename,'-dpng')
    print(filename,'-dpdf')
    print(filename,'-depsc')
    savefig(filename)
end

figure
x = 1:n_iter_needed;
plot(x,Obj_val_RMP_lin_save,'LineWidth',linewidth_figure)
set(gca,'FontSize',fontsize_figure)
xlabel('Iteration number [-]','fontname','Helvetica Neue')
ylabel('Objective value RMP','fontname','Helvetica Neue')

if use_intlin_RMP == 1
    hold on
    plot(x,Obj_val_RMP_intlin_save,'LineWidth',linewidth_figure)
    legend('Objective value RMP','Objective value MP')
else
    legend('Objective value RMP')
end

grid on
if save_results == 1
    filename = ['CG_obj_val_RMP_test',test_schedule,'_steps_',num2str(n_z),...
        '_chargers_',num2str(n_s)];
    print(filename,'-dpng')
    print(filename,'-dpdf')
    print(filename,'-depsc')
    savefig(filename)
end

%save output
if save_results == 1
    filename = ['CG_output',test_schedule,'_steps_',num2str(n_z),'_chargers_',...
        num2str(n_s),'more'];
    if use_intlin_RMP == 1
        if MP_integer_end == 0
            save(filename,'buses_used','computation_time',...
                'Obj_val_RMP_lin_save','time_Obj_val_RMP_lin_save',...
```

```
                'stop_criterion','mean_trips','total_cost',...
                'tasks_used','c_b','combined_solution','Cv',...
                'n_iter_needed','cut_due_to_time','n_s','V_star',...
                'test_schedule','time_step','uv_lin_save','MP_integer_end',...
                'maxiter','use_eq_RMP','use_intlin_RMP',...
                'Obj_val_RMP_intlin_save','rounding')
        else
            save(filename,'buses_used','computation_time',...
                'Obj_val_RMP_lin_save','time_Obj_val_RMP_lin_save',...
                'stop_criterion','mean_trips','total_cost','tasks_used',...
                'c_b','combined_solution','Cv','n_iter_needed',...
                'cut_due_to_time','n_s','V_star','test_schedule','time_step',...
                'uv_lin_save','MP_integer_end','maxiter','use_eq_RMP',...
                'use_intlin_RMP','Obj_val_RMP_intlin_save')
        end
    else
        if MP_integer_end == 0
            save(filename,'buses_used','computation_time',...
                'Obj_val_RMP_lin_save','time_Obj_val_RMP_lin_save',...
                'stop_criterion','mean_trips','total_cost','tasks_used','c_b',...
                'combined_solution','Cv','n_iter_needed','cut_due_to_time',...
                'n_s','V_star','test_schedule','time_step','uv_lin_save',...
                'MP_integer_end','maxiter','use_eq_RMP',...
                'use_intlin_RMP','rounding')
        else
            save(filename,'buses_used','computation_time',...
                'Obj_val_RMP_lin_save','time_Obj_val_RMP_lin_save',...
                'stop_criterion','mean_trips','total_cost','tasks_used','c_b',...
                'combined_solution','Cv','n_iter_needed','cut_due_to_time',...
                'n_s','V_star','test_schedule','time_step','uv_lin_save',...
                'MP_integer_end','maxiter','use_eq_RMP','use_intlin_RMP')
        end
    end
end
```