Master Thesis

Electric Cars - Efficient Centralized Charging

Pit Schneider

Master Thesis DKE-19-24

Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science of Artificial Intelligence
at the Department of Data Science and Knowledge Engineering
of the Maastricht University

Thesis Committee:

Dr. Matúš Mihalák
Dr. Georgios Stamoulis

Maastricht University
Faculty of Science and Engineering
Department of Data Science and Knowledge Engineering

June 20, 2019

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Problem Definition . . . 1
- 1.3 Related Work . . . 2

2 Definitions **4**

3 Observations **6**
- 3.1 First Observations . . . 6
  - 3.1.1 Detecting critical charging stations . . . 6
  - 3.1.2 Schedule ordering influences program cost . . . 6
  - 3.1.3 Balancing leads to minimal waiting . . . 7
- 3.2 Program Conversions . . . 7
  - 3.2.1 Conversion to reduced programs . . . 7
  - 3.2.2 Conversion to independent programs . . . 8
  - 3.2.3 Conversion to serial programs . . . 9

4 Critical Blocks - First Algorithm **12**
- 4.1 Computing i^* . . . 12
- 4.2 Algorithm Approach . . . 13
  - 4.2.1 Single independent schedule . . . 13
  - 4.2.2 Multiple independent schedules . . . 14
- 4.3 Performance Analysis . . . 15
  - 4.3.1 Analyzing schedule charging cost . . . 15
  - 4.3.2 Analyzing total waiting cost . . . 16
  - 4.3.3 Disproving instance . . . 16

5 Critical Blocks - Second Algorithm **18**
- 5.1 Computing j^* . . . 18
- 5.2 Algorithm Approach . . . 19
  - 5.2.1 Time complexity . . . 20
  - 5.2.2 Cost calculation . . . 20
- 5.3 Performance Analysis . . . 20

6 Critical Blocks - Third Algorithm **22**
- 6.1 Distributing Cars . . . 22
  - 6.1.1 Procedure . . . 22
  - 6.1.2 Correctness . . . 23
- 6.2 Distributing Charging Stations . . . 23
  - 6.2.1 Optimum distribution . . . 23
  - 6.2.2 Perfectly unbalanced . . . 24
- 6.3 Algorithm Approach . . . 24

7 Critical Blocks - Fourth Algorithm **26**
- 7.1 Computing $u^*(s)$ . . . 26
- 7.2 Algorithm Approach . . . 26
- 7.3 Performance Analysis . . . 27
  - 7.3.1 Analysis for critical blocks . . . 27
  - 7.3.2 Analysis for arbitrary roads . . . 28
- 7.4 Algorithm Comparisons . . . 29

8 Conclusion **30**

Abstract

A planning and scheduling problem is defined using a path graph, with a subset of the nodes denoting charging stations, to be the model for the road of n electric cars. Centralized planning is used to select a subset of charging stations for every car, in a way that the cost, defined in terms of number of rechargings and time spent queuing at the stations, is minimized during the road traversal. The structure and properties of efficient charging plans are explored, followed by an algorithmic analysis of computing solutions for the problem. More precisely, four algorithms which are designed to solve a specific set of input instances, are presented in an incremental approach. This set of instances is characterized by certain properties of the subset of charging stations induced by the cars' battery capacity. Along the way, methods to compute essential values for given input instances, useful to the understanding and development of an optimum algorithm, are proposed.

Chapter 1

Introduction

1.1 Context

The transition to cars that are powered entirely by electricity is undoubtedly gaining a certain momentum in recent years. Due to more and more governments actively supporting all-electric roads, advancements in battery technology, and a progressively changing public opinion, it seems unavoidable that electrical cars will surge in popularity in the future. However, it is commonly known that charging electric cars takes a considerable amount of time compared to traditional petrol driven cars. On top of that, charging stations along the roads might not be available in large quantities, especially during the early transition phase. One can imagine that, due to the lack of infrastructure, it will be problematic to serve demand optimally. As a result, drivers may encounter long waiting times to charge their cars.

Centralized planning combined with a cooperative strategy could be a way to reduce the impact of this shortcoming and could turn out to be convenient for future inter-connected electric cars. By ensuring that all actors cooperate, valuable time can be saved by making the right choice in terms of which charging station to use. That is why the goal of this work is to make use of centralized planning in order to develop a charging plan for everyone. In particular, the findings presented in this thesis aim to analyze the computation of an optimal plan for all cars on the road. This is conducted by investigating the structure and properties of plans in general. A next undertaking is to decide how to make use of those findings in order to compute optimum, or near optimum plans. For the purpose of determining an optimum, a cost function is defined. It is designed to evaluate the combined travel time of all the cars and to serve as a benchmark for the efficiency of the plan. Furthermore, a simple model to represent a single road with charging stations is created and used throughout this thesis.

This chapter will, in the following, lay out a formal problem definition and make the connection to related work. From there on, the thesis is organized as follows:

- Chapter 2 is dedicated to expand the terminology and definitions of the problem in question.
- Chapter 3 makes use of the definitions to perform some early observations. Those are of particular use in the following algorithmic chapters.
- Chapters 4-7 each introduce and discuss an algorithm tailored to solve a specific set of problem instances.
- Chapter 8, in a concluding way, serves as a review of the most important aspects of this thesis.

1.2 Problem Definition

The problem consists of n electrical cars, contained in the set $A = \{a_1, a_2, ..., a_n\}$. Given car a_i, we refer to i as the index of the car. Also, every car in A has the same battery capacity k. In this work, time is discretized into units of time steps. The problem starts at time $t = 0$ where all the cars are initialized in a fully charged state. Let $G=(V, E)$ denote a path graph having length $\ell=|E|$ and nodes enumerated along the path with indices from 0 to ℓ. The road on which the cars are driving is modeled by G. At $t = 0$ all the cars are located at node v_0 and start driving. For this problem, driving is constrained to a single direction, which is towards v_ℓ. The cars stop driving when having arrived at v_ℓ.

Furthermore, we consider $V = \{0, 1, ..., \ell\}$ to be the set of all node indices in G. We use this to define $K \subseteq V$, denoting the subset containing the indices of all charging stations, ordered increasingly according to the node indices. Since the cars only start at v_0, arrive at v_ℓ, and never charge at the respective nodes, it is irrelevant for the problem whether v_0 and v_ℓ are contained in K or not.

Every charging station in K can serve a single car only at each time step and always recharges a car's battery to full capacity. A queue is associated to every charging station. Cars that want to charge at a charging station, but cannot because it is occupied, are sent to the queue. If there are multiple cars arriving simultaneously at a non occupied charging station, the car with the lowest index has priority to charge, all other cars join the queue. Multiple cars joining a queue at the same time underlies the rule of the cars joining the queue in increasing order of their index.

Traversing an edge in G, as well as recharging at a charging station, takes one time unit. Additionally, an edge traversal decreases the battery level of the driving car by one unit. Hence, since the capacity of the battery is k, any car can drive along at most k edges without charging.

Definition 1.2.1 (**Feasible schedule**). A feasible schedule S_i of a car a_i are the indices of the charging stations at which a_i charges, s.t.

$$\begin{aligned}&\forall q \in S_i : \\ &\exists p \in S_i \cup \{0\} \text{ s.t. } q - p \leq k \text{ and } p < q \text{ and} \\ &\exists r \in S_i \cup \{\ell\} \text{ s.t. } r - q \leq k \text{ and } r > q. \end{aligned} \tag{1.1}$$

Similarly to K, every schedule S_i is seen as ordered increasingly.

In this thesis, only path graphs having $\ell > k$ are considered since $S_i = \emptyset$, i.e. not charging at all, is not of particular interest. Additionally, we restrain the work to feasible schedules only and refer to them as simply "schedules".

The problem is seen as being built around four parameters, named K, n, k, and ℓ. Assigning values to all parameters has the effect of creating a problem instance I. An example instance can be seen with Figure 1.1.

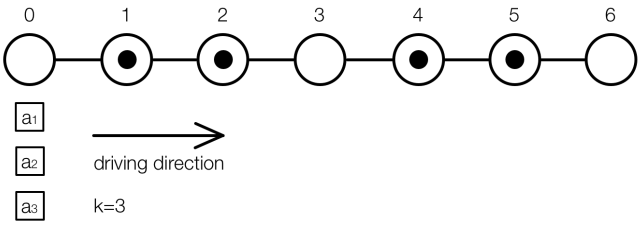

Figure 1.1: Example instance I with $K = \{1, 2, 4, 5\}$, $n = 3$, $k = 3$, and $\ell = 6$, depicted at $t=0$. A possible feasible schedule for any car a_i is $S_i = \{2, 5\}$.

Definition 1.2.2 (**Program**). A program P, for a given instance I, is defined as a sequence of feasible schedules associated to every car, i.e.,

$$P = (S_1, S_2, ..., S_n), \tag{1.2}$$

where S_i is the schedule of car a_i.

Given a program P, two costs are associated to every car a_i:

- The charging cost $c_i(P)$.
  For every recharging of car a_i, the value of $c_i(P)$ is increased by one. Hence, $c_i(P)$ is equal to the number of stops in S_i, i.e. $c_i(P) = |S_i|$.
- The waiting cost $w_i(P)$.
  Similarly, for every time step car a_i has to wait in a queue, $w_i(P)$ is increased by one.

Definition 1.2.3 (**Cost of a program**). The *cost* of a program P is defined as

$$cost(P) = \sum_{i=1}^{n} \Big(c_i(P) + w_i(P) \Big). \tag{1.3}$$

It should be noted that for every car a_i, the value of $c_i(P) + w_i(P)$ is equal to the number of time steps that a_i requires to arrive at v_ℓ, without the time needed for all ℓ edge traversals.

Furthermore, a program P is said to be *optimum* if P has minimum cost. Using (1.3), a minimization problem is defined. EfficientCharging is the problem of computing an optimum program P for any input instance I.

In the context of this thesis, two main tasks are looked at:

- The study of the structural properties of programs. This undertaking will help to establish theorems that, for example, reason about the existence of at least one optimum program with a specific property. Another used logical proposition is that there is no optimum program for a given program structure. All of this knowledge effectively lays the groundwork for the next task.
- The development of an algorithm that solves any problem instance in a reasonable time. To be more precise, the main objective is to analyze the existence of an algorithm that runs in polynomial time. Also, due to the complexity of EfficientCharging, the algorithms presented in this thesis are designed to deal with a subset of problem instances as input only.

1.3 Related Work

After having defined EfficientCharging in a formal way, it is time to connect it to the existing literature. Certainly it can be stated that EfficientCharging falls under the category of planning and scheduling problems. The field often refers to "jobs" and "machines", where the former needs to be processed by the latter. It seems natural that, in the case of the problem in question, cars are represented by jobs and charging stations by machines.

The class of flow shop scheduling problems is probably aligning the most with EfficientCharging. Although the limitation, that no machine can process more than one job at a time, is clearly part of EfficientCharging as well, an obvious difference is the fact that not every job has to be necessarily processed by every machine. The subset of machines has to be carefully selected by considering the battery capacity of the cars.

After some research, to our knowledge, it seems that this exact variation of a flow shop scheduling problem hasn't been studied and published yet. Nevertheless, it should be mentioned that in the context of a Bachelor's thesis [1], the game theoretic questions of the problem have been investigated. Also, the work of an internship [2], that has been focusing on the algorithmic problem of relatively simple instances of EfficientCharging, provides the baseline for this thesis. Those instances will not be covered in the following and we consider this as a continuation of the previous work.

Chapter 2

Definitions

In this chapter, we introduce further terminology that we use to study the problem EFFICIENTCHARGING. To start with, several small definitions concerning cost, schedules and charging stations are given.

Definition 2.0.1 (**Total charging and waiting cost**). The total charging cost $C(P)$ and the total waiting cost $W(P)$ of a program P are defined as

$$C(P) = \sum_{i=1}^{n} c_i(P) \text{ and } W(P) = \sum_{i=1}^{n} w_i(P). \tag{2.1}$$

Definition 2.0.2 (**Cost originating from a charging station**). Given a program P and an index of a charging station $i \in K$, the total number of time steps that all cars use to charge at v_i or wait in the queue at v_i, is called the cost *originating* from v_i.

Definition 2.0.3 (**Greedy schedule**). Given K, k, and ℓ, we denote by S_{greedy} the schedule that does a maximum of edge traversals, with respect to k and the indices in K, before every charging stop. Also c_{opt} is defined as

$$c_{opt} = |S_{greedy}|. \tag{2.2}$$

For instance, using $K = \{1, 2, 4, 5\}$, $k = 3$, and $\ell = 6$ (values used for Figure 1.1), we would obtain $S_{greedy} = \{2, 5\}$ and $c_{opt} = 2$.

Definition 2.0.4 (**Number of unique schedules**). Given a program P, the function $unique(P)$ returns the set containing all unique schedules that are contained in P. Furthermore we denote by s the cardinality of the set, i.e.

$$s = |unique(P)|. \tag{2.3}$$

Definition 2.0.5 (**Unique charging stations**). Given a program P, the function $uniqueStations(P)$ returns the set containing all indices of unique charging stations that are contained in P. Furthermore we denote by u the cardinality of the set, i.e.

$$u = |uniqueStations(P)|. \tag{2.4}$$

Please note that the above definition is also valid in the exact same way for a set of schedules which does not necessarily represent a program.

Definition 2.0.6 (**Critical charging station**). Given K, k, and ℓ, a *critical* charging station is a charging station that is contained in every feasible schedule.

Next, we define the properties *proportional*, *reduced*, *independent* and *serial*. They can be applied to programs and sometimes to sets of schedules as well.

Definition 2.0.7 (**Proportional program**). A program P is *proportional* if every schedule in P has the same number of charging stations.

Definition 2.0.8 (**Reduced program**). A program $P = (S_1, S_2, ..., S_n)$ is *reduced* if P satisfies

$$\begin{aligned} &\forall S_i \in P : \nexists\, m, n, o \in S_i \cup \{0, \ell\} \\ &\text{s.t. } m < n < o \text{ and } o - m \leq k. \end{aligned} \tag{2.5}$$

Definition 2.0.9 (**Independent set of schedules**). A set of schedules is *independent* if no pair of schedules belonging to the set contains a common charging station.

Definition 2.0.10 (**Independent program**). A program P is *independent* if $unique(P)$ is independent.

Definition 2.0.11 (**Serial set of schedules**). An ordered set of schedules Y is *serial* when Y is independent and Y follows the format of

$$\begin{aligned} &Y = \{\{v_1^1, v_2^1, ..., v_{x_1}^1\}, \{v_1^2, v_2^2, ..., v_{x_2}^2\}, ..., \{v_1^s, v_2^s, ..., v_{x_s}^s\}\} \\ &\bullet \text{ with } s > 0 \text{ and } x_1 > 0, x_2 > 0, ..., x_S > 0, \\ &\bullet \ \forall m, n \text{ s.t. } 1 \leq m < n \leq s : 0 \leq x_n - x_m \leq 1, \\ &\bullet \ \forall v_i^j, v_{i'}^{j'} : i = i' \text{ and } j < j' \implies v_{i'}^{j'} < v_i^j, \\ &\bullet \ \forall v_i^j, v_{i'}^{j'} : i < i' \implies v_i^j < v_{i'}^{j'}. \end{aligned} \tag{2.6}$$

An example of a serial set of schedules is the set $Y = \{\{3, 6\}, \{2, 5, 8\}, \{1, 4, 7\}\}$.

Definition 2.0.12 (**Serial program**). A program P is *serial* if $unique(P)$ is serial.

Furthermore, the following Definition 2.0.13 describes a way to evenly distribute (balance) cars over a set of charging stations. The idea is to prevent long queues and waiting times.

Definition 2.0.13 (**Balancing cars over charging stations**). Given K, k, and ℓ, let $A' \subseteq A$ denote a subset of cars and let $K' \subseteq K$, with $|K'| \leq k$, denote a subset of charging stations. In the context of some program P, let $A'_i \subseteq A'$ denote the cars that charge at $i \in K'$. Let *small* denote the smallest node index in K'. Let *center* define the smallest node index overall, s.t.

$$small \leq center \text{ and } center \bmod k = 0. \tag{2.7}$$

A' is *balanced* over K' when every $a_j \in A'$ charges at exactly one station in K', s.t.

$$\begin{aligned}
&\forall i \in K' : |A'_i| = \left\lceil \frac{|A'|}{|K'|} \right\rceil \text{ or } |A'_i| = \left\lfloor \frac{|A'|}{|K'|} \right\rfloor \text{ and} \\
&\forall m, n \in K': \\
&\bullet\ |A'_m| \leq |A'_n| \text{ if } m < n \leq \mathit{center}, \\
&\bullet\ |A'_m| \geq |A'_n| \text{ if } m \leq \mathit{center} < n, \\
&\bullet\ |A'_m| \leq |A'_n| \text{ if } \mathit{center} < m < n. \qquad (2.8)
\end{aligned}$$

Using the next Definition 2.0.14, a subset $\mathcal{I}^{cb}$ of problem instances, is defined. The subsequent Definition 2.0.15 provides further terminology regarding those instances.

Definition 2.0.14 (**Critical blocks**)**.** A *block* of charging stations is defined by the set

$$\begin{aligned}
&B = \{f, f+1, ..., g-1, g\}, \text{ where:} \\
&\bullet\ 0 < f < g < \ell, \\
&\bullet\ \forall i \text{ s.t. } f \leq i \leq g : i \in K, \text{ and} \\
&\bullet\ f-1 \notin K \text{ and } g+1 \notin K. \qquad (2.9)
\end{aligned}$$

In another way, a block can be described as a maximal set of consecutive charging stations. We additionally define

$$\begin{aligned}
\mathit{start}(B) &= f, \\
\mathit{end}(B) &= g, \\
\mathit{length}(B) &= g - f + 1 = |B|. \qquad (2.10)
\end{aligned}$$

A block is *critical* if every car has to charge at at least one charging station belonging to the block. Let b define the total number of blocks, let B_1 denote the block that is the closest to v_0 and let B_m define the mth block along the path graph. If every block is critical, then

$$\begin{aligned}
&\forall m \text{ s.t. } 3 \leq m \leq b\text{: } \mathit{start}(B_m) - \mathit{end}(B_{m-2}) > k \text{ and} \\
&\bullet\ \mathit{start}(B_2) > k, \\
&\bullet\ \ell - \mathit{end}(B_{b-1}) > k. \qquad (2.11)
\end{aligned}$$

Furthermore, we denote by $\mathcal{I}^{cb}$ the set of all problem instances that satisfy (2.11).

Definition 2.0.15 (**Zones inside critical blocks**)**.** The *arriving zone* of a critical block is defined as the set of charging stations that can serve as the first stop inside a block. Similarly, the *leaving zone* of a critical block is denoted as the set of charging stations that can serve as the last stop inside a block.

Using $\mathit{end}(B_0) = 0$ and $\mathit{start}(B_{b+1}) = \ell$ only for the following definition, there is for every B_m

$$\begin{aligned}
&\mathit{arrivingZone}(B_m) \\
&= \{i \mid i \in B_m \text{ and } (i - \mathit{end}(B_{m-1})) \leq k\} \text{ and} \\
&\mathit{leavingZone}(B_m) \\
&= \{i \mid i \in B_m \text{ and } (\mathit{start}(B_{m+1}) - i) \leq k\}. \qquad (2.12)
\end{aligned}$$

We consider zones to be a special type of block, that is why the equations in (2.10) also hold for zones.

Chapter 3

Observations

After having established the terminology and definitions of EfficientCharging, we use this chapter to provide preliminary observations and theorems. Especially the findings of the second section, investigating the conversion of programs, are of importance for the later stages of this thesis.

3.1 First Observations

To start, this section covers some early observations concerning schedule ordering, critical charging stations and waiting time.

3.1.1 Detecting critical charging stations

The definition of a critical charging station has been given by Definition 2.0.6. To detect all critical charging stations, the following simple algorithm is proposed.

Algorithm 1 Detecting all critical charging stations

```
1: function (K, k, ℓ)
2:     Critical ← ∅
3:     left ← 0
4:     station ← K[0]
5:     K ← (K \ K[0]) ∪ {ℓ}
6:     for each right ∈ K do
7:         if (right − left) > k then
8:             Critical ← Critical ∪ {station}
9:         end if
10:        left ← station
11:        station ← right
12:    end for
13:    return Critical
14: end function
```

Given three consecutive charging stations along the path graph, in order to verify the middle *station* for being critical, it is crucial to observe the distance between the $left$ and $right$ neighbouring charging stations. If $right - left > k$, then it is impossible to get from $left$ to $right$ without charging in between, therefore *station* must be a critical charging station and is included in the set $Critical$.

The approach of Algorithm 1 is to consider indices 0 and ℓ as charging stations. This is because all cars start at v_0 and arrive at v_ℓ and this enables the verification of the first and last charging station in K. Every *station*, except v_0 and v_ℓ, is processed one by one and seen as the middle node between $left$ and $right$.

3.1.2 Schedule ordering influences program cost

In this part we explore the effect of reordering schedules inside the program sequence. The first theorem states the requirement to carefully order the schedules in a program P, in order to minimize $cost(P)$ for the given schedules. The second theorem however, shows that the order of the schedules doesn't influence the cost of independent programs.

Theorem 3.1.1. There is a program P with $cost(P)$, where it is possible to create program P' by reordering the schedules inside the sequence P, s.t. $cost(P') > cost(P)$.

Proof. By giving an example, it can be shown that the ordering of the schedules indeed influences the cost of the program. Using instance I with $K = \{2, 3, 4, 5\}$, $n = 4$, $k = 3$, and $\ell = 7$, there is

$$\begin{aligned} &P = (\{2,5\},\{2,5\},\{3,4\},\{3,5\}) \text{ with} \\ &cost(P) = 11 \text{ and} \\ &P' = (\{2,5\},\{2,5\},\{3,5\},\{3,4\}) \text{ with} \\ &cost(P') = 12. \end{aligned} \tag{3.1}$$

In both programs the first and second pair of cars are creating two units of waiting cost, one at charging station v_2 and one at v_3. The difference is the queuing at v_5. In P car a_4 is arriving at v_5 at $t = 7$ and only has to wait for a_2 to finish charging. In P' however, the car with the same schedule (this time a_3) arrives earlier at $t = 6$. This creates two units of waiting cost. One unit for a_3 having to wait for a_1, and a second unit for a_2 having to wait for a_3. □

Theorem 3.1.2. Given an independent program P. By creating independent program P' through reordering the schedules inside P, there is always $cost(P') = cost(P)$.

Proof. Clearly, for any reordering there is

$$C(P') = C(P). \tag{3.2}$$

Since every $S_i \in unique(P')$ retains the same number of cars assigned to itself, compared to P, we are ensured that

$$W(P') = W(P). \tag{3.3}$$

That is why $cost(P') = cost(P)$. □

3.1.3 Balancing leads to minimal waiting

For this subsection, we refer to Definition 2.0.13 and the associated A', K', and A'_i. The idea of Theorem 3.1.3, which is incorporated in later presented algorithms, is to show that the waiting cost can be reduced by balancing cars over charging stations.

Theorem 3.1.3. Considering the set X of programs where:

- Every car in A' has to charge at exactly one charging station in K'.
- All the cars in A'_i arrive at v_i at the exact same time.

Let $W'(P)$ denote the total waiting cost originating from all stations in K'. If A' is balanced over K', then the value of $W'(P)$ is a minimum among all programs in X.

Proof. Let there be $|K'|$ queues that can contain up to n cars each (n spots). Every spot in every queue has a position in its queue. The first spot in the queue has position 0, while the last position is $n-1$.

Assigning all A' cars to a spot in a queue is needed to get a value for $W'(P)$. Let Row_i denote the set of spots in all the queues that have position i in their queue. Also, let $rowCost(Row_i)$ denote the waiting cost associated to every spot in Row_i. It can be stated that

$$rowCost(Row_i) = i \text{ for } 0 \leq i \leq n-1. \tag{3.4}$$

It should be noted that $rowCost(Row_0) = 0$ because the cars associated to the spots in Row_0 can charge straight away.

It is obvious that a minimum for W' is obtained by iteratively filling the spots in each Row_i with $|K'|$ cars from $i = 0$ to $i = n$. Before filling Row_i, one must check whether

$$|A'| - |K'| \cdot (i-1) > |K'|. \tag{3.5}$$

If equation (3.5) does not hold, Row_i is the last row to be filled. In that case, in order to guarantee that A' is balanced over K', the remaining $(|A'| - |K'| \cdot (i-1))$ cars are assigned to spots in accordance to Definition 2.0.13.

If equation (3.5) does hold, we can proceed and fill Row_i with $|K'|$ cars. □

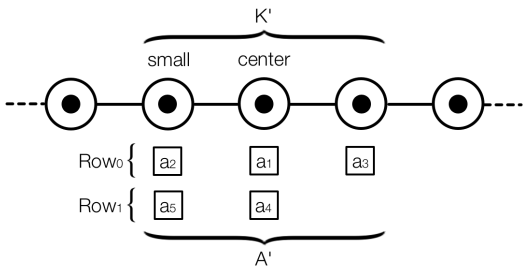

Figure 3.1: A' balanced over K'.

Figure 3.1 illustrates an example where a set of cars $A' = \{a_1, a_2, a_3, a_4, a_5\}$ is balanced over a set of charging stations K'.

3.2 Program Conversions

Next, we will focus on the properties *reduced*, *independent* and *serial*. In the following, given a program, we will analyze the effect of modifying the program in a way that it adapts one of the properties.

3.2.1 Conversion to reduced programs

By referring to Definition 2.0.8 and considering the next two theorems, we can conclude that, for a given instance I, there always exists at least one reduced program among all optimum programs.

Theorem 3.2.1. Given an instance I and a program P that is not reduced, P can be converted into reduced program P' s.t. $cost(P') \leq cost(P)$.

Proof. For every $S_i \in unique(P)$ that contains m, n, and o, s.t.

$$m, n, o \in S_i \cup \{0, \ell\},\ m < n < o, \text{ and } o - m \leq k, \tag{3.6}$$

the charging stop at node n can be removed from S_i. This is realizable since $o - m \leq k$. This is repeated until no m, n and o can be found in any $S_i \in unique(P)$. Following this procedure generates a reduced program P'. It remains to be shown that $cost(P') \leq cost(P)$.

Every time we remove a charging station v_n from S_i, let x denote the number of cars that are associated to S_i in P. An effect of the deletion of n is that $C(P') = C(P) - x$. Additionally we have $W(P') \leq W(P) + x$, because for every a_i among all x cars, there is

$$w_i(P') - w_i(P) \leq 1. \tag{3.7}$$

This is because in the worst case every car has to wait in the queue at o for every time step that it arrives earlier at o compared to P. This reasoning shows that every deletion of charging station v_n from S_i results in $cost(P') \leq C(P) - x + W(P) + x = cost(P)$. □

Theorem 3.2.2. Among all optimum programs for a given problem instance I, at least one program is reduced.

Proof. Given a set of optimum programs for I which does not yet contain a reduced program, we can arbitrarily select one program P in the set and refer to Theorem 3.2.1 in order to transform it into reduced program P' with $cost(P') = cost(P)$. □

3.2.2 Conversion to independent programs

In this thesis, mostly only independent programs are covered. Most significantly, all four main algorithms presented in chapters 4 to 7 return independent programs. This is why the exploration of the effects of modifying non independent programs and turning them into independent programs, seems essential. Starting with Theorem 3.2.3, we try to show that it is possible to convert very basic non independent programs, containing two unique schedules, without increasing the cost.

Theorem 3.2.3. Given a non independent program P with $unique(P) = 2$ schedules, it is possible to convert P into independent program P' s.t. $cost(P') \leq cost(P)$.

Proof. Given $S_1, S_2 \in unique(P)$, let $first$ denote the smallest node index in the set $S_1 \cap S_2$. Furthermore, we define the sets $Start_{S_i}$ and End_{S_i} in a way that

$$\begin{aligned} Start_{S_i} &= \{j \in S_i | j < first\} \text{ and} \\ End_{S_i} &= \{j \in S_i | j > first\}. \end{aligned} \tag{3.8}$$

Using this, we can construct a single new schedule S'_1 in a way that

$$\begin{aligned} x &= \underset{i \in [1..2]}{\operatorname{argmin}} |Start_{S_i}|, \\ y &= \underset{i \in [1..2]}{\operatorname{argmin}} |End_{S_i}|, \\ S'_1 &= Start_{S_x} \cup first \cup Start_{S_y}. \end{aligned} \tag{3.9}$$

All n cars are assigned to S'_1 in order to create independent P', s.t.

$$\begin{aligned} unique(P') &= \{S'_1\}, \\ P' &= (S'_1, S'_1, ..., S'_1). \end{aligned} \tag{3.10}$$

We can express the charging cost of P' by

$$|S'_1| \cdot n = C(P') \leq C(P). \tag{3.11}$$

The comparison to $C(P)$ holds because

$$|S'_1| \leq |S_1| \text{ and } |S'_1| \leq |S_2|. \tag{3.12}$$

Since charging stations can only be left by one car at given time step, waiting cost in P' only originates from the first station in S'_1. For a single station, this waiting cost is however maximal, due to the fact that all n cars arrive there simultaneously. From there on, all cars arrive at the subsequent stops with a one time unit delay between each other.

Let $W_{first}(P)$ denote the waiting cost that originates from v_{first} only in P. There is

$$W(P') - W_{first}(P) = z \geq 0. \tag{3.13}$$

However we can state that every unit in the difference of z is saved by arriving one time step later at v_{first} in P, compared to P', due to an additional charging stop before v_{first}. That is why every gain in waiting cost is canceled out by a loss in charging cost, which makes it impossible that $cost(P') > cost(P)$. □

For more complex programs with a higher number of schedules, a way to show the relations among those schedules is needed. Given a non independent set of schedules Y, we can create a graph denoted F, showing the interferences of the schedules. Nodes in F represent schedules and an undirected edge connects schedules S_i and S_j if it does not hold that

$$S_i \cap S_j = \emptyset. \tag{3.14}$$

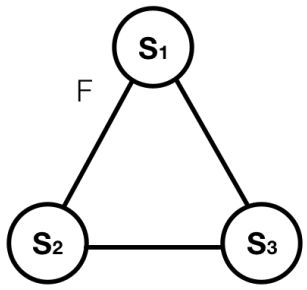

Figure 3.2: Example graph F for schedules S_1, S_2, and S_3.

The graph in Figure 3.2 shows an example where all schedules are fully connected, i.e. every schedule pair shares at least one common charging station.

Theorem 3.2.4. Given a non independent program P and the related graph F. If F is disconnected and every disconnected component contains a maximum of two nodes, it is possible to convert P into independent program P' s.t. $cost(P') \leq cost(P)$.

Proof. Looking back at the previous Theorem 3.2.3, it should be pointed out that the newly created schedule S'_1 does not use any charging stations that are not in S_1 or S_2. That is why, given F, we can apply the same modifications to every disconnected component with two nodes. The resulting P' will be independent and we are ensured that $cost(P') \leq cost(P)$. □

An idea, in order to perform conversion on components in F having more than two nodes, is to assign all cars to the schedules in the maximum independent set of the component only. However, Theorem 3.2.5 shows that this might result in higher program cost. At least we can claim this for one instance where F is fully connected.

Theorem 3.2.5. Given a non independent program P that has been computed for instance I. We are also given the graph F related to P, which is fully connected. Creating independent program P' by assigning all n cars the

schedule S_{greedy}, s.t. $cost(P') \leq cost(P)$, is not always possible.

Proof. To show this we use an instance as depicted on Figure 3.3 and a program which, when modeled as a graph, leads to Figure 3.2.

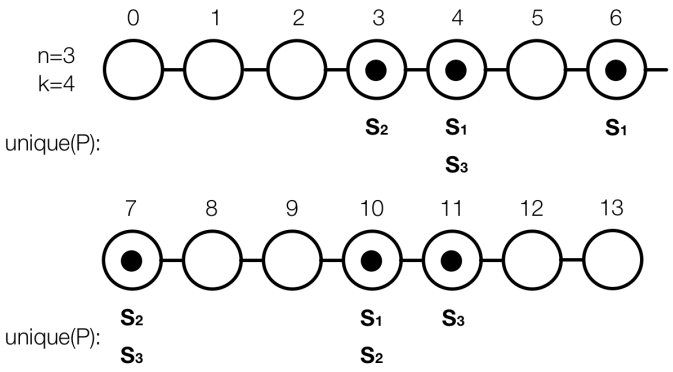

Figure 3.3: Fully connected schedules in $unique(P)$.

Program $P = (\{4, 6, 10\}, \{3, 7, 10\}, \{4, 7, 11\})$ has cost

$$\begin{aligned} cost(P) &= C(P) + W(P) \\ &= 9 + 2 \\ &= 11. \end{aligned} \tag{3.15}$$

Please note that no waiting cost is originating from v_7. Car a_3 (using S_3) arrives one time step later at v_7 than a_2. This is because a_3 had to wait for a_1 to finish charging at v_4.

Assigning S_{greedy} to every car results in $P' = (\{4, 8, 12\}, \{4, 8, 12\}, \{4, 8, 12\})$ with cost

$$\begin{aligned} cost(P') &= C(P') + W(P') \\ &= 9 + 3 \\ &= 12. \end{aligned} \tag{3.16}$$

We obtain $12 = cost(P') > cost(P) = 11$. □

Unfortunately, even when limiting the conversion procedure to path graphs, no evidence of a way to obtain lower or equal cost for the resulting independent program has been found during the research of this thesis. However, for every studied problem instance, we were always able to compute at least one independent program with lower or equal cost, compared to the best non independent program that we found. This is why we very carefully state that, based on intuition and lack of counterexample, it seems that EfficientCharging always incorporates an independent program as a solution. Thus we use the following conjecture to express a possible aspect of future research.

Conjecture 3.2.1. Given an instance I, there exists at least one optimum independent program P.

3.2.3 Conversion to serial programs

Lastly, we explore the conversion to serial programs. In this subsection, three theorems are presented which all aim to show the possibilities in terms of converting programs s.t. they satisfy Definition 2.0.11.

Theorem 3.2.6. Given an independent and proportional program P, it is possible to convert P into serial program P' s.t. $cost(P') = cost(P)$.

Proof. We denote X the sequence consisting of all charging stations in P, ordered increasingly. In order to create serial P', we reassign every charging station in X to a new schedule $S'_i \in P'$, in a way that the jth charging station in X is assigned to schedule $S'_{s-((j-1) \bmod s)}$ (round robin assignment).

To show that every $S'_i \in P'$ is still feasible, we consider every two consecutive charging stations

$$b, c \in (0 \cup S'_i \cup \ell), \text{ s.t. } b < c \tag{3.17}$$

and try to argue that $c - b \leq k$. This can be done by stating that there can be at most $s-1$ charging stations in between b and c in the sequence $0 \cup X \cup \ell$. The reason for this is the round robin method used to reassign charging stations.

Let $S_x \in P$ denote one schedule that has no charging station between b and c in P. Also, we define $first$ and $last$ to be the first and last charging station in S_x. There will always be

$$\begin{aligned} &\bullet\ a, d \in S_x \text{ s.t. } a \leq b < c \leq d, \text{ or} \\ &\bullet\ last \in S_x < b, \text{ or} \\ &\bullet\ first \in S_x > c. \end{aligned} \tag{3.18}$$

Every case states that $S_x \in P$ shows that $c-b \leq k$. Either it is possible in P to get from a to b, from $last$ to ℓ or from 0 to $first$. In every case the distance $c-b$ is smaller, thus also smaller or equal than k. Reassigning the charging stations ensures that $cost(P') = cost(P)$. This is because every schedule S'_i retains the same amount of charging stations and associated cars than schedule S_i. □

For the next theorem, a few adjustments are made. On one hand, the constraint requiring P to be proportional is lifted, on the other hand $uniqueStations(P')$ is considered instead of $cost(P')$.

Theorem 3.2.7. Given an independent program P, it is possible to convert P into serial P' s.t. $|uniqueStations(P')| = |uniqueStations(P)|$.

Proof. For this proof, we refer to the above Theorem 3.2.6 and the sequence denoted X. Since P is independent, there is

$$|X| = |uniqueStations(P)|. \tag{3.19}$$

Theorem 3.2.6 shows that we can reassign every charging station in X to a new schedule $S_i' \in P'$ s.t. P' is serial. By doing this we clearly ensure that

$$|uniqueStations(P')| = |uniqueStations(P)|. \quad (3.20)$$

□

However, it turns out that serial conversion is not always possible, especially when a lower or equal cost is desired as a result.

Theorem 3.2.8. There exists an instance $I \in \mathcal{I}^{cb}$ and program P s.t.:

- P is independent, non serial, and a program for I.
- P cannot be converted into serial program P' s.t. $cost(P') \leq cost(P)$.

Proof. All conditions are satisfied when using:

- Instance I:
  $K = \{2, 3, 4, 6, 7, 8, 9, 11, 12, 13\}$,
  $n = 5$,
  $k = 3$,
  $\ell = 15$.
- Program P:
  $unique(P) = \{\{3, 6, 9, 12\}, \{2, 4, 7, 8, 11, 13\}\}$,
  $S_1 = S_2 = S_3 = \{3, 6, 9, 12\}$,
  $S_4 = S_5 = \{2, 4, 7, 8, 11, 13\}$,
  $P = (S_1, S_2, S_3, S_4, S_5)$.

We can express the cost of P as

$$\begin{aligned} cost(P) &= C(P) + W(P) \\ &= 24 + 4 \\ &= 28. \end{aligned} \quad (3.21)$$

When creating serial P', the value of s must either be 1 or 2 in order for P' to be independent (part of the serial Definition 2.0.11). Having 3 or more schedules in an independent program for I is not possible. A precise way of showing this will be shown with Theorem 4.1.1 in Chapter 4.

s=1

Clearly, in this case the serial P' with lowest cost is

$$\begin{aligned} &S_{greedy} = \{3, 6, 9, 12\} = S_1 = S_2 = S_3 = S_4 = S_5, \\ &P' = (S_1, S_2, S_3, S_4, S_5). \end{aligned} \quad (3.22)$$

We express the cost of P' as

$$\begin{aligned} cost(P') &= C(P') + W(P') \\ &= 20 + 10 \\ &= 30. \end{aligned} \quad (3.23)$$

s=2

Since P' has to be serial, there is only one possibility for the set $unique(P')$, which is

$$unique(P') = \{\{3, 6, 8, 11, 13\}, \{2, 4, 7, 9, 12\}\}. \quad (3.24)$$

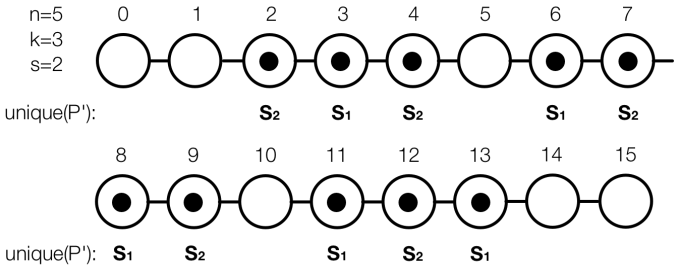

Figure 3.4: Visualization of only possibility for the schedules in $unique(P')$ for I with $s = 2$.

Figure 3.4 displays the only possible 2 schedules in serial $unique(P')$, obviously considering that $k = 3$. The reasoning is as follows:

- Both schedules need to charge at either v_2 or v_3 as their first stop. Because of the serial Definition 2.0.11, S_1 has to to charge at a station with a higher index than S_2, which has to be v_3.
- Schedule S_1 can now choose between v_4 and v_6 for the next stop. Only v_6 preserves the serial constraint however, that is why we are left with v_4 to be used by S_2.
- Schedule S_2 is now forced to charge at v_7 next, since it cannot reach v_8. Moreover, S_1 needs to charge at v_8 because v_9 would block S_2 from reaching S_{11} or later charging stations.
- In order to reach v_{11} or later charging stations, S_2 has to charge at v_9. Only option for the next charging stop for S_1 is v_{11}.
- Finally, v_{12} is the only option for the next stop of S_2. Only remaining charging station for S_1, which is needed to reach v_{15} at the same time, is v_{13}.

When it comes to the assignment of cars, we try to reduce $W(P')$ as much as possible, knowing that $C(P')$ is already fixed at $C(P') = 5 \cdot 5 = 25$. Since waiting cost originates from the first charging station in every schedule in $unique(P')$ only, we can refer to Theorem 3.1.3 with $K' = \{2, 3\}$ and $A' = \{a_1, a_2, a_3, a_4, a_5\}$ to achieve minimum $W(P')$. We obtain

$$\begin{aligned} &S_1 = S_2 = S_3 = \{3, 6, 8, 11, 13\}, \\ &S_4 = S_5 = \{2, 4, 7, 9, 12\}, \\ &P' = (S_1, S_2, S_3, S_4, S_5). \end{aligned} \quad (3.25)$$

Evaluating the cost of P' leads to

$$\begin{aligned} cost(P') &= C(P') + W(P') \\ &= 25 + 4 \\ &= 29. \end{aligned} \tag{3.26}$$

Considering both cases, $s = 1$ and $s = 2$, we can conclude that a serial P' will always have $cost(P') > 28 = cost(P)$ for instance I. □

Chapter 4

Critical Blocks - First Algorithm

Now that a few first observations and theorems have been covered, we will proceed with the algorithmic part of the thesis. In the context of an internship [2], the problem of EfficientCharging has already been studied for problem instances where $K = V \setminus \{\ell, \ell-1, ..., \ell-x\}$, with x bound by

$$0 \leq x \leq k. \tag{4.1}$$

A next step to approach more generic instances is the set of critical block instances $\mathcal{I}^{cb}$. That is why this chapter is devoted to develop a first algorithm, targeted to solve every instance $I \in \mathcal{I}^{cb}$. Please note that the notations used in Definition 2.0.14 will be applied in this chapter.

The main idea of FirstAlg is to reduce the queuing of the cars by distributing them over a maximum amount of independent schedules. The algorithm returns an independent program P_{first} that uses an upper bound for $s = |unique(P_{first})|$.

Definition 4.0.1 (Value of i^*). Given K, k, and ℓ, let Y_{i^*} denote the set consisting of a maximum amount of independent schedules. We denote

$$i^* = |Y_{i^*}|. \tag{4.2}$$

Since P_{first} uses an upper bound for s, it follows that $s = i^*$.

4.1 Computing i^*

In the following we will present a method to compute the value of i^* for any $I \in \mathcal{I}^{cb}$. To start, a brief definition and a linked observation is needed.

Let gap_m denote the number of nodes between block B_m and block B_{m+1}, i.e.

$$gap_m = start(B_{m+1}) - end(B_m) - 1. \tag{4.3}$$

The length of the arriving and leaving zone of any block B_m can be computed by exploiting

$$\begin{aligned} &length(arrivingZone(B_m)) \\ &= min(length(B_m), k - gap_{m-1}) \text{ and} \\ &length(leavingZone(B_m)) \\ &= min(length(B_m), k - gap_m). \end{aligned} \tag{4.4}$$

Algorithm 2 uses this approach and returns the start and end index of every zone after having calculated its length.

Algorithm 2 Computing the start and end of all zones

```
1: function GetZones(K, k, ℓ)
2:     counter ← 0
3:     inBlock ← false
4:     for i in [1..ℓ−1] do
5:         if i∈K != inBlock then
6:             inBlock ←!inBlock
7:             L.append ← counter
8:             counter ← 0
9:         end if
10:        counter ← counter + 1
11:    end for
12:    L.append ← counter
13:    if inBlock then
14:        L.append ← 0
15:    end if
16:    j ← L[0]
17:    for i in [1..|L|−1] do
18:        if i mod 2 == 1 then
19:            lengthA ← min(L[i], k − L[i − 1])
20:            arriving ← [j + 1, j + lengthA]
21:            j ← j + L[i]
22:            lengthL ← min(L[i], k − L[i + 1])
23:            leaving ← [j − lengthL + 1, j]
24:            Zones.append ← [arriving, leaving]
25:        else
26:            j ← j + L[i]
27:        end if
28:    end for
29:    return Zones
30: end function
```

In the above procedure, a first step is to generate the list L, which follows the structure

$$L = (gap_0, length(B_1), gap_1, ..., length(B_b), gap_b). \tag{4.5}$$

We should note that

$$\begin{aligned} 1 \in K \text{ implies } gap_0 = 0 \text{ and} \\ \ell - 1 \in K \text{ implies } gap_b = 0. \end{aligned} \tag{4.6}$$

Based on this, a three dimensional array *Zones*, storing all start and end indices of all zones, is returned. The first dimension refers to the different blocks, the second dimension differentiates between arriving ("0") and leaving zones ("1"), and the third dimension references either the start ("0") or the end ("1") index.

Algorithm 3 can deduct the upper bound for the

number of independent schedules by using the array information and returning the length of the shortest zone, thus providing a value for i^*.

Algorithm 3 Returning the length of the smallest zone

```
1:  function ShortestZone(Z)
2:      min ← ℓ
3:      b ← |Z|
4:      for i in [0..b−1] do
5:          for j in [0..1] do
6:              if Z[i][j][1] − Z[i][j][0] < min then
7:                  min ← Z[i][j][1] − Z[i][j][0]
8:              end if
9:          end for
10:     end for
11:     return min
12: end function
```

Theorem 4.1.1. Algorithm 3 returns the value of i^* for any $I \in \mathcal{I}^{cb}$.

Proof. Among all arriving and leaving zones, Algorithm 3 returns the length of the zone that contains the fewest charging stations. Let *shortestZone* denote this zone and let d be defined by

$$d = length(shortestZone). \tag{4.7}$$

We can easily reason that it is impossible to have a set of $d+1$ independent schedules, since every schedule has to charge in every zone, as well as in *shortestZone*, at least once. Having $d+1$ schedules would imply having at least one pair of schedules that has at least one charging station in common.

However, a set of d independent schedules is always possible to exist. Let $S_1, S_2, ..., S_d$ denote a set of independent schedules. Since $length(arrivingZone(B_m)) \geq d$, it is possible for the schedules to arrive in B_1. From here on and for every block B_m, every schedule traverses k edges (Figure 4.1) which leads to retention of the schedule order. To make it easier to refer to the schedules in B_m, we denote S_1 the schedule who's first charging stop in B_m has the highest node index among all schedules. S_1 is followed by S_2, S_3, ..., S_d (same as on Figure 4.1).

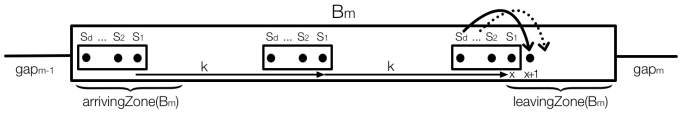

Figure 4.1: Possible behaviour of d independent schedules inside a critical block.

This is repeated until a subset of the schedules reaches $leavingZone(B_m)$ (on Figure 4.1 only S_1). Please note that it is certainly possible that all schedules pass, i.e. charge before and after, $leavingZone(B_m)$, due to k being too large compared to $length(arrivingZone(B_m))$. This case can be solved by backshifting every schedule by the same amount of node indices, s.t. S_1 charges at index $end(leavingZone(B_m))$.

For the general case however, let x denote the index of the charging station that S_1 arrives at in $leavingZone(B_m)$. We can assign the remaining schedules, that are not part of the subset reaching the leaving zone, to charging stations in $leavingZone(B_m)$ in the following way:

$$\begin{aligned} S_d &\rightarrow x+1 \\ S_{d-1} &\rightarrow x+2 \\ &\vdots \end{aligned}$$

Those assignments are feasible since we know from 4.4 that $length(arrivingZone(B_m)) \leq k$, which implies that $d \leq k$. Furthermore, the assignment method shows that every S_i, that didn't reach $leavingZone(B_m)$ with k edge traversals each, has been shifted by exactly d indices.

Knowing that $length(leavingZone(B_m)) \geq d$, the schedules can leave B_m in order to arrive in the next block B_{m+1}.

We have shown that $d = length(shortestZone)$ is the upper bound for the number of independent schedules and thus provides the value for i^* for any $I \in \mathcal{I}^{cb}$. □

4.2 Algorithm Approach

Having established a way to compute the value of i^* for any $I \in \mathcal{I}^{cb}$, it is time to dive into the details of the algorithmic approach.

4.2.1 Single independent schedule

Although FirstAlg is designed for multiple independent schedules (Subsection 4.2.2), it is worth discussing the occurrence of a single independent schedule ($i^* = 1$).

Here the situation arises where at least one zone of some block consists of a single charging station. The consequence is that the node in question is a critical charging station (Definition 2.0.6). The problem can be divided into two subproblems:

- Subproblem 1 denoting the cost needed to arrive, wait, and charge at the first critical charging station.
- Subproblem 2 denoting the cost needed to drive from the first critical charging station to v_ℓ.

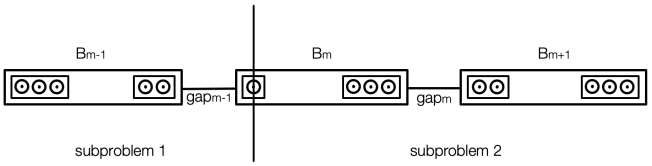

Figure 4.2: Arriving zone of B_m consists of a single critical charging station and serves as the splitting point of two subproblems.

Referring to Figure 4.2, both subproblems can be solved optimally by assigning S_{greedy} to all cars. Because charging stations can only be left by one car at a given time step, we are guaranteed that cars arrive as early as possible at subproblem 2 (since every car needs to stop at the critical charging station), which can be accomplished without queuing at all (due to the consecutive time steps at which cars enter subproblem 2) and with a minimum amount of recharges for every car (due to S_{greedy}).

This shows that a trivial solution for roads with at least one critical charging station is the program $P = (S_{greedy}, S_{greedy}, ...S_{greedy})$. This even generalizes to instances where not all blocks are critical. As long as there is at least one critical charging station, which constrains the number of independent schedules to a maximum of one, minimal cost is attained by using this approach.

4.2.2 Multiple independent schedules

In general (for $i^* \geq 1$), let us discuss in the following the computation of i^* independent schedules. For this purpose, let S_i denote the ith independent schedule with i being bound by

$$1 \leq i \leq i^*. \tag{4.8}$$

The program uses all i^* independent schedules, which are generated by Algorithm 4. The method employs an array named *Road* that contains an entry for every node in the path graph, except for v_0 and v_ℓ. Let e be the entry for v_j. The value e underlies the rule that states

$$\begin{aligned} &e = 0 \text{ if no schedule stops at } v_j \text{ and} \\ &e = i \text{ if } S_i \text{ stops at } v_j. \end{aligned} \tag{4.9}$$

A second array *Last* stores the current last stop of every schedule during the execution of the procedure. *Last* is initialized with

$$\begin{aligned} &Last[i-1] \\ =\ &min(k-(i-1), end(leavingZone(B_1))-(i-1)). \end{aligned} \tag{4.10}$$

After initialization, every schedule attempts to traverse k edges (from the last stop of the schedule) before recharging. If traversing k edges does not lead to a charging station, the algorithm follows a "fixing" procedure that we describe right after Algorithm 4.

Algorithm 4 Computing all independent schedules

```
1: function ComputeSchedules(K, k, ℓ)
2:     Z ← GetZones(K, k, ℓ)
3:     i* ← ShortestZone(Z)
4:     b ← |Z|
5:     for v in [1..ℓ−1] do
6:         Road[v] ← 0
7:     end for
8:     for i in [1..i*] do
9:         Last[i−1] ← min(k−(i−1), Z[0][1][1]−(i−1))
10:    end for
11:    for m in [1..b] do
12:        for i in [1..i*] do
13:            while Last[i − 1] < Z[m − 1][1][0] do
14:                new ← Last[i − 1] + k
15:                Last[i − 1] ← new
16:                if new < ℓ then
17:                    Road[new] ← i
18:                end if
19:            end while
20:        end for
21:        Shift(Road, Z[m−1][1][0], Z[m−1][1][1])
22:    end for
23:    return ExtractSchedules(Road)
24: end function
```

Since not every node is a charging station, it may happen that an attempt to traverse k edges may not be possible and leads to a non charging station. That is why, from time to time, such an attempted last charging stop needs to be corrected, and shifted to a node with a lower index, being a charging station in the leaving zone of the block. That is the purpose of passing a reference of *Road* when calling Algorithm 5.

Algorithm 5 Shifting every schedule if necessary

```
1: function Shift(Road, start, end)
2:     for v in [ℓ..end+1] do
3:         if Road[v] > 0 then
4:             Shift.append ← Road[v]
5:             Road[v] ← 0
6:         end if
7:     end for
8:     for v in [end..start] do
9:         if Road[v] > 0 and |Shift| > 0 then
10:            Shift.append ← Road[v]
11:        end if
12:        if |Shift| > 0 then
13:            Road[v] ← Shift[0]
14:            Shift[0].remove
15:        end if
16:    end for
17: end function
```

Informally, the above procedure operates by iterating over the nodes using a decreasing index. If the node index of the last stop of a schedule in some B_m exceeds $end(B_m)$, it is backshifted into $leavingZone(B_m)$. No charging stop is backshifted more than necessary. Schedules that do not need any adjustments (i.e. are already in $leavingZone(B_m)$) are moved to a charging station with a smaller index in case it is necessary to "free up" stations with a higher index. Every schedule keeps its position concerning how "late" it charges in B_m compared to the other schedules. All of this is done with the help of the queue $Shift$, which operates in a "first-in, first-out" manner.

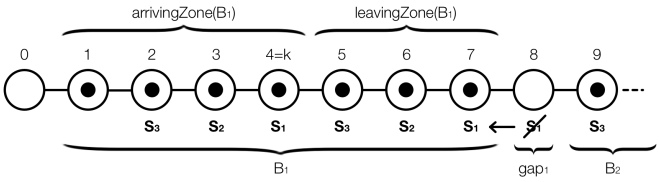

Figure 4.3: Example instance with $i^* = 3$. S_1 is shifted from v_8 to v_7 and pushes S_2 and S_3 to v_6 and v_5 respectively.

On a side note, it can be pointed out that, because of the way Algorithm 5 shifts the schedules, their ordering along the path graph is preserved and P_{first} is actually a serial program.

The next step consists of using Algorithm 6 to extract all i^* schedules from $Road$ and returning them as an array of sets of charging stations.

Algorithm 6 Extracting all independent schedules

```
1: function ExtractSchedules(Road)
2:     for v in [1..ℓ−1] do
3:         if Road[v] > 0 then
4:             Y[Road[v]−1] ← Y[Road[v]−1] ∪ {v}
5:         end if
6:     end for
7:     return Y
8: end function
```

Finally, to put it all together, every car is assigned a schedule as defined in Algorithm 7. The procedure balances the set of all cars A over the set

$$\begin{aligned} FirstStops = \{ & \\ & min(k, end(B_1)), \\ & min(k-1, end(B_1)-1), \\ & ..., \\ & min(k-(i^*-1), end(B_1)-(i^*-1)) \\ \} & \end{aligned} \tag{4.11}$$

in order to achieve a minimum for $W(P_{first})$ for the given number of $s = i^*$ schedules (Theorem 3.1.3).

Algorithm 7 Implementation of FirstAlg

```
1: function FirstAlgorithm(I ∈ 𝓘^cb)
2:     Y ← ComputeSchedules(K, k, ℓ)
3:     for i in [1..n] do
4:         S_i ← Y[(i−1) mod |Y|]
5:     end for
6:     return P_first = (S_1, S_2, ..., S_n)
7: end function
```

4.3 Performance Analysis

Concerning the performance of FirstAlg, we will shortly comment on minor observations related to the charging cost of the schedules and the total waiting cost. This is followed by the evaluation of the algorithm on a specific instance.

4.3.1 Analyzing schedule charging cost

To be able to analyze the charging cost of the schedules in P_{first}, we use the following notation:

$$\begin{aligned} c_{S_i} &= \text{number of stops of } S_i \in unique(P_{first}), \\ c_{S_i B_m} &= \text{number of stops of } S_i \in unique(P_{first}) \text{ in } B_m, \\ c_{optB_m} &= \text{number of stops of } S_{greedy} \text{ in } B_m. \end{aligned} \tag{4.12}$$

Theorem 4.3.1. In the context of P_{first}, computed by FirstAlg when applied to any instance $I \in \mathcal{I}^{cb}$, it holds that $\forall i$ s.t. $1 \le i \le i^* : 0 \le c_{S_i} - c_{S_1} \le 1$.

Proof. By considering the node indices of the charging stations that are used by all the schedules in $unique(P_{first})$ to charge in $arrivingZone(B_m)$, a position can be associated to every $S_i \in unique(P_{first})$ for B_m. Let $pos(S_i, B_m)$ denote the position of S_i in B_m. Also, let S_i stop at the charging station with the jth highest node index among all charging stations in B_m that are actually used by one schedule in $unique(P_{first})$. The position of S_i for B_m is defined using the value of j, s.t.

$$\forall S_i \in P : pos(S_i, B_m) = j. \tag{4.13}$$

An example illustrating the positions for all schedules in B_m is shown with Figure 4.4.

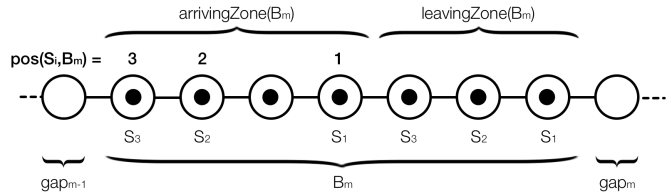

Figure 4.4: Example position association for every S_i in B_m.

From (4.4) it is deductible that the size of any zone cannot exceed k. A first observation is that

$$\forall i, m : c_{S_i B_m} = c_{optB_m} \text{ or } c_{S_i B_m} = c_{optB_m} + 1. \quad (4.14)$$

To show that no schedule charges more than c_{S_1}+1 times, let there be schedules S_i and S_j. Using (4.14), there is for B_m

$$\begin{aligned} &pos(S_i, B_m) < pos(S_j, B_m) \\ &\text{implies } 0 \leq c_{S_j B_m} - c_{S_i B_m} \leq 1. \end{aligned} \quad (4.15)$$

Given a block B_m, let $O \subseteq unique(P_{first})$ denote the subset of the schedules that charge c_{optB_m} times in B_m and let $O' \subseteq unique(P_{first})$ denote the subset of the schedules that charge c_{optB_m}+1 times in B_m. There is for B_m and B_{m+1}

$$\begin{aligned} &\forall S_i \in O \text{ and } \forall S_j \in O' : \\ &pos(S_i, B_m) < pos(S_j, B_m) \\ &\text{implies } pos(S_i, B_{m+1}) > pos(S_j, B_{m+1}). \end{aligned} \quad (4.16)$$

Commenting on (4.15) and (4.16), it can be said that a schedule, that is associated to a smaller position in B_m, charges less or an equal amount of times in B_m, compared to a schedule with a bigger position. Also, having charged $c_{optB_m} + 1$ times in B_m means being associated a smaller position in B_{m+1}, compared to all schedules that charged c_{optB_m} times in B_m. It can be stated in a more informal way that if a schedule "wins" in B_m compared to some other schedule, it does not "win" again in B_{m+1} compared to this schedule. It will "loose", or "draw" in the best case.

So it can be stated that

$$\begin{aligned} &\forall i \text{ s.t. } 1 \leq i \leq i^* : \\ &c_{opt} \leq c_{S_1} \leq c_{S_i} \text{ and} \\ &0 \leq c_{S_i} - c_{S_1} \leq 1. \end{aligned} \quad (4.17)$$

□

The explanations of Theorem 4.3.1 illustrate what can be deducted from the fact that P_{first} is serial in a more detailed way and specifically applied to FirstAlg.

4.3.2 Analyzing total waiting cost

When analyzing the total waiting cost of P_{first}, a first observation is that

$$n \leq i^* \text{ implies } W(P_{first}) = 0. \quad (4.18)$$

Looking at the case where $i^* \leq n$, it can be stated that FirstAlg balances the set A over $FirstStops$, defined with (4.11). Referring to Theorem 3.1.3, this means that the waiting cost originating from the charging stations in $FirstStops$ is a minimum. Also, it can be observed that no additional waiting cost is originating from all subsequent charging stations.

The only way to reduce $W(P_{first})$ is by balancing A over a set of charging stations with a higher cardinality than $|FirstStops| = i^*$. This would naturally lead to a non independent program. Although this reduces the waiting cost originating from $FirstStops$, it is likely that cars will have to queue again when arriving at the shortest zone at the latest. As already mention in relation to Conjecture 3.2.1, the study of non independent programs are not part of the scope of this thesis. That is why we are unfortunately unable to comment on the effects of increasing the cardinality of $FirstStops$.

4.3.3 Disproving instance

Next, we want to mention that an instance that disproves optimality for P_{first} has been found during research and is presented hereafter.

Theorem 4.3.2. P_{first} computed by FirstAlg is not optimum for every $I \in \mathcal{I}^{cb}$

Proof. To show this we construct an instance I by using a road layout as depicted on Figure 4.5, $n = 3$ cars, and $k = 4$. This produces a smallest zone of $i^* = 3$. The three resulting schedules in $unique(P_{first})$ look as follows.

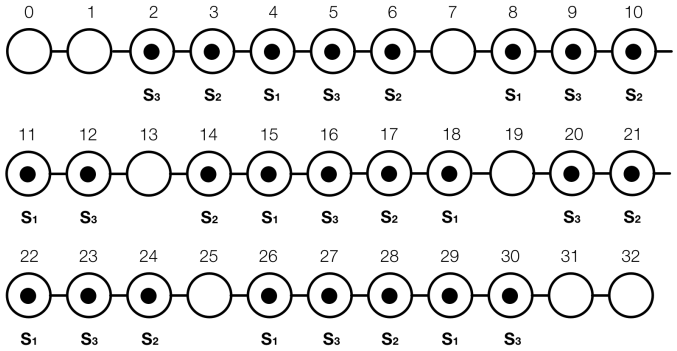

Figure 4.5: Showing the charging stops for S_1, S_2 and S_3.

By following the algorithm and assigning every car to one of the three schedules, this gives us

$$\begin{aligned} &P_{first} = (\\ &\{4, 8, 11, 15, 18, 22, 26, 29\}, \\ &\{3, 6, 10, 14, 17, 21, 24, 28\}, \\ &\{2, 5, 9, 12, 16, 20, 23, 27, 30\}). \end{aligned} \quad (4.19)$$

When calculating the cost, we obtain

$$\begin{aligned} cost(P_{first}) &= C(P_{first}) + W(P_{first}) \\ &= (8 + 8 + 9) + 0 \\ &= 25. \end{aligned} \quad (4.20)$$

However, lower cost can be achieved by using S_{greedy} for all three cars (Figure 4.6).

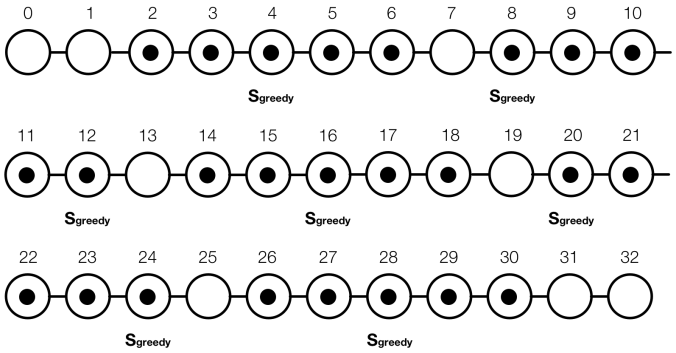

Figure 4.6: Showing the charging stops for S_{greedy}.

We obtain the program

$$\begin{aligned} &P_{greedy} = (\\ &\{4, 8, 12, 16, 20, 24, 28\}, \\ &\{4, 8, 12, 16, 20, 24, 28\}, \\ &\{4, 8, 12, 16, 20, 24, 28\}) \end{aligned} \tag{4.21}$$

and the related cost

$$\begin{aligned} cost(P_{greedy}) &= C(P_{greedy}) + W(P_{greedy}) \\ &= (7 + 7 + 7) + (0 + 1 + 2) \\ &= 24 < cost(P_{first}). \end{aligned} \tag{4.22}$$

□

Chapter 5

Critical Blocks - Second Algorithm

Since FirstAlg turned out to be non optimum, a new improved algorithm will be presented in this chapter. Inspired by the instance presented in Theorem 4.3.2, SecondAlg tries to address the main weakness of FirstAlg, which seems to be the fixed number of independent schedules. That is why the new approach is designed to be more flexible in terms of the used value for s, denoting the number of unique schedules in a program. The task however, remains the same, which means that SecondAlg is still designed for trying to solve instances that are elements of the set $\mathcal{I}^{cb}$.

A first hurdle, required for the proper functioning of the algorithm, is to find a way to compute j^*, the maximum number of independent schedules which all charge a minimum amount of times.

Definition 5.0.1 (Value of j^*). Given K, k, and ℓ, let Y_{j^*} denote the set consisting of a maximum amount of independent schedules that all charge c_{opt} times. We define j^* as

$$j^* = |Y_{j^*}|. \tag{5.1}$$

It should be noted that j^* is not limited to critical blocks instances.

Since j^* is defined with an additional constraint (compared to i^*) which requires that every schedule has to charge c_{opt} times, it can be stated that

$$j^* \leq i^*. \tag{5.2}$$

5.1 Computing j^*

In the following, two procedures will be presented that, when used together, allow the computation of j^* for any instance I.

Algorithm 8 Computes greedy schedule that doesn't charge in X

```
1: function Greedy(K, X, k, ℓ)
2:     S_i ← ∅
3:     station ← 0
4:     while station < ℓ − k do
5:         next ← station + k
6:         while next ∈ X or next ∉ K do
7:             next ← next − 1
8:             if next == station then
9:                 return ∅
10:            end if
11:        end while
12:        S_i ← S_i ∪ {next}
13:        station ← next
14:    end while
15:    return S
16: end function
```

Algorithm 8 returns a greedy schedule that makes charging stops after a maximum amount of edge traversals. However, an additional constraint prevents the schedule to use any charging stations in the set X.

Algorithm 9 Computes set with j^* schedules that all charge c_{opt} times

```
1: function GetJStar(K, k, ℓ)
2:     Y ← ∅
3:     X ← ∅
4:     S_i ← Greedy(K, X, k, ℓ)
5:     X ← X ∪ S_i
6:     opt ← |S_i|
7:     while |S_i| == opt do
8:         Y ← Y ∪ S
9:         S_i ← Greedy(K, X, k, ℓ)
10:        X ← X ∪ S_i
11:    end while
12:    return Y
13: end function
```

Algorithm 9 computes greedy independent schedules by iteratively calling Algorithm 8 and enhancing the set X by the charging stations of the just computed schedule by Algorithm 8. The procedure stops when a schedule, that charges more than c_{opt} times, has been found. This schedule is discarded and not included in the final returned set. Another stopping criterion is the event in which charging stations in X prevent the existence of an additional schedule.

The purpose of the following two theorems is to show that the cardinality of the returned set of schedules, returned by Algorithm 9, indeed represents the value of j^*.

Theorem 5.1.1. The set of schedules returned by Algorithm 9 is serial.

Proof. Let Y_{alg} denote the set returned by Algorithm 9. Every $S_i \in Y_{alg}$ is computed entirely before the start of the computation of S_{i+1}. By being greedy, every schedule charges after a maximum of edge traversals without using a charging station that has already been used by a schedule with a lower index. Thus it can be concluded that Y_{alg} is independent.

We denote

- m the jth charging stop of S_{i-1},
- n the jth charging stop of S_i, and
- o the jth charging stop of S_{i+1}. (5.3)

Because of the greediness of S_{i-1}, it is not possible that $n > m$. This implies that $n < m$ (independent Y_{alg} prevents $n = m$). Also, it can't be the case that $n < o < m$. This is again because of the greediness of S_i, leaving no charging station unoccupied that is between m and n and feasible to reach. Thus S_i enforces $o < n < m$.

This shows that the order of every $S_i \in Y_{alg}$, when considering a given charging stop, is always preserved and the schedules keep on alternating. This clearly makes the set Y_{alg} serial. □

Theorem 5.1.2. The cardinality of Y_{alg} equals j^*.

Proof. In Theorem 3.2.6 we have established that every independent and proportional program can be converted into a serial program without modifying its cost. Thus it can be deducted that at least one serial program, containing j^* schedules which all charge c_{opt} times, exists.

In Theorem 5.1.1 we have established that Y_{alg} is serial. By design, Algorithm 9 only appends schedules to Y_{alg} in case they charge c_{opt} times. This leaves us with the task to show that Y_{alg} contains a maximum number of schedules, a requirement needed for the cardinality of Y_{alg} to serve as the value for j^*.

Let Y_{j^*} denote a serial set of schedules containing j^* schedules that all charge c_{opt} times. We can adjust Y_{j^*} by applying the following modification to every jth charging stop, with j starting at 1 and ending at c_{opt}: In the sequence of considering every i from 1 to j^*, we modify the jth charging stop of $S_i \in Y_{j^*}$. The charging stop is modified to be at the charging station with the highest index possible (s.t. the schedule is still feasible with respect to the $(j-1)$th charging stop) that is not yet used by some other schedule with a lower index for the same jth stop. We define max to be this possibly new node index. We are ensured that:

- No other schedule S_h uses max as the $(j+1)$th charging stop.
  If this was the case, S_h would charge $c_{opt}+1$ times in Y_{j^*}, since it has been shown that j charging stops are sufficient to reach max.
- It is not possible that $max \geq \ell$.
  If this was the case, every schedule would charge $c_{opt}+1$ times in Y_{j^*}, since the last charging stop would not be necessary to reach v_ℓ.

Thus it is possible to transform every $S_i \in Y_{j^*}$ into a greedy schedule that behaves the exact same way as in Algorithm 9. This shows that we are able to transform Y_{j^*} into Y_{alg}, ensuring that it is impossible that there are more schedules in Y_{j^*}, since they would have been discovered by Algorithm 9. That is why the cardinality of Y_{alg} is j^*. □

5.2 Algorithm Approach

SecondAlg returns, similarly to FirstAlg, an independent program. The addition, compared to FirstAlg, is that the approach involves iterating over a range of values for s, starting with $s = j^*$ and ending at $s = i^*$. During the execution, all computed programs P are first evaluated in terms of $cost(P)$, before the method finally returns the program P_{second} with the observed minimum cost.

We can state that exploring an algorithm using $s < j^*$ is not beneficial, since having more independent schedules in a program P implies the possibility of a lower $W(P)$. Also, concerning $C(P)$, $s < j^*$ is not improving the total charging cost either, since using $s = j^*$ already gives every car the possibility to reach v_ℓ with c_{opt} charging stops.

Given that the pseudo code of FirstAlg is generalized to use a variable number s of independent schedules instead of i^*, pseudo code for SecondAlg can be expressed with Algorithm 10.

Algorithm 10 Implementation of SecondAlg

```
1: function SecondAlgorithm(I ∈ I^cb)
2:     P_second ← FirstAlgorithm(j*)
3:     minCost ← cost(P_second)
4:     for s in [j*+1..i*] do
5:         P ← FirstAlgorithm(s)
6:         cost ← CostIndependent(P)
7:         if cost < min then
8:             P_second ← P
9:             minCost ← cost
10:        end if
11:    end for
12:    return P_second
13: end function
```

From here on, two more aspects remain to be addressed: The runtime of the algorithm and the approach to calculate the cost of a given independent program.

5.2.1 Time complexity

Given the addition of the "for loop" which iterates over the s values, we want to investigate whether the computational complexity increases in a way s.t. the algorithm does not run in polynomial time.

We are able to deduce from (4.4) that no arriving or leaving zone can have length more than k. Combining this upper bound for i^* with a lower bound of 1 for j^* leads to

$$i^* - j^* \leq k - 1. \tag{5.4}$$

This means that there is a maximum of k calls to $FirstAlgorithm()$ and $Cost()$ (both running in polynomial time), preserving polynonmial runtime for SecondAlg.

5.2.2 Cost calculation

A requirement for the implementation of Algorithm 10 is a way to calculate the cost of a program. Fortunately, for independent programs, there is a relatively trivial way to compute the cost.

This is because the waiting cost is only originating from the first charging station of every schedule. Since only a single car can leave the first charging station of a schedule at a given time step, all subsequent charging stations in that schedule will be reached at successive time steps, thus no more queuing. A method to return the cost of an independent program is presented with Algorithm 11.

Algorithm 11 Returns $cost(P)$ for an independent program P

```
1: function CostIndependent(P)
2:     C(P) ← 0
3:     for S_i in P do
4:         C(P) ← C(P) + |S_i|
5:     end for
6:     W(P) ← 0
7:     for S_j in unique(P) do
8:         cars ← 0
9:         for S_i in P do
10:            if S_j == S_i then
11:                W(P) ← W(P) + cars
12:                cars ← cars + 1
13:            end if
14:        end for
15:    end for
16:    return C(P) + W(P)
17: end function
```

5.3 Performance Analysis

This chapter has introduced an improved algorithm which, due to its brute force nature concerning the value of s, allows us to state that

$$\forall I \in \mathcal{I}^{cb}: \\ cost(SecondAlgorithm(I)) \leq cost(FirstAlgorithm(I)). \tag{5.5}$$

Unfortunately however, as the following Theorem 5.3.1 shows, SecondAlg is not optimum for every possible critical blocks instance either.

Theorem 5.3.1. The program returned by SecondAlg is not optimum for every $I \in \mathcal{I}^{cb}$.

Proof. To show this, we make use of the instance depicted in Figure 5.1, which has parameters equal to $K = \{2, 3, 4, 6, 7, 8, 9, 11, 12, 13\}$, $n = 4$, $k = 3$, and $\ell = 15$. Let I denote this instance for the rest of the theorem.

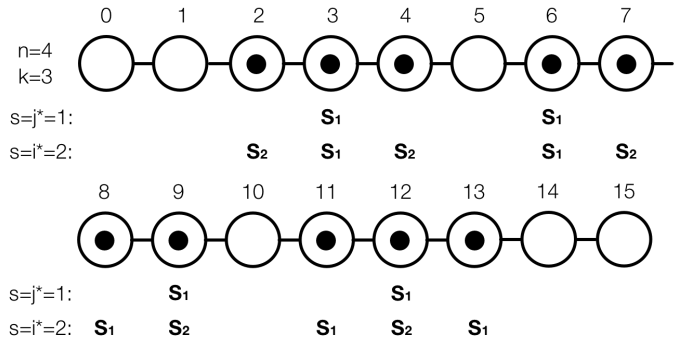

Figure 5.1: SecondAlg applied to $I \in \mathcal{I}^{cb}$.

Calculating the cost, we first evaluate program P_1 using $s=j^*=1$ schedule. We obtain

$$\begin{aligned} cost(P_1) &= C(P_1) + W(P_1) \\ &= 16 + 6 \\ &= 22. \end{aligned} \tag{5.6}$$

Using $s=i^*=2$ schedules, we can compute $cost(P_2)$ after having balanced all 4 cars over $K' = \{2, 3\}$ in order to minimize $W(P_2)$. We obtain

$$\begin{aligned} cost(P_2) &= C(P_2) + W(P_2) \\ &= 20 + 2 \\ &= 22. \end{aligned} \tag{5.7}$$

To sum up, there is

$$cost(P_{second}) = min(cost(P_1), cost(P_2)) = 22. \tag{5.8}$$

A lower cost can be obtained by creating a program P', where

$$\begin{aligned} &P'(unique) = \{\{3, 6, 9, 12\}, \{2, 4, 7, 8, 11, 13\}\}, \\ &S_1 = S_2 = S_3 = \{3, 6, 9, 12\}, \\ &S_4 = \{2, 4, 7, 8, 11, 13\}, \\ &P' = \{S_1, S_2, S_3, S_4\}. \end{aligned} \tag{5.9}$$

The cost of P' is

$$\begin{aligned} cost(P') &= C(P') + W(P') \\ &= 18 + 3 \\ &= 21 \end{aligned} \tag{5.10}$$

and we can conclude that

$$21 = cost(P') < cost(P_{second}) = 22. \tag{5.11}$$

□

Notable is that, when comparing P_2 and P', we observe:

- That $uniqueStations(P_2)$ and $uniqueStations(P')$ return the exact same set with a cardinaltiy of 10.
- That the cardinalities of the unique schedules are different. The number of charging stations in the two schedules of P_2 are 5 and 5, respectively, whereas the number of charging stations in P' are 4 and 6, respectively.
- That the number of cars assigned to the schedules is different. P_2 balances the cars by assigning 2 cars to each schedule. P' assigns 3 cars to the first schedule and 1 to the second schedule.

Those crucial observations lead us to the next chapter of this thesis.

Chapter 6

Critical Blocks - Third Algorithm

Looking back at the observations made at the end of Chapter 5, it can be concluded that the modification needed to obtain P' from P_2, can be summed up as:

- Decreasing the number of charging stations of the first unique schedule by one, while increasing the number of charging stations of the second unique schedule by one.
- Increasing the number of cars assigned to the first unique schedule by one, while decreasing the number of cars assigned to the second unique schedule by one.

We can generalize this by denoting x_i the number of cars that are assigned to schedule $S_i \in unique(P)$. Given $S_i, S_j \in unique(P)$, we can create P' by

- Decreasing $|S_i|$ by 1,
- Increasing $|S_j|$ by 1,
- Increasing x_i by 1,
- Decreasing x_j by 1,

which modifies $cost(P)$ in a way that

$$\begin{aligned} & cost(P') - cost(P) \\ = \; & -x_i + (|S_i| - 1) + x_i + (x_j - 1) - |S_j| - (x_j - 1) \\ = \; & |S_i| - |S_j| - 1. \end{aligned} \tag{6.1}$$

We assume that we decrease the schedule with the fewer elements among S_i and S_j. Given this, the result in (6.1) shows that, the greater the difference $\Big||S_i| - |S_j|\Big|$, the better the gain in total cost when applying this set of modifications. Obviously, this is a purely theoretical analysis. Since the existence of two independent schedules with the prescribed number of stops might not be possible, those changes are not always feasible.

Having analyzed this, it seems that studying the following two concepts is essential in order to develop an improved algorithm for any $I \in \mathcal{I}^{cb}$:

- Given that, during computation, we already know $|uniqueStations(P)|$ beforehand, what is the best distribution of the charging stations among a fixed number $s = |unique(P)|$ of independent schedules?
- Given the resulting set Y of s independent schedules, how to distribute the n cars to the schedules to create a program P of minimum cost (among all programs with $unique(P) = Y$)?

Let us start analyzing the second concept first.

6.1 Distributing Cars

So far, for FirstAlg and SecondAlg we were not obliged to dive into this question. Since both algorithms return a serial set of schedules, balancing the cars while referring to Theorem 3.1.3 was sufficient. Given a non serial set of schedules however, the number of charging stations in each schedule can differ by more than one. This change makes the distribution of cars among schedules less trivial.

6.1.1 Procedure

Let Y denote a set of independent schedules. Also, we define by o the cardinality of a subset of cars, s.t.

$$o = |\{a_1, a_2, ..., a_o\}| \leq n. \tag{6.2}$$

Among all programs with o cars using the schedules in Y, we denote $D(o)$ the distribution of the first o cars over the schedules in Y that leads to the program with the lowest cost.

We once again reuse the notation of x_i being the number of cars assigned to S_i. The distribution $D(o)$ is defined as a two dimensional array, s.t.

$$D(o) = \Big[[|S_1|, x_1], [|S_2|, x_2], ..., [|S_s|, x_s]\Big]. \tag{6.3}$$

To assign all n cars, we initialize $D(0)$ according to

$$D(0) = \Big[[|S_1|, 0], [|S_2|, 0], ..., [|S_s|, 0]\Big]. \tag{6.4}$$

In order to get from $D(0)$ to $D(n)$, it is important to state that we can obtain $D(o)$ by taking $D(o-1)$ and adding the oth car to one of the schedules. This is why we can build $D(n)$ from $D(0)$, in a way that

$$\begin{aligned} & min = \underset{i \in D(o)}{\operatorname{argmin}}(i[0] + i[1]), \\ & D(o)[min][1] += 1, \\ & D(o+1) = D(o). \end{aligned} \tag{6.5}$$

Commenting on (6.5), by adding the oth car, the procedure selects the schedule that increases the total cost the least when assigning a new car to it. This is done by summing the schedule's cardinality and the waiting cost w_o of car a_o. Next it updates the value x_o accordingly.

With Algorithm 12 a recursive method is expressed in pseudo code that can be called for $o = n$ to assign all cars to an independent set of schedules Y.

Algorithm 12 Assigning oth car to set of independent schedules Y

```
1: function AssignCars(Y = {S_1, S_2, ..., S_s}, o)
2:     if o == 0 then
3:         return [[|S_1|, 0], [|S_2|, 0], ..., [|S_s|, 0]]
4:     else
5:         D_{o-1} ← AssignCars(Y, o−1)
6:         minCost ← D_{o-1}[0][0] + D_{o-1}[0][1]
7:         min ← 0
8:         for i in [1..s] do
9:             cost ← D_{o-1}[i][0] + D_{o-1}[i][1]
10:            if cost < minCost then
11:                min ← i
12:                minCost ← cost
13:            end if
14:        end for
15:        D_{o-1}[min][1] ← D_{o-1}[min][1] + 1
16:        return D_{o-1}
17:    end if
18: end function
```

6.1.2 Correctness

However, this procedure can only be claimed correct if it can be established that creating $D(o)$ from $D(o-1)$ doesn't require more change, other than adding the oth car to one of the schedules.

Theorem 6.1.1. Let Y be a set of s independent schedules and $D(o-1)$ a distribution that leads to P_{o-1} with minimal $cost(P_{o-1})$ (with respect to the fixed Y and o). It is possible to preserve $D(o-1)$, add the oth car to one schedule which leads to $D(o)$, and obtain P_o with minimal $cost(P_o)$ with respect to the fixed set of schedules.

Proof. We can model the car assignments as a matrix (Figure 6.1) with n rows, s columns, and cells $Cell_{j,i}$, s.t.

$$1 \leq j \leq n \text{ and}$$
$$1 \leq i \leq s. \tag{6.6}$$

Every cell $Cell_{j,i}$ contains the cost value that is added to the total cost when the number of cars, that use schedule S_i, is increased from $j-1$ to j.

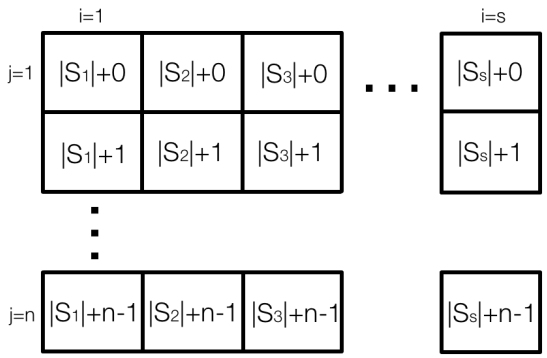

Figure 6.1: Car assignment matrix with cell costs.

To assign all cars, we need to select n cells and sum the cell values to obtain the total cost of the program. However, when selecting, there is one restriction: One can only select $Cell_{j,i}$ if $Cell_{j-1,i}$ has already been selected (selecting $Cell_{1,i}$ is always possible). This is because, in order to join the queue as the jth car, the $j-1$th car has to be in the queue already.

A crucial observation is that the cell values are increasing by one from top to bottom for a given S_i. To show that it is possible to obtain $D(o)$ by taking $D(o-1)$ and adding the oth car to one of the schedules, it is sufficient to state that the cell with the lowest value (that hasn't yet been selected), can always be selected for the oth car. □

6.2 Distributing Charging Stations

Moving on to the next concept, we want to steer the focus to Definition 2.0.5 that states that

$$u = |uniqueStations(P)|. \tag{6.7}$$

Given values for u and s, this section investigates all distributions of the number of charging stations among s schedules. We assume that, in a second step, we apply (6.5) to assign the cars in an optimum way.

6.2.1 Optimum distribution

Unrelated to any problem instance, the distribution of u among s that leads to the lowest $cost(P)$ is defined as the optimum distribution D_{opt}, s.t.

$$D_{opt} = [1, 1, ..., u-(s-1)], \tag{6.8}$$

with $D[i]$ denoting the number of charging stations in schedule S_{i+1}.

The aim of the following Theorem 6.2.1, is to show that D_{opt} is optimum for any u and s.

Theorem 6.2.1. Given u and s, D_{opt} is optimum.

Proof. For this theorem, we refer to Theorem 6.1.1 and the related cost matrix displayed on Figure 6.1. For the purpose of distributing charging stations, we modify the matrix in a way that it only displays the waiting cost (not considering the schedule cardinality), as illustrated with Figure 6.2.

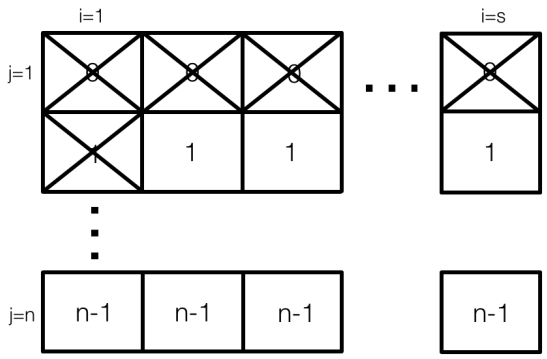

Figure 6.2: Modified charging station distribution matrix.

To obtain a distribution, we have to mark u cells as non selectable for the next step, which is the car assignment. This is needed to attribute every schedule a number of charging stations. Similar to the previous theorem, $Cell_{j,i}$ can only be marked as non selectable if $Cell_{j-1,i}$ has already been marked (marking $Cell_{1,i}$ is always possible). On top of that, we have to mark at least $Cell_{1,i}$ for every S_i, in order to prevent $S_i = \emptyset$.

To clarify and provide an example, let us assume that we have, besides every $Cell_{1,i}$, marked $Cell_{2,1}$ as non selectable for S_1 (Figure 6.2). This means that S_1 has cardinality $|S_1| = 2$ and the first car assigned to S_1 (if assigned in the next step) adds the value of $Cell_{3,1}$, being equal to 2, to the total cost of the program.

We can use an exchange argument to show that D_{opt} is optimum and that the best strategy is to mark the highest value cells as non selectable. Whenever we can switch from marking a cell with value *low* to a cell with value *high*, s.t.

$$high > low, \tag{6.9}$$

this is to be done. During the car assignment in the next step, this single change means that we have the possibility to add the value of *low* instead of the value of *high* to the total cost. That is why this switch will result in a program with lower or equal cost compared to a program that doesn't incorporate the switch. By repeating this until no more such switch is possible, we end up with D_{opt}. □

6.2.2 Perfectly unbalanced

Obviously, studying the optimality of D_{opt} is a theoretical analysis again. This is because having schedules following D_{opt} is usually only feasible for a low value of ℓ (allowing to reach v_ℓ with one stop only) and an unnecessarily high value of u. Thus we need a definition for an independent program P, given an instance I, where the charging stations are distributed as optimally as possible.

Definition 6.2.1 (Perfectly unbalanced program). Given an independent program P for instance I and two schedules $S_i, S_j \in unique(P)$. Program P is said to be *perfectly unbalanced* if it is impossible (leads to unfeasible schedules) to decrease $|S_i|$ by one charging station, while increasing $|S_j|$ by one charging station, s.t. the obtained schedule cardinalities are

$$|S_i| < |S_j|. \tag{6.10}$$

6.3 Algorithm Approach

Theorem 6.2.1 provides us with the knowledge that, for a given number u of charging stations, an unbalanced distribution among the schedules is desired and of advantage for the cost of the program.

Unfortunately, FirstAlg and SecondAlg have been designed without incorporating this knowledge. Logically, this is the correction that ThirdAlg tries to make. Using the greedy Algorithm 8 from Chapter 5, ThirdAlg computes as many greedy independent schedules as possible.

Algorithm 13 Implementation of ThirdAlg

```
1: function ThirdAlgorithm(I ∈ 𝓘^cb)
2:     Y ← ∅
3:     X ← ∅
4:     S_i ← Greedy(K, X, k, ℓ)
5:     while |S_i|! = 0 do
6:         S_i ← Greedy(K, X, k, ℓ)
7:         Y.append ← S_i
8:         X ← X ∪ S_i
9:     end while
10:    return P_third = CreateP(AssignCars(Y, n))
11: end function
```

In contrast to FirstAlg and SecondAlg, it is not necessary to know the number of schedules in advance or to run the algorithm for multiple values of s. Based on the assignment of cars according to Algorithm 12, a final program P_{third} is created and returned. It should be noted that some schedules might not be used by any cars in the end.

Figure 6.3 shows the result of applying ThirdAlg to the exact same instance that has been used at the end of Chapter 5, to show that SecondAlg is not optimum.

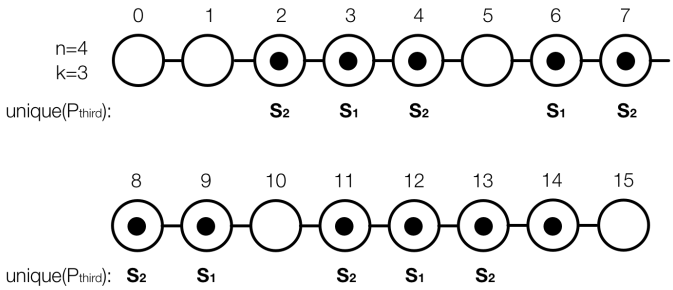

Figure 6.3: Visualization of $unique(P_{third})$.

Using the car assignment procedure presented in this chapter, we end up with

$$\begin{aligned} P_{third} =&(\{3,6,9,12\},\{3,6,9,12\},\\ &\{3,6,9,12\},\{2,4,7,8,11,13\}). \end{aligned} \tag{6.11}$$

It turns out that $P_{third} = P'$, with P' being the program in Chapter 5 that beats SecondAlg (shown in (5.11)).

What is worth mentioning is that, although the initial task was to solve any instance $I \in \mathcal{I}^{cb}$, this algorithm doesn't make use of any specific features of critical blocks. Without any mathematical support, it can be carefully stated that one could expect decent results for arbitrary roads as well.

Because of the greedy approach, we estimate P_{third} to be perfectly unbalanced. However, we do not seek evidence for this because we are missing another crucial information. We don't have any knowledge about whether u represents a minimum or in general a value that allows us to compute an independent program with minimal cost. Although no instance $I \in \mathcal{I}^{cb}$, that shows that ThirdAlg is not optimum has been found, a proof for optimality is impossible without any further reasoning about u. That is why the next chapter will go more in depth concerning the used number of charging stations.

Chapter 7

Critical Blocks - Fourth Algorithm

This chapter makes yet another attempt to design an algorithm that solves any $I \in \mathcal{I}^{cb}$. As a guidance, we will rely on all discoveries that have been made related to the topics of the last chapters.

This all starts with the ability to reason more about the value u, being the number of unique charging stations in a program or set of schedules. That is why a definition for a minimal u is given in the following.

Definition 7.0.1 (Value of $u^*(s)$). Given an independent program P using $s = |unique(P)|$ schedules for an instance I. We denote $u^*(s)$ the minimum value for $|uniqueStations(P)|$, s.t.

$$u^*(s) \leq u = |uniqueStations(P)|. \tag{7.1}$$

Making use of this definition, the goal is to develop an algorithm that computes an independent program P_{fourth} that satisfies:

- That the number of schedules s is brute forced in the range from j^* to i^* by last chapter's Algorithm 12 (car assignment).
- That cars are assigned in an ideal way using the same procedure.
- That $|uniqueStations(P_{fourth})|$ is equal to $u^*(s)$.
- That P_{fourth} is perfectly unbalanced.

7.1 Computing $u^*(s)$

The next two theorems propose a way to compute the value of $u^*(s)$ for any $I \in \mathcal{I}^{cb}$, which is needed for FourthAlg.

Theorem 7.1.1. Given an instance I and $s \leq i^*$, there exists a serial program P that has $|uniqueStations(P)|$ equal to $u^*(s)$.

Proof. Given an instance I and $s \leq i^*$, let P_{u^*} denote an independent program that has

$$|uniqueStations(P_{u^*})| = u^*(s). \tag{7.2}$$

We can refer to Theorem 3.2.7 to transform P_{u^*} into serial P' s.t.

$$|uniqueStations(P')| = |uniqueStations(P)|. \tag{7.3}$$

□

Theorem 7.1.2. Let Y_{second} denote the set of schedules that SecondAlg computes for every $I \in \mathcal{I}^{cb}$ and s in the range from j^* to i^*. For every s there is $|uniqueStations(Y_{second})| = u^*(s)$.

Proof. In Chapter 4 we have established that FirstAlg (basis for SecondAlg) computes serial schedules. Looking back at the previous Theorem 7.1.1, we proved that for every instance I and $s \leq i^*$, a serial program or set of schedules exists that has $|uniqueStations(P)| = u^*(s)$.

We claim that the serial sets of schedules computed by SecondAlg have $|uniqueStations(Y_{second})|$ equal to $u^*(s)$, since they use the least amount of charging stations among all serial sets of schedules for that s. This is because in every block B_m, every schedule traverses k edges before recharging. All the schedules that end up charging at a node index higher than $end(leavingZone(B_m))$, are shifted back into $leavingZone(B_m)$. This is done in an "ideal" way, guaranteeing that the highest possible charging station indices are used. To sum up, every schedule charges as late as possible, while still following the serial property, leading to the use of a minimal amount of charging stations in total for all s schedules. □

7.2 Algorithm Approach

Not only have we established a way to compute $u^*(s)$, but also, as a side effect, are we provided with an algorithm that uses the value. Nonetheless, the shortcoming of SecondAlg is that the charging stations are equally divided over the unique schedules (serial program). This is what we try to avoid with FourthAlg and for that purpose we introduce a new version of Algorithm 5, which has the responsibility of backshifting the charging stops of the schedules into $leavingZone(B_m)$.

The new version presented with Algorithm 14 is not based on the alternating ordering of the schedules, but instead it prioritises schedules with lower indices and assigns them to charging stations with the highest index in $leavingZone(B_m)$. If necessary, schedules that don't need to be shifted (e.g. S_1 and S_2 in $leavingZone(B_2)$ on Figure 7.1) are nevertheless moved to stations with lower indices in order to free up stations for schedules with lower indices. The priorities of the schedules remain unchanged for every block B_m and that is the reason why $S_1 = S_{greedy}$ in FourthAlg.

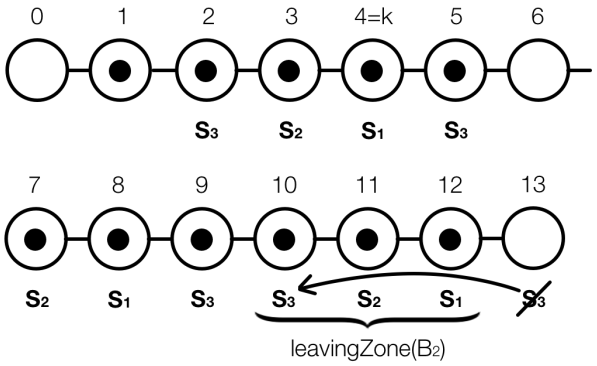

Figure 7.1: Since S_1 and S_2 have priority over S_3, they keep their stop and S_3 has to charge at v_{10}.

Algorithm 14 Shifting schedules based on prioritising lower schedule indices

```
1: function PriorityShift(Road, start, end)
2:     for v in [ℓ..end+1] do
3:         if Road[v] > 0 then
4:             Shift.insertOrdered ← Road[v]
5:             Road[v] ← 0
6:         end if
7:     end for
8:     for v in [end..start] do
9:         if |Shift| > 0 then
10:            if Road[v] == 0 then
11:                Road[v] ← Shift[0]
12:                Shift[0].remove
13:            else
14:                Shift.insertOrdered ← Road[v]
15:                Road[v] ← Shift[0]
16:                Shift[0].remove
17:            end if
18:        end if
19:    end for
20: end function
```

Throughout the algorithm, indices are added to the list $Shift$ in a way that $Shift$ stays ordered increasingly. Also, we rename $ComputeSchedules$ (Algorithm 4) to $NewComputeSchedules()$ after the only change to now make the call to $PriorityShift()$ instead of $Shift()$ (Line 21). Hence after adding the call to $AssignCars()$, which was introduced in the last chapter, we denote pseudo code for FourthAlg computing P_{fourth} by Algorithm 15.

Algorithm 15 Implementation of FourthAlg

```
1: function FourthAlgorithm(I ∈ 𝓘^cb)
2:     Y ← NewComputeSchedules(K, k, ℓ)
3:     return P_fourth = CreateP(AssignCars(Y, n))
4: end function
```

Important to note is that we do not have to run the algorithm for several values of s. FourthAlg computes a maximum of $s = i^*$ independent schedules, knowing that S_i will not be influenced by the addition of S_{i+1} (due to the priority shifting). Same as in ThirdAlg, the function $AssignCars()$ can modify the value of s by not assigning any cars to some schedules. Concerning this, we know that given S_i, S_j, there is

$$i < j \implies |S_i| \leq |S_j|. \tag{7.4}$$

This is why $|S_i| = |S_j|$ might be the only scenario where S_i is not assigned any car while S_j is used by one. In general however, schedules are left out (not assigned any car) by decreasing schedule index. That is why, for the resulting s, we are ensured that P_{fourth} uses $u^*(s)$.

7.3 Performance Analysis

As usual, in the following section we will discuss the performance of the newly introduced algorithm.

7.3.1 Analysis for critical blocks

To claim that FourthAlg returns the independent program with the lowest cost for any $I \in \mathcal{I}^{cb}$, we base ourselves on the next three theorems. First of all, Theorem 7.3.1 explains why FourthAlg uses $u^*(s)$ unique charging stations. To do this, we are required to study the effect of the newly introduced $PriorityShift()$ function and compare the behaviour to SecondAlg.

Theorem 7.3.1. Given program P_{fourth}, computed by FourthAlg for any instance $I \in \mathcal{I}^{cb}$, $|uniqueStations(P_{fourth})|$ is always equal to $u^*(s)$, for any $j^* \leq s \leq i^*$.

Proof. In Theorem 7.1.2 we have shown that the serial sets of schedules, denoted Y_{second} and computed by SecondAlg for any s from j^* to i^*, use $u^*(s)$ unique charging stations. Furthermore, we know that the only difference between the schedule computation in SecondAlg and FourthAlg is the new shifting procedure based on schedule priorities.

Given a program P for any $I \in \mathcal{I}^{cb}$, we denote by $Last(B_m)$ the set containing the indices of the last charging stop of every $S_i \in unique(P)$ in block B_m. In the following we aim to show that this set contains the same indices for P_{second} and P_{fourth} for every B_m and any s from j^* to i^*. If this is the case, both algorithms clearly use the exact same number of charging stations since their schedule behaviour inside block B_m is the same (attempting to do k edge traversals). Hence we would obtain

$$\begin{aligned} & |uniqueStations(P_{fourth})| \\ = \; & |uniqueStations(P_{second})| = u^*(s). \end{aligned} \tag{7.5}$$

To do this, we denote by $In(B_m)$ the set of charging station indices that are reached by the schedules that arrive

in $leavingZone(B_m)$ by doing k edge traversals before each stop. Also, we define by $out(B_m)$ the number of schedules that arrive at an index which is higher than $end(leavingZone(B_m))$, s.t.

$$out(B_m) = |unique(P)| - |In(B_m)|. \tag{7.6}$$

Lastly, let $Free(B_m)$ denote the set of the highest possible indices of unoccupied (free) charging stations in $leavingZone(B_m)$ before $PriorityShift()$ begins, s.t.

$$\begin{aligned} |Free(B_m)| &= out(B_m) \text{ and} \\ Free(B_m) \cap In(B_m) &= \emptyset. \end{aligned} \tag{7.7}$$

Independently whether we make the call to $Shift()$ or $PriorityShift()$, $Last(B_m)$ will always be composed by

$$Last(B_m) = In(B_m) \cup Free(B_m). \tag{7.8}$$

This is because, in both procedures, no schedule is backshifted more than necessary. Hence, SecondAlg and FourthAlg use the exact same charging stations for any s from j^* to i^* and $|uniqueStations(P_{fourth})| = u^*(s)$. □

Moreover, an explanation why P_{fourth} is indeed perfectly unbalanced, is necessary.

Theorem 7.3.2. For any instance $I \in \mathcal{I}^{cb}$, program P_{fourth} is perfectly unbalanced.

Proof. To start the proof, we define the schedules $S_h, S_i, S_j \in unique(P_{fourth})$ in a way that

$$1 \leq h < i < j \leq s. \tag{7.9}$$

In order to possibly disprove that P_{fourth} is perfectly unbalanced, we would need to be able to reduce the cardinality $|S_i|$ by one and increase the cardinality $|S_j|$ by one. This is because of Definition 6.2.1 about perfectly unbalanced programs and last section's implication (7.4).

Since S_i is greedy, it's cardinality can only be reduced if it is modified to stop at a charging stop that is used by S_h (which has priority over S_i). If making this switch indeed leads to S_i charging one time less often, we end up by increasing the number of charging stops of S_h by one (restoring the minimum of $u^*(s)$), which can now use the initial charging stop of S_i. This leads to a less balanced distribution of charging stations. In general, we are unable to increase the number of charging stops of S_j without also doing the same for S_h, when decreasing the number of charging stops for S_i. This shows that P_{fourth} is perfectly unbalanced. □

Lastly, we still have to investigate programs that use $u > u^*(s)$ unique charging stations and compare them to perfectly unbalanced programs using $u^*(s)$ unique charging stations.

Theorem 7.3.3. For a given instance I and s in the range from j^* to i^*, let P_{u^*} denote an independent and perfectly unbalanced program using $u^*(s)$ unique charging stations. Also, let P' define an independent program that uses u unique charging stations for the same s, s.t. $u > u^*(s)$. There is always $cost(P_{u^*}) \leq cost(P')$.

Proof. Let x denote the difference in unique charging stations between P_{u^*} and P', s.t.

$$x = |uniqueStations(P')| - |uniqueStations(P_{u^*})|. \tag{7.10}$$

Let $S_{high} \in P_{u^*}$ define the schedule in P_{u^*} with the highest number of charging stations. Given $S_i \in P_{u^*}$ and $S'_i \in P'$, s.t. $i \neq high$, obviously the best possible distribution of charging stations among schedules for P' is that

$$\begin{aligned} |S'_i| &= |S_i| \text{ and} \\ |S'_{high}| &= |S_{high}| + x. \end{aligned} \tag{7.11}$$

In the best case, no car will be assigned to S'_{high}, leading to $cost(P_{u^*}) = cost(P')$. This explains why we add the x charging stations to S_{high}, increasing the probability of no car being assigned to it. The car assignment procedure clearly considers S_{high} as the last schedule, when it comes to assigning the first car to each schedule. Any other case, where S'_{high} is used by one car at least, states that $cost(P_{u^*}) \leq cost(P')$. □

7.3.2 Analysis for arbitrary roads

Having shown that P_{fourth} uses a minimum amount of charging stations, is perfectly unbalanced and determines s by assigning the cars in an ideal way, we can ask ourselves why this algorithm cannot be applied to instances that are not in $\mathcal{I}^{cb}$. Two aspects prevent us from doing this:

- So far we haven't studied i^* for arbitrary roads. That is why we have no knowledge about the upper bound of independent schedules. Letting the algorithm compute a set of independent schedules according to the definition of the procedure, does not provide a guarantee that this is actually the maximum number of schedules. As a result, waiting time could be higher than necessary.
- Furthermore, the way of computing the schedules in FourthAlg is specifically designed for critical blocks. We haven't established a method to compute $u^*(s)$ for any arbitrary road and hence do not know how to decide on the behaviour of the schedules.

7.4 Algorithm Comparisons

In the context of a recapitulating section, we compare FourthAlg to the three algorithms presented earlier in this thesis. To do this, we employ the relatively long road layout of the instance at the end of Chapter 4. This time however, we use $n=8$, while retaining $k=4$.

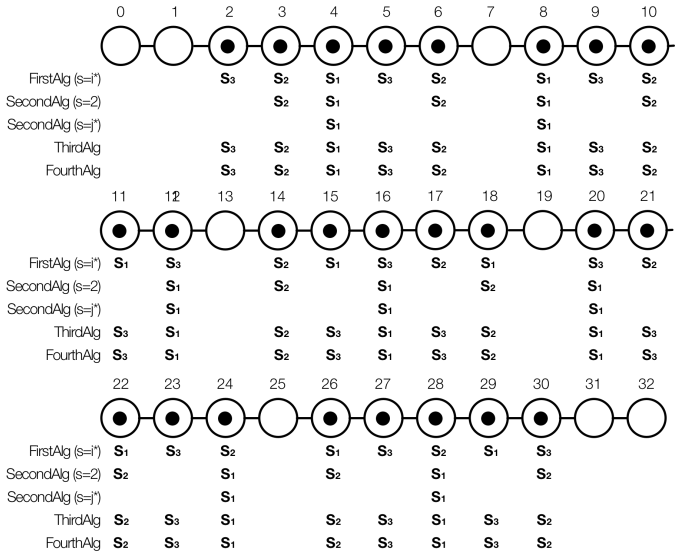

Figure 7.2: Overview of the main four algorithms applied to the same instance.

By applying the algorithm's methods to assign the 8 cars to the schedules shown in Figure 7.2, we end up with

$$\begin{aligned} cost(P_{first}) &= 73, \\ cost(P_{second}) &= 72, \\ cost(P_{third}) &= 71, \\ cost(P_{fourth}) &= 71. \end{aligned} \tag{7.12}$$

In fact, for any n, there is

$$cost(P_{fourth}) = cost(P_{third}) \leq cost(P_{second}) \leq cost(P_{first}). \tag{7.13}$$

The value of $n=8$ has been analyzed in a more precise way, because it is the smallest value where the cost is decreasing from P_{first} to P_{second} to P_{third}.

Furthermore, it can be observed on Figure 7.2 that $unique(P_{third}) = unique(P_{fourth})$. No clear evidence has been found that this is the case for any $I \in \mathcal{I}^{cb}$. However, by comparing the greedy schedule computation and the shifting based on schedule priorities, it seems that both algorithms are likely to produce the same output. That is why we note the following statement as a conjecture.

Conjecture 7.4.1. For any $I \in \mathcal{I}^{cb}$, P_{third} computed by ThirdAlg, and P_{fourth} computed by FourthAlg, there is always $unique(P_{third}) = unique(P_{fourth})$ and hence $cost(P_{third}) = cost(P_{fourth})$.

Chapter 8

Conclusion

By reflecting back at the previous chapters, we started this thesis by providing terminology, definitions and early observations for EfficientCharging. Next, all subsequent chapters discussed an algorithm tailored to solve inputs coming from the set of critical blocks instances. Gradually, we gathered new insights about the shortcomings of the procedure and hence used this information to adapt the next chapter's attempt.

Overall, the most significant contributions of this thesis can be summed up as follows:

- We have proposed a way to compute the value j^* for any instance I and the values i^* and $u^*(s)$ for any instance $I \in \mathcal{I}^{cb}$.
- We have shown that having an unbalanced distribution of charging stations over schedules is allowing lower program cost (in contrast to balancing cars over stations reducing $W(P)$).
- Ultimately, we have developed FourthAlg which computes the independent program with the lowest cost for every critical blocks instance.

Next to the restrictions on the studied problem instances, two aspects of EfficientCharging that unfortunately couldn't be solved in a meaningful way, should be named.

- First of all, we refer to Conjecture 3.2.1 about the conversion to independent programs. Without being able to prove the conjecture, the search for an algorithm that returns an optimum program for every instance turns out to be very difficult. Only minor insights regarding this topic have been given in this thesis, hence creating a suited starting point for future work.
- Secondly, it seems to be a difficult task to reason about which value for the number of unique schedules s can lead to a program with the lowest cost. Algorithm 12, responsible for assigning cars to the schedules, can adjust s by not adding any cars to some schedules. However, it seems that knowing the correct value for s before the actual computation of the schedules would create an advantage during the development stage of an algorithm.

In a more general comment, it seems likely that the solution for any instance of EfficientCharging can be computed in polynomial time. However, it turned out that the scope of this thesis has not been sufficient enough to address this question in a substantial way.

Finally, we should also mention that the problem discussed in this thesis underlies the framework set by EfficientCharging itself. This enforces that cars have the same battery capacity, traverse a simple path graph, start and end at the same node, leave the start node at the same time, and drive in the same direction. Therefore it is safe to claim that the broad problem of efficient centralized charging for electric cars is still far from solved. Lifting one ore more of the just mentioned constraints opens doors for new areas of research.

Bibliography

[1] Small Jonty. “Electric cars and the effects of cooperation on total charge time”. MA thesis. Department of Data Science and Knowledge Engineering, Maastricht University, 2018.

[2] Schneider Pit. “Electric Cars - Efficient Centralized Charging”. In: 1 (2019).