TECHNICAL UNIVERSITY OF MUNICH

Optimization of Driver Shift (and Break) Schedule using Simulated Annealing in Ride-Pooling Services

Master Thesis

Submitted in partial fulfillment of

the requirements for the degree of

Master of Science,
Transportation Systems

by

Shivam Arora

Academic Supervisor: **Qin Zhang**

Professorship for Modelling Spatial Mobility
Technical University of Munich

External Advisor: **Nico Kühnel**

MOIA GmbH, Berlin, Germany

December 23, 2021

Contents

Acknowledgments

This year, 2021, has been an exciting time of intellectual growth and rewarding, with many learning opportunities for me. It has been a busy year, not just with professional or academic pursuits but also with personal commitments. Despite the challenges, I would like to take this opportunity to thank everyone who helped me complete these projects for their generous contributions.

First of all, I would like to express my sincere gratitude to my external advisor, Dr Nico Kühnel, for his continued support and expert guidance throughout the thesis. Along with answering my questions patiently, he helped me improve the code and gave me plenty of new ideas/suggestions to incorporate into my thesis. I am indebted to him for his generosity.

Secondly, I am grateful to my academic supervisor, Dr Qin Zhang, for mentoring my thesis, giving me expert insights during the analysis of the results and thesis writing. Her constant constructive feedback kept me motivated throughout. I, too, am grateful to her for the help she provided in the last few days.

Thirdly, I would like to acknowledge the feedback I received from Felix Zwick, who helped me shape my thesis, and I thank him for providing a detailed introduction to the topic.

Lastly, I would like to thank my family and friends for their love and support. Also, for motivating and helping me throughout these years.

Abstract

Ride-pooling is an on-demand service that offers convenient and cheap mobility solutions to reduce traditional car traffic by pooling several trips together in a single-vehicle. Due to the involvement of humans, they are not as efficient. Researchers have conducted several simulation studies to examine and improve this current inefficient system with the help of MATSim. This open-source Java application is helping them simulate large-scale ride-pooling operations. However, the simulations did not consider operational challenges in the ride-pooling studies until transport specialists developed an extension that incorporated the inefficient human factors that form the driver shift (and break) schedule.

It is shift (and break) schedules that determine drivers' working hours. A driver's active shift time is every working hour, while their non-working hours account for their breaks. The active shift serves as a Travel Supply to the system, serving ride-pooling requests from customers that impact the static Travel Demand. Due to the uncertainty of travel supply and demand at any given moment, ride-pooling is inefficient. In order to eliminate uncertainty, demand and supply must be balanced to maintain an optimal equilibrium. One can only manipulate the Travel Supply, not the Travel Demand to achieve such optimality. The proposed model aims to resolve this issue by optimizing the shift schedules of drivers to reduce excess demand of unserved rides by a heuristic algorithm. Furthermore, the model ensures no excess Travel Supply of driver schedules that could potentially increase operating costs.

Following a comprehensive literature review, the Simulated Annealing algorithm was adopted as the heuristic algorithm in the model due to its various advantages, including its ability to provide a globally optimal solution, its guarantee of convergence, and the lack of complicated mathematical equations. Nevertheless, it raises the question of whether a heuristic algorithm like Simulated Annealing can optimize drivers' shift (and break) schedules in ride-pooling services?

Having analyzed the model results, this thesis model is discerned for its potential, strength, and weaknesses in answering the research question. The model seemed to produce promising results under certain parametric conditions, so it was concluded that the model and its algorithm have the potential to optimize driver shift (and break) schedules.

Acronyms

AMoD Autonomous Mobility on-Demand

BS Break Scheduling

DRT Demand Response Transit

ESP Employee Scheduling Problem

GA Genetic Algorithms

GESP General Employee Scheduling Problem

IPU Iterative Proportional Updating

MATSim Multi-Agent Transport Simulation

MITO Microscopic Transport Orchestrator

MSB Multiple Solution Based

SA Simulated Annealing

SDP Shift Design Problem

SSB Single Solution-Based

TNC Transportation Network Companies

TS Tabu Search

VKT Vehicle Kilometres Travelled

Glossary

active shifts per hour A LinkedHashMap with keys as time in seconds and values as the number of active shifts (1s) in a shift plan, in reference to 3.4. Serves the model as the supply side.

average rejection rate Using the rejection rate per hour (LinkedHashMap), all the rate values in it is summed up and divided by the number of values to get the average rejection rate.

driver (shift and break) schedule A duty schedule or roster of a driver that contains the work plan (shift) and break time.

maximum rejection rate Using the rejection rate per hour (LinkedHashMap), the maximum rate value that is obtained from any of the index becomes the maximum rejection rate.

MOIA A ride-pooling operator company in Hamburg and Hanover

rejection rate per hour A LinkedHashMap with keys as time in seconds and values as the ratio of rejections and submitted requests. A metric that measures how much supply must be delivered to meet demand.

rejections per hour A LinkedHashMap with keys as time in seconds and values as the number of rejected requests

submitted requests per hour A LinkedHashMap with keys as time in seconds and values as the number of submitted requests from the travel demand. Serves the model as the demand side.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Due to technological advancements in the transportation sector, new transport options are available to address traffic congestion, parking shortages, vehicular emissions, non-renewable energy consumption, and expensive travel costs. Shared mobility companies or TNC like (Uber Pool , 2021; Clevershuttle , 2021; MOIA , 2021) are emerging as the key players who can change the transportation industry by offering ride-hailing and ride-pooling services. By definition, ride-pooling is when several passengers are picked up from different locations and dropped off to different destinations on the same vehicle, thus conserving resources, such as energy and cost, and increasing vehicle occupancy, leading to fewer vehicles on roads. These services reduce emissions because of smaller Vehicle Kilometers Traveled (VKT) value and improve transport efficiency. To sustain such an efficient and intricate service, one has to use another vital resource. Labour consists of professional vehicle drivers or operators to transport passengers, software developers that develop apps to match passengers and drivers, and transportation engineers and operations researchers like us to help maintain a seamless and efficient mobility ride-sharing system. Hence, optimal resource allocation is of significant interest to researchers and practitioners in the ride-hailing, ride-pooling service industry. A loss of efficiency is inevitable when employees are not utilized or appropriately organized and operation challenges are not resolved. Therefore, a vital component of effectively utilizing them is scheduling the shifts and break times of the workforce that directly contributes to the cost-effectiveness and efficiency of the system (Kletzander and Musliu, 2021).

From an economic perspective, vehicle drivers working in shift schedules can be considered supply and passenger requests as demand. When the demand is higher than supply, prices tend to increase. The price the system has to pay is the non-served rides or the rejected requests, leading to a loss of profit for the ride-pooling company. However, when the supply is higher than demand, the system loses efficiency, which is also not suitable for the company. Therefore, to avoid increasing the system's price or becoming inefficient, one must lead the system to

an operational system equilibrium when supply is enough to meet the demand. The primary factor influencing the active supply is the driver costs (hourly salary), which are influenced by their shift schedule. So, in theory, an optimal system will equalize the operational supply and demand that will reduce the prices and improve efficiency. As a transportation researcher, this thesis study will apply a heuristic technique to help us search for the optimal operational system that will benefit the ride-pooling industry.

1.2 Background

In light of expected population growth, urbanization and traffic vehicles on roads, it is likely that urban mobility will need to be reshaped to achieve a comparable level of mobility (Wilkes et al., 2021). As a result, space will become a scarce resource in the future, leading to an increase in traffic congestion on roads, overcrowding in public transport and a decrease in parking space on streets. Ride-hailing and ride-sharing services, which bundle trips into one and use a single vehicle to eliminate several trips, have been introduced that promise to reduce traffic levels, reduce consumption in urban areas and remove the necessity to look for parking (Henao and Marshall, 2019). In an environment with higher vehicle occupancy, space on the roads will improve, leading to lesser congestion. There would be fewer emissions per kilometre travelling with ride-pooling vehicles when compared to travelling with personal vehicles. Therefore, (Chan and Shaheen, 2012) ride-hailing or ride-pooling services can reduce congestion of vehicles, energy utilization, and emissions. Ride-pooling services scale positively: as fleet sizes grow and demand increases, empty travel will decrease, and the potential for pooling trips will increase. Large-scale ride-pooling systems can provide a reliable, convenient, and sustainable transportation alternative to the current urban transport system.

Pooling vehicle journeys can be traced back to the 1920s in Los Angeles, when streetcar passengers were picked up for a shared rate by vehicle owners (Shaheen and Cohen, 2019; MIT, 2009). The conventional methods of carpooling and ride-sharing resulted in unplanned trips that would have occurred in a driver's private car by chance, where the passenger may or may not pay the driver. Whereas a pooled or shared-ride has become paid service, a company-hired driver makes a trip because there is a demand for such trips. As the pooling industry has evolved, riders and drivers were previously matched manually in bulletin boards or via the telephone. Recently, they have been matched through app-based services using matching and dispatching algorithms.

As for Germany, the authorities distinguish between services that offer ride-hailing and ride-pooling services primarily to regulate passenger transport. Local governments usually issue limited taxi licences in German cities and regulate their fares, forcing TNCs to hire drivers privately as they are not subjected to the regulations set by the local governments. Nonetheless, Germany will amend such regulatory practices in future and bring less stringent rules to attract cheaper digital mobility solutions. As a result, many transportation and operation researchers have started taking an interest in ride-pooling or ride-hailing studies for German cities (Zwick et al., 2021; Ennen and Heilker, 2020; Kuehnel et al., 2021-05; Wilkes et al., 2021).

Small-scale pilot studies are being conducted in major cities with limited fleet sizes to determine the effects of ride-pooling systems in Germany (e.g., (Münchner Verkehrsgesellschaft

GmbH , 2021; Clevershuttle , 2021; ioki , 2021) from Munich, Berlin, and Hamburg respectively). (MOIA , 2021), a recently founded company in 2019 that entered the Hamburg and Hanover markets, operates the largest ride-sharing fleet in Europe, has a fleet of approximately 500 vehicles; several simulation studies were researched and analysed in (Multi-Agent Transport Simulation , MATSim) implementing the ride-pooling services based on their demand data. (Zwick and Axhausen, 2020) They compared the performance of the two extensions (DRT and AMoD) in MATSim used in the study. To help study ride-pooling services more realistically with the inclusion of operational challenges like driver shifts and breaks, (Kuehnel et al., 2021-05) piloted a study where they modified the (Bischoff et al., 2017)'s (Demand Responsive Transit , DRT) extension to implement the operational aspects and presented the influence that the aspects have on the efficiency and performance of the system.

As (Bösch et al., 2018-05) reports, the high operational cost of ride-pooling services is majorly due to the driver's salary. However, it was also discovered that non-served passenger rides or rejected passenger requests have an underlying cost, which should be considered operational costs. These costs are vital in contributing to the cost of the solution (shift plan) that needs optimization. This thesis proposes a heuristic model to optimize an initial solution containing the driver shifts and breaks schedules, using the demand data from MOIA. The optimization goal is to reduce the costs (driver hours and rejections per hour) by searching for an improved neighbourhood solution with reformed schedules that achieve the desired acceptance rate per hour and improve efficiency. The acceptance rate depends on the number of pooling requests (served rides) upon the number of submitted requests (total requested rides). The developed model is made to iteratively optimize the solution along with each MATSim's iteration, similar to the co-evolutionary algorithm in MATSim. By developing and evaluating a distinctive model that performs in a ride-pooling setting. Additionally, it utilized a fast heuristic algorithm that had not yet been implemented. In order to reduce the computational time as it is an NP-complete problem (Widl and Musliu, 2014), the model was applied to a small study area.

Since my thesis topic relates to shift and break scheduling using Simulated Annealing in the context of a ride-pooling scenario, no relevant literature on the subject has been found. There has been researched literature on scheduling problems, specifically in public transportation (Wren and Wren, 1995; Ciancio et al., 2018), utilizing genetic algorithms to solve them. Despite GA being another heuristic algorithm, it was deliberately not chosen since it could not optimize a single solution. A population-based heuristic requires multiple solutions as inputs and generates multiple solutions as outputs. In MATSim, (Kuehnel et al., 2021-05)'s extension only incorporates a single shift plan for a day's pooling service that consists of all drivers' shifts and breaks. After that, the extensive literature on shift design. (Di Gaspero et al., 2007; Gaertner et al., 2001; Bonutti et al., 2017; Di Gaspero et al.,

2013) and break scheduling (Widl and Musliu, 2014; Beer et al., 2008; Di Gaspero et al., 2013; Widl and Musliu, 2010; Beer et al., 2010) matched what the model sought to achieve, but their algorithms employed different approaches that did not apply to this case. As this thesis looks at a unique optimization framework due to location-specific legal requirements and company-specific workplace constraints, finding specific literature is also challenging. Lastly, literature on Simulated Annealing was predominantly used in a hospital (Kundu et al., 2008; Wong et al., 2014) setting for scheduling nurse shifts, in a university (Norgren and Jonasson, 2016; Abramson, 1991) for course scheduling, and in a general or non-specific field for scheduling personnel (Kletzander and Musliu, 2019; Catoni, 1998; Bailey et al., 1997). Since SA uses several constraint parameters in its algorithm and is problem-specific, there was no exact literature whose values could have been chosen for the model in the thesis. In addition, perturbation techniques were also problem-specific to have inspired the algorithm used in the thesis. In summary, the literature on Simulated Annealing was mainly used in a hospital setting to schedule nurse shifts, a university setting to schedule courses, and a non-specialized setting to schedule personnel. Researchers hardly focused on the transportation sector, except for public transit as mentioned above (Xie et al., 2013) or freight transport (Acuna and Sessions, 2017).

The Simulated Annealing algorithm was preferred over other algorithms for several reasons, including its superior performance, ease of implementation due to not involving complex mathematical formulae, and relative computational speed. A significant reason was that the single-solution-based Monte Carlo method suits our scenario. Additionally, other reasons would be its iteration dependency and improved hill climbing technique for accepting worse solutions to escape local minima.

1.3 Research Question

Based on improving the acceptance rate per hour and reducing driver costs, simulated annealing, a heuristic algorithm, was applied to optimize shift schedules to an initial shift plan. Utilizing MITO synthesized travel demand data for Holzkirchen city from (Zwick et al., 2021; Kuehnel et al., 2021-05) the shift DRT extension of MATSim to simulate the model. As a result, the model should eventually provide an optimal shift plan (solution), based on cost-effective shift schedules, that can meet the requests generated during simulation while ensuring that the rejection rate never goes over 20% at any time. Thus, asking the research question: ***Can a heuristics algorithm like Simulated Annealing optimize shift and break schedules of drivers in ride-pooling services?*** This thesis aims to successfully apply the (Simulated Annealing) method on a small Study Area in Germany, to test its robustness, to find factors that influence the quality of an optimized neighbour solution, and to devise ideas that may lead to faster optimization convergence.

1.4 Thesis Overview

Introduction and Literature Review are covered in the first two chapters, allowing the reader to familiarize themselves with the motivation behind the topic and provide information on the current ride-pooling scenario, the research gaps, the objective of the thesis and reason for the selection of Simulated Annealing algorithm in the model. After that, the thesis reviews academic studies that have discussed and addressed employee scheduling problems in various fields, problems that involve Shift Design Problem and Break Scheduling, explained the advantages and disadvantages of meta-heuristics and explained the concept of Simulated Annealing and how it could be modified for the study's case.

The third chapter discusses the methodology, including terminologies such as shift plan, shift, and break. An explanation of what basis requests are requested and how they affect the objective function follows. The concept of Simulated Annealing was elucidated with an illustration of a flowchart and distinguished various cooling schedules. Finally, encoding and perturbation procedures were demonstrated with pictures.

The results of the three different scenarios are discussed in the fourth chapter and their respective outcomes. Sensitivity analyses were performed on several variables that influenced the algorithm's solution quality and processing time. Additionally, it was hypothesized that secondary heuristics involving regression of supply and demand might help accelerate the run-time.

The last chapter analyzes the optimization model's algorithm results and summarizes its benefits and limitations. Furthermore, the study highlights the contributions made by the study and recommends topics for future research that needs to be conducted to improve the model.

Chapter 2

Literature Review

2.1 Ride-pooling and its challenges

Transport has been instrumental in forging the sharing economy revolution by introducing mobility on demand' platforms, which enabled new modes of mobility like ride-sharing, ride-pooling, and ride-hailing. By using ride-hailing apps, people looking for a particular trip they want to take are matched with drivers willing to serve their transportation needs making it convenient for both parties. A ride-hailing platform can eliminate cash transactions and adjust prices dynamically according to supply and demand without exchanging cash (Shaheen, 2018). Information and communication technologies are used for developing ride-hailing applications that offer reliability, coverage, and operating costs superior to competing modes, such as taxicabs on the street (Rodier, 2018). Likewise, the spread of digital communication technology-assisted in improving dispatching of customers and vehicles led to the development of Transportation Network Companies (TNC) and ride-pooling services that are available on-demand, named by (Shaheen et al., 2016). Traditional ride-sharing programs differ in that hired exclusive drivers offer the erstwhile while the latter is offered by non-contractual drivers, who will not expect to be profitable by participating in these services.

Researchers have focused on ride-sharing services because they have changed mobility and raised some debates. It has been hotly debated whether ride-sharing services may increase traffic congestion. Those who support ride-sourcing argue that it complements current modes in the transportation system, bringing down automobile ownership and reducing traffic congestion. (Li et al., 2016) draw their conclusion from Uber and Urban Mobility Report data that the introduction of (Uber Pool , 2021) reduces traffic congestion in American cities. Critics argue that by providing more accessible and comfortable rides, Transportation Network Companies increase traffic jams by diverting passengers away from space and energy-efficient modes to less efficient ones. A recent report asserts that the industry is around 6 billion miles (Vehicle Kilometres Travelled) to drive in several urban areas throughout the United States (Schaller, 2018).

Several TNCs have gained substantial market shares in the ride-hailing market worldwide, but those companies face stringent regulations in the European and German mobility markets for their business models. Many exclusive ride-sharing services are offered in Germany, such as (Clevershuttle , 2021; Münchner Verkehrsgesellschaft GmbH , 2021; MOIA , 2021). These ride-sharing systems must be simulated before they will be allowed without fleet size restrictions. Before analyzing large-scale and long-term effects, a detailed demand and fleet control model must be developed to capture the current situation (small fleet size) (Wilkes et al., 2021).

Many simulation frameworks and operational strategies have been developed to assess the effects of (pooled) on-demand mobility, mainly in the context of autonomous vehicles. (Zwick and Axhausen, 2020) claims it is difficult to compare different pooling strategies when simulation frameworks and assignment and pooling strategies differ. MATSim by (W Axhausen et al., 2016) was extended to include two of these frameworks. In his Autonomous Mobility on-Demand (AMoD) extension, (Ruch et al., 2020) implemented different pooling strategies and found that in urban environments, the (Alonso-Mora et al., 2017) strategy performs the most efficient in terms of shared mileage and time savings. In 2017, (Bischoff et al., 2017) introduced a second pooling strategy for demand-responsive transport called MATSim's Demand Response Transport extension (DRT). There is a difference between the DRT module and the AMD module, in that the DRT module assigns or rejects requests immediately after submission, whereas the AMD module keeps all requests for a predefined period, and optimizations are made every 30 seconds; re-assignments are also possible. The advantages of DRT outweighed AMoD in terms of VKT and computation time (Zwick and Axhausen, 2020).

On-demand systems were evaluated in some simulations using artificially generated demand, such as scenarios (Zhang et al., 2015; Farhan and Chen, 2018). In the last several years, more and more studies have been looking at real-world scenarios from synthetic populations included in transport models (Zwick et al., 2021). The model demand was defined either by the percentage of previous rides that were pooled or by the mode choice model used (Moeckel et al., 2020). Therefore, there are few studies on taking shift schedule times of drivers into account of these simulations. (Kuehnel et al., 2021-05) developed an extended version of the DRT extension that included shift schedules and break times to study existing services more realistically and to account for operational challenges, but come with their own set of drawbacks. There is a concern that shifts do not necessarily end at the starting location such that the starting location is only determined at the time of vehicle assignment, which may lead to new driver assignments.

2.2 Employee Scheduling Problem

Effective and efficient human resource management is the critical factor contributing to an organization's product or service quality. Considerable improvements in managing labour and its associated cost are directly proportional to reducing a company's expenditure and increasing the productivity of its daily operations. Hence, many industrial sectors, especially the labour-intensive manufacturing and service industry, have realized the potential of full human capital utilization as a crucial factor in the company's strategies to incorporate employee schedules or costs.

This thesis focuses on optimizing the employees' schedules, who have direct contact with customers daily, thereby affecting their satisfaction. Optimization of personnel scheduling, conceptualized by (Edie, 1954) and later formulated by (Dantzig, 1954) in the 1950s, primarily deals with the development of systematic and logical timetables of all staff employees in a particular department, taking into account their staffing requirements, the administration rules of the company, and the terms defined by the labour regulations of the country. It had the drawback of making the combinatorial problem challenging to solve if it allowed large degrees of flexibility.

Since these schedules tend to form the "supply" aspect of a company, one needs to consider their direct consequence on the demand of the product or service offered by the company. Thus, sub-optimal scheduling may necessitate hiring temporary workers, raising costs, while schedules that do not adhere to all regulatory requirements may result in penalties and employee dissatisfaction. The (General Employee Scheduling Problem) is solved by considering a broad range of constraints (Kletzander and Musliu, 2019). The authors also proposed a framework that allows the specification of multiple requirements without introducing new formulations for each variant of the problem. The formulation was specified using an XML format to make it readable for humans and machines alike. Over a specific period, GESP enables employees to schedule shifts, optional tasks, breaks and other activities.

It has become increasingly important to satisfy employee needs in scheduling decisions and consider a fair and equitable distribution of work among employees and their workspace constraints. The literature, therefore, provides a variety of different approaches to solving such constrained and complex workforce scheduling problems, which have been executed in an array of service spheres, including airlines, healthcare systems, law enforcement, and call centers, among others (Ernst et al., 2004), as well as identifying several modules that can be applied to rostering problems. Given the specific needs of different companies, planners and managers of these companies are aided by these consequent diverse models in defining schedules for their employees. In staff demand modelling (SDM) and staff scheduling (SS),

authors of (Ernst et al., 2004) discussed a planning horizon that determines the level of staffing required within constraints and, in particular, monetary constraints in order to achieve the goals of an organization.

According to a new investigation study by (Van den Bergh et al., 2013), which examined numerous articles and categorized problems into their various characteristics were published. Casual, temporary, and full-time employment are frequently governed by contractual constraints, including skill requirements as one of such categories. The scheduling process sometimes includes crew assignments in addition to individual assignments. A typical planning decision is when to schedule tasks, schedule groups, schedule shift sequences, or schedule time. There are many options for arranging shifts throughout the day, such as reserving a specific start and end time based on the shift design. It is commonly known that coverage constraints are divided into hard constraints and soft constraints (Kletzander, 2018). Schedules must adhere to strict rules and non-negotiable constraints, such as labour laws and the number of employees employed each day. The computation usually takes longer under hard constraints than under soft constraints, which indicates rostering rules that are less rigid and should be met wherever possible. There are different ways to handle overstaffing and understaffing.

Another review (Bruecker et al., 2015) places its emphasis on the shift times of employees, as well as their skills. Besides explaining the differences between hierarchical and categorical skill classes, the author discusses skill substitution methods. The paper examines in detail the way different papers define and assign skills. This paper (Glover and McMillan, 1986) introduced (Employee Scheduling Problem), showing that a variety of common concepts can be applied to problems of this type. It offers the advantage of removing limitations such as lack of connections across periods and employee homogeneity. At the same time, it lacks practical application for large-scale scenarios.

Decision support systems can help provide the correct amount of work time to the right employees at a reasonable cost, a fit within a company's budget while attaining a level of contentment for employees at their workplace. It is possible to solve the problems optimally, but numerous combinatorial explosions are associated with them (Laarhoven, Van and Aarts, 1987). As many of these problems are associated with NP-hard problems, solving the problems becomes computationally expensive (Garey and Johnson, 1990). Although not all problems will include various constraints, creating a new specific model and solver for every case is challenging. It would be highly beneficial to have a framework that can be universal in its creation of novel formulations. In such cases, heuristic solver algorithms usually prove the best option. There are general heuristic frameworks such as hyperheuristics that were proposed by researchers (Burke et al., 2013), but, as far as it is known, there is not yet a framework for analyzing employee scheduling problems as a whole.

2.3 Shift Design Problem with Breaks

An alternative form of the ESP, the shift design problem, was introduced in 2001 (Gaertner et al., 2001; Akkermans et al., 2019) that focuses on the time constraint of the problem and where it considers the shifts to be cyclic, i.e. shifts can have more than 24 hours. Additionally, the objective should be to minimize the number of dynamic shifts, regardless of the company be adequately or inadequately staffed. Various studies have demonstrated that shift scheduling is a similar issue. Despite the similarities between shift design problems and shift scheduling, shift design problems differ in several (Kocabas, 2015).

Shift scheduling involves the computation of shifts sizes, the assignment of employees to each shift during a working day and scheduling them. A shift is typically assigned to few employees without understaffing, so breaks are usually scheduled throughout the shift. (Moondra, 1976) did develop a first implicit formula for the shift scheduling problem without considering breaks, though he did not consider shift length. Another distinction is that in shift design, instead of one single day as a planning horizon, it considers several consecutive days. Several authors have discussed shift design problems and have developed many methods to deal with them. These include Tabu Search (TS), SA, integer programming (IP).

For each shift, (Dantzig, 1954) developed an integer variable formulation for set-coverage. Parameters like start time, length, and break times are predetermined, and feasible shifts must be identified and integrated into the model. However, integer variables can be used more when different break options are used. The (Bechtold and Jacobs, 1990)'s Integer programming model differs from the set-covering model concerning implicitly modelled breaks. Every shift is presumed to contain a single break window made up of contiguous planning periods, and the break duration is the same for all shifts. In addition, it cannot schedule breaks within their respective break windows if a significant overlap occurs in their respective break windows, thereby reducing the combinatorial problem size substantially, reducing the time needed to solve it. According to (Jacobs and Bechtold, 1993), it is one of the most critical factors contributing to improved labour utilization if breaks can be placed at the right time. It is flexible to set the break times at any location within a predetermined break window.

Similarly, the formulation of (Aykin, 1996) integer programming model was compared to a similar model developed simultaneously. The IP model approach had fewer variables, whereas the model from (Bechtold and Jacobs, 1990) had few constraints, and Aykin's model was deemed superior. Two other implicit models developed by (Widl and Musliu, 2010) were better than previous approaches, such as Aykin's first model, which involved multiple-break windows with different duration.

Previously, break scheduling was primarily addressed in conjunction with shift scheduling. Problem formulations with a few breaks have been discussed in several different ways. Breaks are scheduled as part of a shift schedule (Dantzig, 1954; Bechtold and Jacobs, 1990; Aykin, 1996; Beer et al., 2010); call centres, hospitals, transportation, and hotels have much more significant problems with break scheduling than the problems formulated in previous shift scheduling studies. Further, the lunch break schedules became more complex as time windows for lunch breaks or time restrictions for breaks increased, which enlarged the search space for optimizing such schedules (Musliu et al., 2004). (Rekik et al., 2010) fractionable breaks are introduced, where shift breaks following the technique are restricted to a lower and upper limit but lack robustness as impacts of multiple breaks are not considered. (Widl and Musliu, 2010) evaluated the problems with break scheduling when staffing requirements and shift selection are given. Their objective function minimizes deviations from demand in their model. The placement and length of breaks are subject to several time constraints. An employee may not schedule a break too soon after starting the shift; breaks must be scheduled appropriately concerning the end of the shift.

For this problem, (Beer et al., 2010) developed a heuristic based on minimum conflict, which has since been used by supervision personnel in a real-world setting. The main task is to schedule break times efficiently to existing shifts, assuming that the shift assignment has already occurred. They solved the scheduling problems in two phases, first by assignment of shift and then by scheduling breaks, thereby increasing run time. A real-life benchmark example with more than 10 breaks was also introduced by him, which was later on improved by memetic algorithms presented in (Widl and Musliu, 2010). The formula combines a heuristic approach with staffing demand coverage and ergonomic criteria to find a solution that diminishes the aggregated amount of work regulation violations.

2.4 Heuristics

(Osman and Laporte, 1996) states that "The term meta-heuristic formally refers to an approach that guides a subordinate heuristic towards finding near-optimal solutions using multiple iterative strategies for investigating and leveraging the search space as well as learn tactics to compile all the data collected in order to search near-optimal solutions". Since NP-complete problems are complex and demanding despite several attempts to solve them mathematically, heuristic methods are the only viable option when solving such complex problems. Typically, heuristic methods are evaluated by the output quality and the time taken to reach this outcome. Regardless of the considered optimization problem, a compromise must be found since these two criteria are opposing. Scheduling problems that are either too complex or difficult to solve using an exact solution method has been solved mainly through heuristics. Methods based on heuristics are typically robust and can produce reasonable "good"results, albeit not ideal results. Heuristics cannot guarantee the quality of a solution or its time efficiency. As heuristics can be categorized into"general"heuristics or meta-heuristics, first coined by (Glover, 1986), are often realized for obtaining approximate optimal results and are frequently used for various types of combinatorial optimization problems, whereas "tailored"heuristics or specific heuristics are unique to a specific problem (Laarhoven, Van and Aarts, 1987). These meta-heuristics rely on nature for inspiration (derived from physics, biology, or ethology); stochastic elements are involved in some of them, and they need continuous tuning of parameters to fit the problem.

Researchers have focused significantly on meta-heuristics over the past few years because of their ability to reach optimality at a faster rate. A few steps can be outlined that have characterized meta-heuristics throughout history, among the pioneer contributions (Kirkpatrick et al., 1983)'s Simulated Annealing method. (Glover, 1986) proposed the Tabu Search algorithm, and the first genetic programming patent was filed and published in (Koza and Koza, 1992). The Genetic Algorithm was published by (Goldberg 1989). The evolution of meta-heuristics is undoubtedly linked to the continuous improvement of computing power and the emergence of massively parallel architectures. Meta-heuristics are CPU-intensive, but these improvements relativize their costs (Talbi, 2009).

Optimizing processes require a balance between diversification, a method of exploring the search space, and intensification, which can be achieved by exploiting the neighbourhood (Blum and Roli, 2003). Utilizing the accumulated search experience is essential to increase the intensity of the search. Differentiation between the existing meta-heuristics appears mainly in how each of them tries to achieve this balance (Birattari et al., 2001). Meta-heuristics can be categorized according to many factors. For example, consider the classification of meta-heuristics based on their features, whether the search path they follow, the way memory

is used, how neighbours are explored, or the number of solutions repeated from one iteration to another. Some scholars and researchers classify and categorize meta-heuristics according to their size of solution into single-solution or multiple-solution based meta-heuristics (SSB, MSB), as an important distinction (Talbi, 2009).

In general, SSB meta-heuristics tend to be more exploitative, while MSB meta-heuristics tend to be more exploratory. Meta-heuristics suffer from several drawbacks. They cannot reduce the search space or define stopping criteria. In addition, scheduling problems with highly constrained elements pose a challenge to meta-heuristics because infeasible ones often separate feasible regions. Mathematics programming and exhaustive search are alternative approaches that, although they will ultimately realize the optimality of a given problem, will rarely do so within a reasonable time frame since the resources of typical machines are finite (Burke et al., 2010). Thus, meta-heuristics are applied to general optimization problems when finding near-optimal solutions in the shortest time.

Due to this thesis' use case, the study focuses on SSB Meta-Heuristics, because of being the single initial solution optimization, tend to discover a single optimal solution. The exploration process can be traced using such meta-heuristic methods called trajectory methods. The more commonly used trajectory search methods are Tabu Search, Simulated Annealing, variable neighbourhood, and guided or iterated local search.

2.5 Simulated Annealing

Among many meta-heuristic algorithms, the (SA) algorithm is generally considered the oldest and one of the first algorithms to include mechanisms for bypassing local optima through the acceptance of worse solutions. The name derives from a metallurgical process called annealing. Material is then subjected to high temperatures, changing its physical and chemical properties and making it more ductile. After maintaining the temperature for a specific time, the temperature is gradually lowered. The goal in both cases is to transform the material into a well-ordered solid-state with low energy (thereby avoiding the instability found in local minima), as metals are heated to achieve a well-ordered solid-state with low energy. If the temperature is high, SA is more likely to discover new territory; it is more likely to exploit existing territory at a lower temperature.

Exploring the search space is achieved by introducing a fictitious initial temperature $\boldsymbol{T}$ into the acceptance equation that decreases with the search progress; the objective function is then minimized. It is common in annealing schedules to keep the initial temperature as a function of the iteration number, therefore

$$T = \frac{initial\_temperature}{iteration\_number + 1}$$

Temperature calculation avoids division by zero errors by adding one to the iteration number as iterations usually start at zero. Reducing temperature is similar to how the energy of the material is minimized. At $\boldsymbol{0}$ K, crystallized particles occur at their fundamental (lowest) energy level. An ideal crystal is considered a global solution, whereas mass deformations represent a local optimum. The simulation method was first found in statistical mechanics in the Metropolis algorithm (Metropolis et al., 1953), who came up with it first and firstly applied in (Kirkpatrick et al., 1983).

Applied to optimization problems, Simulated annealing (SA) is a stochastic process that permits conditional acceptance of the worst solution. In this strategy, the objective is to escape local minima and attain convergence to avoid potential suboptimality. After a solution and a temperature parameter $\boldsymbol{T}$ have been established, the algorithm gets started by creating a new neighbouring solution from the initial solution (either randomly or using a heuristic) and then keeps generating these subsequent solutions with every iteration or decrease in temperature. During each iteration, an alternative solution is randomly chosen near the current one, keeping in mind that whenever possible, the alternative lowering the energy of the substance (improvement of the objective function) is preferred. However, it is essential to consider that the probability of adopting inferior solutions decreases over time as the objective function deteriorates with the temperature decrease. Boltzmann distribution and Metropolis function describe the likelihood $\boldsymbol{P}$ of an inferior solution being adopted and given by the ratio

of the difference in objective functions of the current and new solution to the ratio of the difference in objective functions ***f(x)***, x being the current and new solution, and system's temperature.

$$P = \frac{f(current) - f(new)}{temperature} < r$$

Where ***r*** ranges from ***0*** to ***1*** generated at random, thus, as ***T*** decreases during searching, poor solutions have a higher likelihood of being accepted early in the search and a lower likelihood of being accepted later in the search. Ideally, a high temperature at the start of a search will aid the algorithm in finding the global optima, and a low temperature later will allow it to refine its search for the global optima.

Similar to the paper in (Wong et al., 2014), constraints related to scheduling can be categorized into hard and soft. Any feasible model must satisfy a constraint known as a hard constraint generally governed by legal and administrative regulations; In contrast, a soft constraint can be violated. However, it incurs a penalty on the objective function (the more significant the penalty, the greater the violation). They are usually connected to the quality of work, service quality, and employee satisfaction. Among the advantages of this heuristic are that it can be implemented relatively quickly, that it can be administered to theoretically any combinatorial optimization problem, and it can be combined with other algorithms. (Elmohamed et al., 1997) It has some drawbacks, though it is still a robust technique. For good results, the perturbation and all of the tune-able parameters (such as the cooling rate) should be carefully selected, the runs can take a lot of computer time, and many runs may be needed.

Chapter 3

Methodology

A simulation annealing technique is implemented in the thesis to optimize driver shift (and break) schedules to reduce the number of driver hours and the non-served rides maintaining an acceptance rate of served rides at 80%. The algorithm is written in java and simulated within the open-source java application, (Multi-Agent Transport Simulation , MATSim), with the help of its (Demand Responsive Transit , DRT) extension developed by (Bischoff et al., 2017). The DRT extension that was modified to include driver shift (and break) (Kuehnel et al., 2021-05) schedules to make ride-pooling simulations more real. In comparison, the new shift and break extension allow the temporal constraint for the drivers that determine their availability. With the help of (Microscopic Transportation Orchestrator , MITO), was developed by (Moeckel et al., 2020), generated the travel demand or ride-pooling requests for Holzkirchen (Zwick et al., 2021).

Requests for ride-pooling services are simulated using the DRT extension and assigned to available drivers. A vehicle matching algorithm searches for the best vehicle to handle the trip request based on the pick-up and drop-off location coordinates, the maximum wait time, and the maximum waiting time for the passengers who wait outside and inside the vehicle. A request inserted into a scheduled route is usually assigned to a vehicle that incurs the smallest amount of additional operating time.

A ride-pooling service simulated using the DRT extension makes it possible to centrally dispatch and match on-demand trip requests from passengers to the available drivers. The trip requests attach pick-up and drop-off locations from the demanding passenger. The dispatching algorithm searches for drivers near these passengers who do not violate their maximum wait and detour time. If trip requests are accepted, keeping the travel time delay low for all onboard customers, routes are assigned to that driver to transport all accepted passengers. The route assignments are done by the insertion algorithm, an algorithm in the extension that keeps travel time within a threshold. The travel time exceeding a certain threshold leads to the rejection of submitted customers' trip requests. A vehicle driver's route assigned to a customer

cannot be changed thereafter.

Additionally, trips can be rejected based on: vehicle seating capacity, vehicle's availability (driver shift), exceeding the maximum waiting time, or exceeding the maximum detour time (Bischoff and Maciejewski, 2020). The extension uses two types of service operations (Zwick and Axhausen, 2020): door-to-door, where all pick-ups and drop-offs are in the exact coordinate location as requested by customers, and stop-based, where all customers access and egress to an adjacent stop to get to the ride-pooling vehicle. According to (Bischoff et al., 2019), door-to-door operations are better in terms of travel time during non-peak hours. For the same reason, only door-to-door operations were considered in the shift optimizer model. A draconian scenario also considered in the model is a scenario where all car trips change to DRT trips, as opposed to a laissez-faire scenario where all DRT modes are autonomous.

With the help of (Kuehnel et al., 2021-05)'s extension, ride-pooling services are now studied more efficiently and effectively as it takes schedules of drivers into account. All daily driver shift (and break) schedules packed into an XML file are the initial solution, an input for MATSim simulation. These drivers' shifts (and break) schedules govern the availability and non-availability of drivers, resembling a real-world scenario. A shift dispatching algorithm is put into the extension that assigns these shift (and break) schedules to driver agents on ride-pooling vehicles. Trips requests are now served only when a driver is available according to his shift (and break) schedule, rejecting all other requests. A shift, in turn, is determined by its start and end time. If the shift (and break) schedule's duration is 8 hours, breaks are also scheduled within them. During breaks, drivers are not available, and they do not serve requests during this time. Given the different times of different driver schedules, the demand may not be served efficiently, leading to many rejections of trip requests.

This thesis proposes a method to optimize the driver shift (and break) schedules to improve ride-pooling service's performance and efficiency, leading to minor rejections and low operational costs. In the proposed methodology, a heuristic algorithm (Simulated Annealing) applied in thesis mode brings improvements to these schedules. The reasons why Simulated Annealing is implemented in the model:

1. The single-solution-based algorithm takes a single initial solution and produces a single optimized final solution. It is important as the (Kuehnel et al., 2021-05) made extension functionally works with single solution inputs.

2. The algorithm can create an objective function or cost to measure the quality of all driver schedules based on their rejection rate and working hours. A low-cost solution will have good schedules with minor rejections, and their rejection rates are well below the desired value.

3. The algorithm uses stochasticity to manipulate the schedules using different perturbing operations, leading to different costs. Randomness helps search global optimums, whereas greedy algorithms may even lead to failed searches.

4. The algorithm iteratively tries to search for globally optimal quality (low cost) neighbor solutions using an improved hill-climbing that accepts worse solutions in early iterations to escape local optima.

3.1 Study Area

Located in Bavaria, a federal state in Germany, Holzkirchen is the largest populous town in the Miesbach district (Landkreis), with over 16,000 (2008) persons. Situated at a convenient location with excellent highway and rail accessibility, Munich District is also well connected and is accessible by the S-Bahn (S3) that comes every 20 minutes. Holzkirchen and surrounding areas were chosen as the study area for the thesis research as the city's population was the lowest among the other areas researched (Zwick et al., 2021). The main reason for this scenario's selection was due to the computational time of the algorithm. To test an algorithm's robustness and effectiveness, it is always better to start with small case scenarios (Gopakumar et al., 2018). Therefore, Holzkirchen city's scenario has a population of 16750 inhabitants (für Statistik, 2019) and a service area of 34.1 km2. A draconian operation changing all mode trips to only ride-pooling trips is considered for simulation within the service area. Figure 3.1 shows Holzkirchen's network area map, and the zoomed-in part shows the city center of Holzkirchen to give an idea of the extent of the service area.

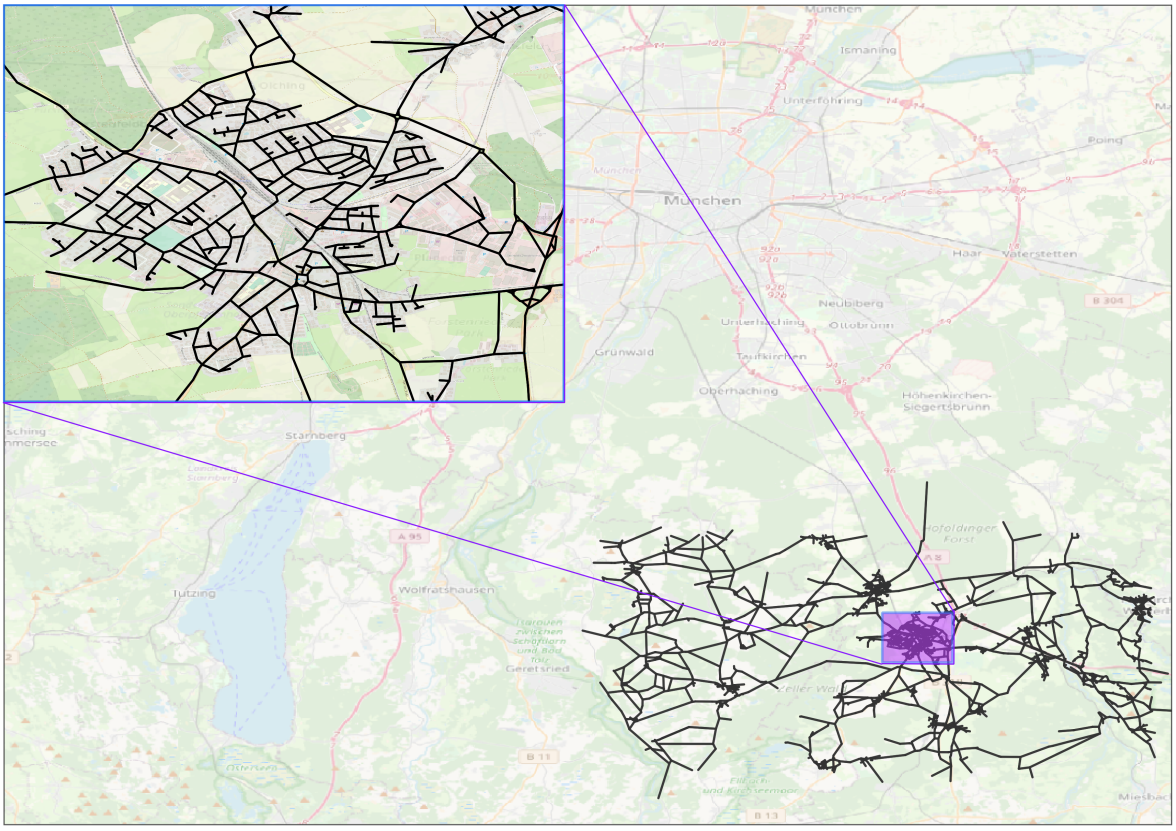


Figure 3.1: Study Area Network - Holzkirchen and surrounding areas

3.2 Travel Demand - submitted requests per hour

The static travel demand for Holzkrichen's scenario was synthesized using the open-source simulation model, MITO (Moeckel et al., 2020). MITO uses a modified four-step approach to generate agent-driven trips, supported by two trip- and agent-based models. The population synthesizer, an IPU-based method, was the input (Moreno and Moeckel, 2018). At the same time, trips with six different purposes, destinations, modes, and times of day were generated as output. MATSim is a simulation framework that iteratively and co-evolutionarily improves the transport agent's score consisting of daily activity plans. The score increases by working (doing activities) and decreases by traveling (not doing activities). The score improvement ensures that agents should work more than travel. So, after every iteration, MATSim changes the agents' plans in different routes or modes, leading to an equilibrium where no plans are changed as final scores do not improve any further. For this thesis, MATSim (W Axhausen et al., 2016) was used only for route choice to assign on-demand requested trips to the service network. MITO aided in mode choice to mimic the draconian case of only ride-pooling trips within the service area. Thus, MITO and MATSim generate the static demand for the shift optimizer model (Zwick et al., 2021). Even though ride pooling requests are arbitrarily requested during the service, the thesis uses a new variable **submitted requests per hour** for the static travel demand to keep the cost within the exact temporal resolution. Since the cost of the solution also incorporates the hourly salary rate of drivers that depends on the **active shifts per hour**.

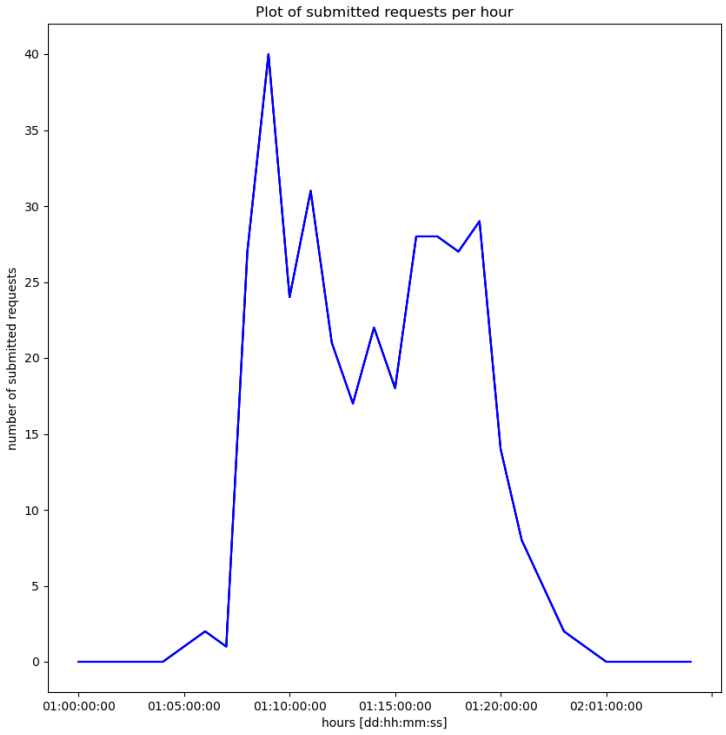


Figure 3.2: Plot of submitted trip requests per hour

3.3 Travel Supply (Shift Plan) - active shifts per hour

All the daily driver shift (and break) schedules are combined in an XML format, directly read by MATSim's shift and break extension. The shift plan is the model's supply that serves the ride-pooling **submitted requests per hour** demand for the Holzkirchen scenario. As a human and machine-readable programmable format, XML allows for structuring large amounts of specification information. Since, (Multi-Agent Transport Simulation , MATSim) is programmed in (Java Library), the shift plan is converted into a DrtSolution or an Individual object. The DrtSolution object is used by the extension, whereas the Individual object is used in the model developed for this thesis. These objects are interchangeable, such that the DrtSolution can be converted to the Individual and vice versa. This object conversion in the code is necessary to ensure that the original code for the shift and break extension is not manipulated. However, to avoid confusion, **solution** or **shift plan** is frequently used throughout the thesis to refer to the Individual object or the DrtSolution object, containing the driver (shift and break) schedule. It is evident from the screenshot of 10 driver shift (and break) schedules in the figure below 3.3. Description:

- *<shifts>..</shifts>*: Shifts with "s" is a list of driver shift(s).
- *<shift>..<shift>*: Shifts without "s" is one single Shift defined here in subsection 3.4.1
- *<break../>*: Break is the break corridor defined here in subsection 3.4.2

```
<shifts>
    <shift id="0" start="14400" end="45000">
        <break earliestStart="28800.0" latestEnd="32400.0" duration="1800.0"/>
    </shift>
    <shift id="1" start="14400" end="45000">
        <break earliestStart="28800.0" latestEnd="32400.0" duration="1800.0"/>
    </shift>
    <shift id="2" start="14400" end="45000">
        <break earliestStart="28800.0" latestEnd="32400.0" duration="1800.0"/>
    </shift>
    <shift id="3" start="14400" end="45000">
        <break earliestStart="28800.0" latestEnd="32400.0" duration="1800.0"/>
    </shift>
    <shift id="4" start="14400" end="45000">
        <break earliestStart="28800.0" latestEnd="32400.0" duration="1800.0"/>
    </shift>
    <shift id="5" start="10800" end="32400">
        <break earliestStart="19800.0" latestEnd="21600.0" duration="1800.0"/>
    </shift>
    <shift id="6" start="57600" end="86400">
        <break earliestStart="72000.0" latestEnd="75600.0" duration="1800.0"/>
    </shift>
    <shift id="7" start="57600" end="86400">
        <break earliestStart="72000.0" latestEnd="75600.0" duration="1800.0"/>
    </shift>
    <shift id="8" start="57600" end="86400">
        <break earliestStart="72000.0" latestEnd="75600.0" duration="1800.0"/>
    </shift>
    <shift id="9" start="57600" end="86400">
        <break earliestStart="72000.0" latestEnd="75600.0" duration="1800.0"/>
    </shift>
    <shift id="10" start="57600" end="86400">
        <break earliestStart="72000.0" latestEnd="75600.0" duration="1800.0"/>
    </shift>
</shifts>
```

Figure 3.3: Shift Plan in XML format

3.4 Shifts and Breaks

3.4.1 Shift

A Shift is made of:

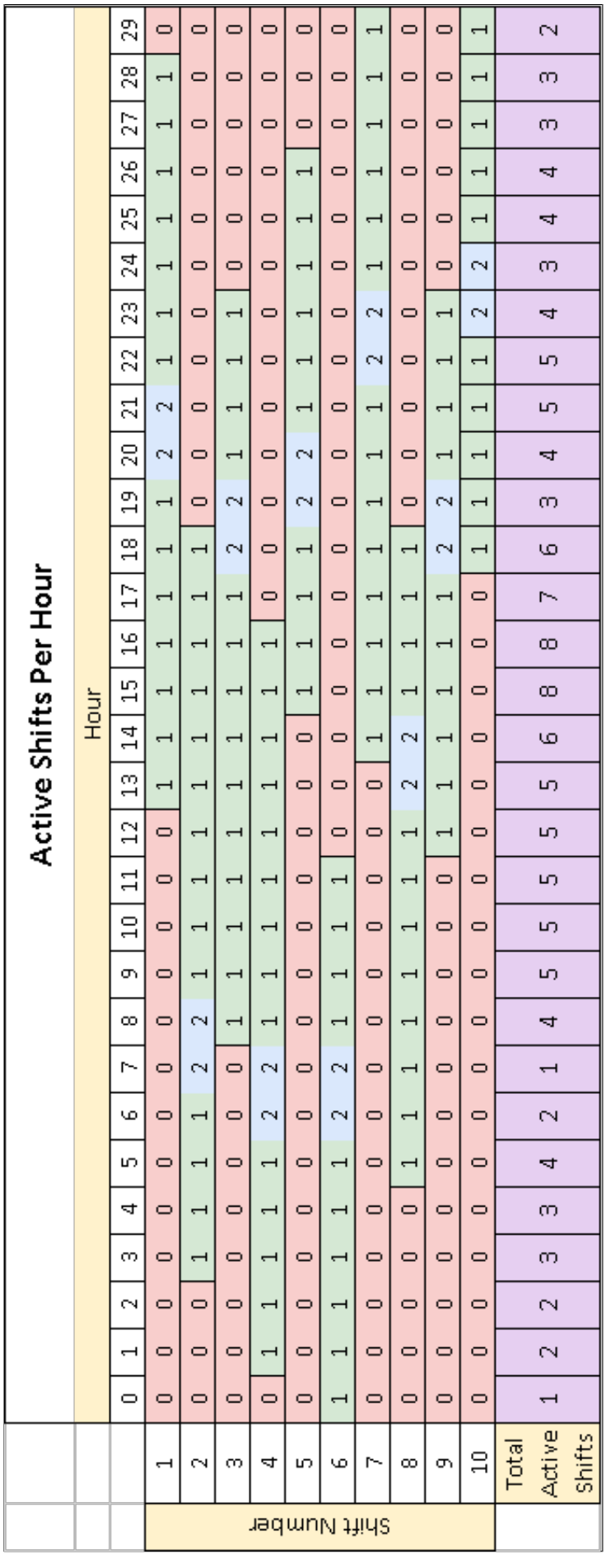
- **id**: A unique identification value that distinguishes every shift
- **start**: Time at which drivers **start** working
- **end**: Time at which drivers **stop** working

A shift is defined by the period of work a driver executes by picking up and dropping off customers to serve customers with ride-pooling ride requests. Shifts have pre-decided *start* and *end* times that determine the duration of their working hours. A driver is only paid his hourly salary during this time. Ride-pooling passenger requests are rejected beyond this time for the particular driver. A shift may or may not have a break scheduled with them based on the number of hours a driver works. Shifts with short duration may not have breaks scheduled but will consist of break corridors. Break times are assigned within a particular range of time, known as the Break Corridor. According to German Labour Laws (für justiz und verbraucherschutz, 2021), two types of shift timings has been considered in this thesis:

- 8 hour shift schedule with 30 minutes break
- Less than 8 hour shift schedule with no break

It is essential to note that a driver shift (and break) schedule's length must be at least 5.5 hours due to the legal regulations, so all shifts considered in the optimization model are also modeled similarly. Shifts always *start* and *end* within the START_SERVICE_TIME and the END_SERVICE_TIME fixed by the modeler. Therefore, different drivers may have different shift and break schedules overlapping each other. Since the optimization model only takes the shift plan as the input for simulation run, hence it is essential to understand the terms that are crucial to objective function calculation:

1. **Active Shifts**: Driver's active working period is known as active shift. Such that drivers do not have breaks scheduled at this time. To calculate the value of active shifts per hour parameter, refer to figure 3.4, describing the hourly summation of all drivers actively working to serve on-demand requests. Active shifts are "**1**" in the encoding given in green. The summed up value of all"1s"gives active shifts per hour in purple.
2. **Inactive Shifts**: The drivers do not work during this time. These periods do not contribute to the solution or the objective function. It can be represented as "**0**"in the encoding colored in red.



Active Shifts Per Hour

Shift Number	Hour 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	2	2	1	1	1	1	1	1	1	0
2	0	0	0	0	1	1	1	2	2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	1	1	1	1	0	0	0	0	0	0
4	0	1	1	1	1	1	2	2	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	2	2	1	1	1	1	1	1	0	0	0
6	1	1	1	1	1	1	2	2	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	2	2	1	1	1	1	1	1
8	0	0	0	0	0	1	1	1	1	1	1	1	1	2	2	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	2	2	1	1	1	1	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	2	2	1	1	1	1	1
Total Active Shifts	1	2	2	3	3	4	2	1	4	5	5	5	5	5	6	6	8	7	6	3	4	5	5	4	3	4	4	3	3	2

Figure 3.4: Example: active shifts per hour (3.4) (LinkedHashMap) for a Shift Plan with 10 shifts

3.4.2 Break

The relief period when drivers rest or eat at the hubs or in-field facilities. No pooling request will be served during this time. Scheduling breaks and assigning them within their defined time frame called the break corridor is necessary. It is only possible to pick up and drop off customers before and after the end time of breaks. Similar to shifts, break corridors are also characterized by *earliestStartBreakTime* and *latestEndBreakTime*, which, as the name suggests, are bounding parameters. The model has fixed the duration of break corridors at 1 hour or 3600 seconds, like in (Kuehnel et al., 2021-05), so breaks can only be of $1/2$ hour or 1800 seconds and be assigned to the first half or second half of the break corridor. The break corridor determines the amount of freedom a scheduled break may have. In reference to the figure 3.4, a Break Corridor is made of:

- **earliestStartBreakTime**: It is the earliest possible time at which a break can be assigned. Before this time, a break can not be assigned and is given as the starting blue colored "**2**" in an encoded shift.
- **latestEndBreakTime**: It is the latest possible time at which a break can be scheduled. Beyond this time, no break is scheduled. Given as the ending blue colored "**2**" in an encoded shift.
- **duration**: It is the time length of breaks. Fixed at 1800 seconds.

3.5 Travel Price - rejections per hour

Since the extension can serve multiple requests simultaneously, it is handled by an abstract dispatch algorithm called an optimizer (Bischoff et al., 2016). Firstly, When a user submits a ride-pooling request, the optimizer will dispatch the request and assign it to one of the driver vehicles closest to the requesting user with the least travel time, both in terms of pick-up time and drop-off time. Users who do not meet the specific threshold of waiting and detour times are rejected, and accepted onboard users are given priority by the dispatching algorithm such that they are not removed from the system. In the second step, the user compares the travel cost of all accepted vehicles and requests for the vehicle with the lowest traveling cost. Requests that are not accepted by vehicles or users will appear in the system as "rejected".

In the optimization model, maintaining the exact temporal resolution (hourly) utilizes the variables rejections per hour and rejection rate per hour to calculate the quality of the solution in terms of cost. The object is generated after each MATSim iteration to track all the submitted and rejected requests. The rejection rate is computed by the rejected and submitted requests ratio. The algorithm in the study aims to achieve DESIRED_REJECTION_RATE of 20% or 0.2. The following can cause on-demand submitted requests to be rejected:

- vehicle capacity - A request may be rejected if the vehicle can not accommodate more passengers
- vehicle availability - Rejection due to non-availability of close vehicles
- maximum wait time - Longer waiting times than the threshold for pick-up passengers
- travel time (either in pick-up of drop-off) - Rejection due to large travel times to destination

3.6 Simulated Annealing

Solving combinatorial optimization problems using such a method was first applied and introduced by (Černỳ, 1985; Kirkpatrick et al., 1983) respectively. It is a heuristic algorithm that has been in existence since the 1980s for solving NP-hard optimization problems. Simulated annealing originates from experiments done by (Metropolis et al., 1953). To imitate thermal equilibrium with a fixed temperature based on the results of their studies on different optimization techniques, researchers have created a probabilistic-based technique algorithm to minimize the energy of the system's state. In a physical system, the position of each atom describes the energy state of the structure. After selecting a starting state, Metropolis generated successive states of the system using the Monte Carlo method. A new state is obtained whenever an atom engages in an infinitesimally small random movement. Let us assume **δE** as the energy difference caused by a disturbance of this kind. In the case of a decrease in system energy (**δE** $<$ ***0***), the neighbor state is accepted. On the other hand, if (**δE** $>$ ***0***), a specific probability determines if it is accepted or not. The acceptance probability is represented as **P** and characterized by this exponential Boltzmann acceptance equation:

$$P = \exp(\frac{-(E_{new\_state} - E_{current\_state})}{T}) = \exp(\frac{-\delta E}{T}) \tag{3.1}$$

where **T** is the instantaneous temperature of the system and $\mathbf{E}_{\textbf{current state}}$ being the energy of the current state, $\mathbf{E}_{\textbf{new state}}$ is the energy of new neighbor state. A random value is generated, let's say **r** [0,1] at each temperature cycle, that will help to decide whether a new state with a lower energy is accepted or not. The state is accepted if **r** is less than or equal to probability equation 3.1, otherwise the state is rejected.

Table 3.1: Acceptance probability cases

Case	Description
P $>$ r	Accept the new state
P $<$ r	Reject the new state

Since **P** depends on **T**, due to exponentiality, high **T** will make the fraction smaller bringing **P** closer to **1**. Therefore, if the temperature is high early on, it is more likely that the algorithm will accept high energetic states or any new state for that matter. The randomness **r** along with acceptance of worse states initially will ensure that the algorithm escapes local minima and reaches global minima. Lower temperatures make it more difficult for a high energetic state to be accepted. This type of rule of probability is used repeatedly while lowering the temperature by Metropolis to lead the system structure to thermal equilibrium or the state of lowest energy. Thus it became Simulated Annealing, a meta-heuristic technique used in

solving complex problems for optimization.

In order to comprehend the acceptance probability formula, for example, the objective is to optimize or minimize the cost of a solution. Suppose an initial solution that costs 100€, an initial temperature of 1000, and a new neighbor solution that costs 180€. As it is known that temperature should decrease in every iteration, let us also assume that temperature decreased to 20 in the last iteration. For the first iteration, the acceptance probability P would be0.92, increasing the likelihood to accept this high-cost new solution. At temperature 200 and iteration 10: the probability of acceptance of the worse solution goes down to 0.67, lowering the chances of acceptance; in the last iteration at 100, where the temperature drops to 20, the probability also drops to 0.018. This method ensures that with time the method will lead to a minimized cost solution.

Since the algorithm has been described for a general use case, it has been applied slightly differently in the thesis. For simplicity, let's call it the "**shift optimizer**"model. Solution or Shift Plan is analogous to state in the system, and its energy is the cost of the solution. The current state is the accepted solution, and the new state is the perturbed current solution.

So, firstly the shift optimizer initializes a reference solution called the **accepted solution**, which is a copy of the **current solution**, which is the copy of the initial shift plan that is going to be optimized. The reference solution is the one that stores the current low-cost solutions. The costs of both the **current solution** and **accepted solution** are initially kept at positive infinity to ensure that the algorithm starts minimizing after the first iteration. Before starting the algorithm, the initial temperature, the cooling scheme, and the number of iterations should also be decided. The model was calibrated by configuring parameters and running the model with different initial shift plans, temperature, cooling rates, or iterations.

Secondly, after the initializations, the current solution is simulated in MATSim iteratively using the shift and break extension (Kuehnel et al., 2021-05) that produces the rejected trip requests data from the simulation. The DRT simulations use the ride-pooling demand data, generated by MITO and based on **submitted requests per hour**, to produce ride-pooling trip requests. If the travel supply **active shifts per hour** is not enough for the simulation framework to meet the demand, the price of the system increases, inducing rejected trip requests. This rejected data is transformed into the **rejections per hour** and **rejection rate per hour** variables to calculate the cost of the solution (objective). The cost is a function of active shifts per hour, rejected requests per hour, and rejected rates per hour. It is important to mention that if the travel supply is more than enough, the cost of the solution will still increase, making the solution less optimal. Bringing down the cost will help the shift optimizer find the minimized solution. Therefore, to make that happen, **accepted solution** keeps track of the low cost simulated **current solution**. The **acceptance solution** becomes the

current solution whenever the current solution's cost is low. However, in the early iterations, a high-cost current solution is also accepted due to the model's high-temperature value. This Metropolis' acceptance probability equation ensures that only a global minimized solution is found ultimately. In the next step, the algorithm perturbs the current solution generating a new neighbor solution. The perturbations are essential to explore the space of neighbor solutions. This perturbed current solution is simulated in the next MATSim iteration, producing a new cost. As mentioned before, in this new iteration, the accepted solution accepts or rejects the new neighbor solution depending on its cost.

Lastly, the steps are repeated in every iteration till the last iteration has reached. The disadvantage here is that the SA algorithm does not know how many iterations are needed to find the minimized solution finally. Therefore, it is assumed that when the cost is not improved after a while, optimality has reached, and the accepted solution is the optimized solution that has the DESIRED_REJECTION_RATE and has the reduced number of driver shifts.
Characteristics of a general Simulated Annealing (SA) algorithm:

1. It is local Search method that is capable of reaching global minima and not getting stuck in the local minima (Kirkpatrick et al., 1983)
2. No complex mathematical functions involved in the algorithm (Osman and Laporte, 1996)
3. It is an improved version of the hill-climbing algorithm, where it uses an iterative approach to find new neighbor solutions, even worse solutions (to help escape local minima) (Nikolaev and Jacobson, 2010a)
4. It solves combinatorial optimization problems easily where the solutions are in the discrete search space (Garey, 1979)
5. It is a Markov Chain Monte Carlo method, that is, it uses stochasticity to perturb solutions (Nikolaev and Jacobson, 2010a)
6. It is a Single Solution-Based (SSB) algorithm that optimizes a single solution (Banchs, 1997c)
7. It guarantees a convergence in infinite time (Nourani and Andresen, 1998)
8. It has a faster convergence rate than other heuristics - GA and TS (Sinclair, 1993)

Authors (Aarts and Van Laarhoven, 1985; Nikolaev and Jacobson, 2010b) state that the SA algorithm stays more time in the low-temperature period than in the high-temperature period slowing down the total run time. The success of SA is attributed to its fast run-time, excellent solution quality, easy application, applicability, and flexibility. Despite this, creating a

practical algorithm for a given problem is not always trivial, as the implementation is easy and flexible; The efficiency of SA depends mainly on an excellent objective function and a cooling schedule with a good speed.

Neighbor states of a current system include all the possible states that can be acquired by the slight metallurgical changes of the system, similarly in the shift optimizer model, a solution is changed by moving shifts (start and end) timings, moving break corridor (earliestStartBreakTime and lastestEndBreakTime) timings, increasing the shift (and break) schedule's duration, decreasing the shift (and break) schedule's duration, inserting a new driver shift (and break) schedules, or by removing existing driver shift (and break) schedules. This process of generating a new neighbor state is called Perturbation.

To find new solutions in the coding space, encoding of a solution to a genotype becomes helpful, as the meta-heuristics algorithms usually operate on two types of spaces, coding space (Genotype) and solution space (phenotype) (Kumar, 2013). Phenotype describes their outward appearance in the initially kept format (read: shift plan in XML format), and Genotype presents the encoded format used in the objective function to calculate the solution's cost quickly. Even though MATSim, can read the Phenotype (XML) format, Genotype (Encoded) format is required to calculate active shifts per hour and perturb for new neighbor solutions. Overview of the implementation (figure 3.5):

1. Set the model parameters, including initial temperature, cooling pattern (schedule), and the number of iterations. Initialize the current or accepted solution to the initial shift plan that needs to be optimized and the solutions' costs to positive infinity. In the first iteration, the current solution is simulated in every iteration in MATSim where it a system is created to satisfy the demand (submitted requests per hour) with the supply (active shifts per hour) increasing the price (rejections per hour and rejection rate per hour) of the system. The algorithm's objective (cost) function utilizes all the hourly attributes to determine the solution's cost in every iteration.

2. The algorithm produces a new neighbor solution by perturbing (changing all or some driver schedules) the current solution randomly. Now the current solution becomes the perturbed solution simulated in every iteration subsequently. Whenever the new current solution possesses a lower cost than the accepted solution, the shift optimizer accepts it either by becoming the current low-cost solution or accepts it with the probability P. If the P is greater than the random value [0,1], then it is accepted. PS: In the first iteration, the accepted solution becomes the current solution due to the positive infinity cost of the accepted solution. Later, whenever the cost of the new current solution is lower than the previously accepted solution cost, they are accepted or rejected based on temperature value and random probability.

3. The above steps are repeated till the shift optimizer reaches the last iteration. The model parameters, cooling schedule, initial temperature, and the number of iterations have a vital contribution in finding a solution with the lowest cost in a short time.

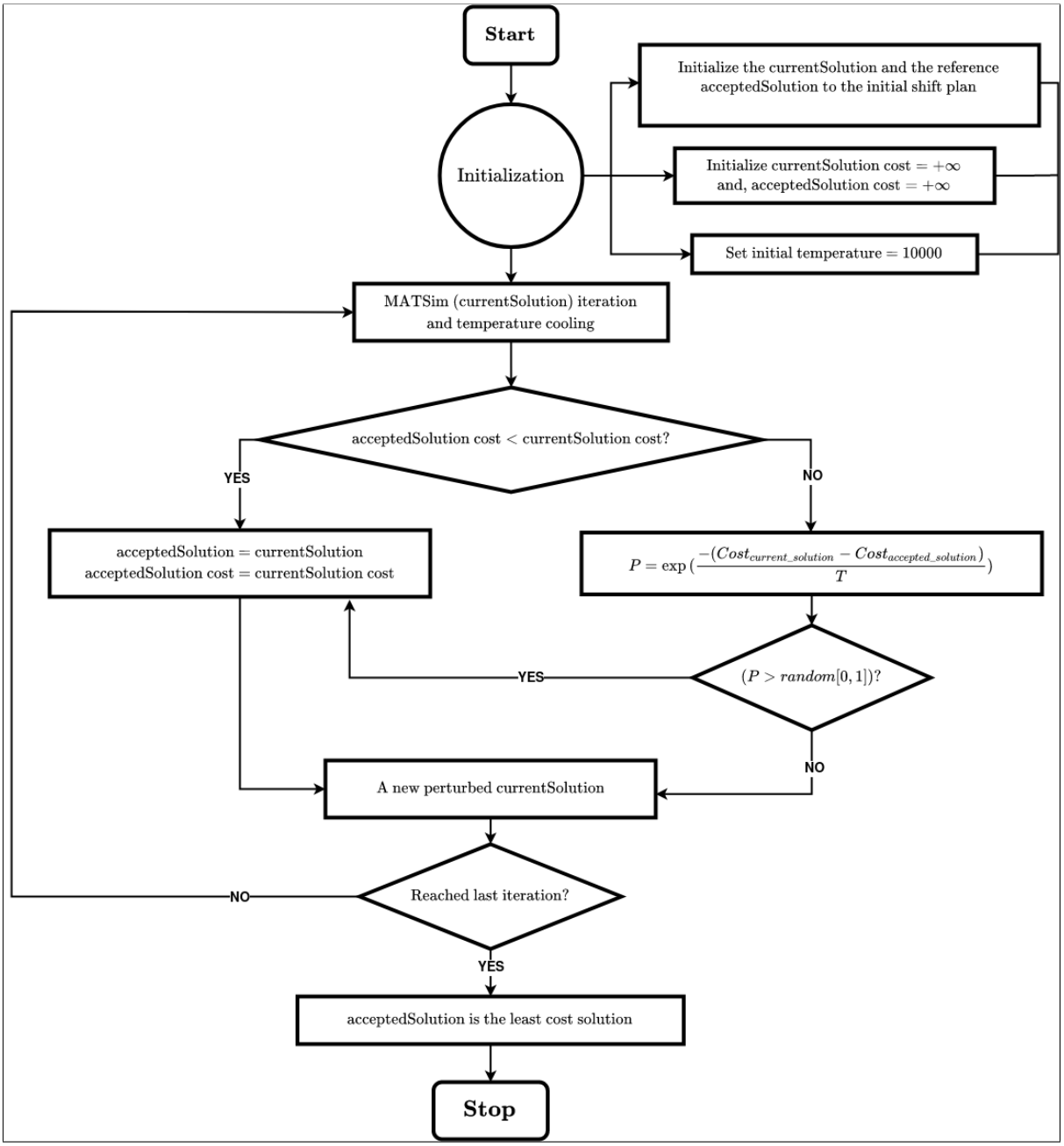


Figure 3.5: Flowchart of shift schedule optimization using Simulated Annealing

3.7 Encoding

Traditionally, encoding is applied to solutions in Genetic Algorithms (GA), where they are transformed to represent the solutions into a computer-compatible format. The mapping of these transformations can either be done numerically or alphabetically to store the exact information in the encoding. The GA borrows a lot of its terminology from genetics and is inspired by the biological genetics evolutionary model. Hence, Genotype represents the encoded solution in a machine-readable and code-formattable format. In comparison, the Phenotype term describes the actual human-readable format. A chromosome is a code-formattable data structure of a specific length containing all the necessary information of the actual solution's data structure. Here, the actual solution's data structure is the XML format, which the figure 3.3 shows. Even though XML is machine-readable, it is not code-formattable. A LinkedHashMap was chosen as the code-formattable data structure efficiently representing the driver shift (and break) schedules.

A **LinkedHashMap** is an alternative implementation of the **Map** structure with keys as an index to an associated data value. Hence, a **Map** contains several unique unordered keys and their corresponding values. With **LinkedHashMap**, the indexed keys are in an ordered format. The order was necessary for the encoding as the keys were meant to store the temporal data.

The following properties are taken into account when choosing encodings:

- A chromosome's structure should be well represented since only a good representation will narrow the search space, whereas a poor representation widens it. Moreover, if coding schemes are inappropriate, the convergence rate will be slower, and the computational burden will be higher. (Banchs, 1997a).
- The genotype should provide a tractable way to map Genotype data to Phenotype data and vice versa; that is, it should be easy to decode the encoded Genotype data. (Fox and McMahon, 1991).
- The genotype should contain all the vital information from the actual data, as missing information would lead to errors and widen the search space.

Encoding schemes can be classified into binary, value, octal, hexadecimal, 1D, or 2D based on the structure (Kumar, 2013). For the shift optimizer model, a single shift schedule has been encoded using a mixture of a 2D and value encoding scheme to a LinkedHashmap structure, where the keys represent the time of day in seconds. The values represent the integer numbers ***0***, *1* and ***2***. Values are described as follows:

Table 3.2: Value encoding description

Encoded Value	Description
0	Non-active shift schedule or no shift
1	Active shift
2	Break corridor

The indexed keys are in the increasing order of time, starting from the START_SERVICE_-TIME and ending at END_SERVICE_TIME, with every addition of TIME_INTERVAL. TIME_-INTERVAL is the time bin size set for the model as it helps in representing the time for the indexed keys appropriately. It was assigned to $^1/_2$ hr or 1800 seconds. The chosen value accommodates a break of 1800 seconds or 30 minutes set by German Labour Law. Therefore, the number of unique keys depends on the START_SERVICE_TIME and END_SERVICE_-TIME parameters set to 0 seconds (0 hr) and 108000 seconds (30 hr), respectively. Thirty hours are chosen as it includes the ride-pooling trips and shift (and break) schedules after midnight so that MATSim can simulate for trips starting very late in the day.

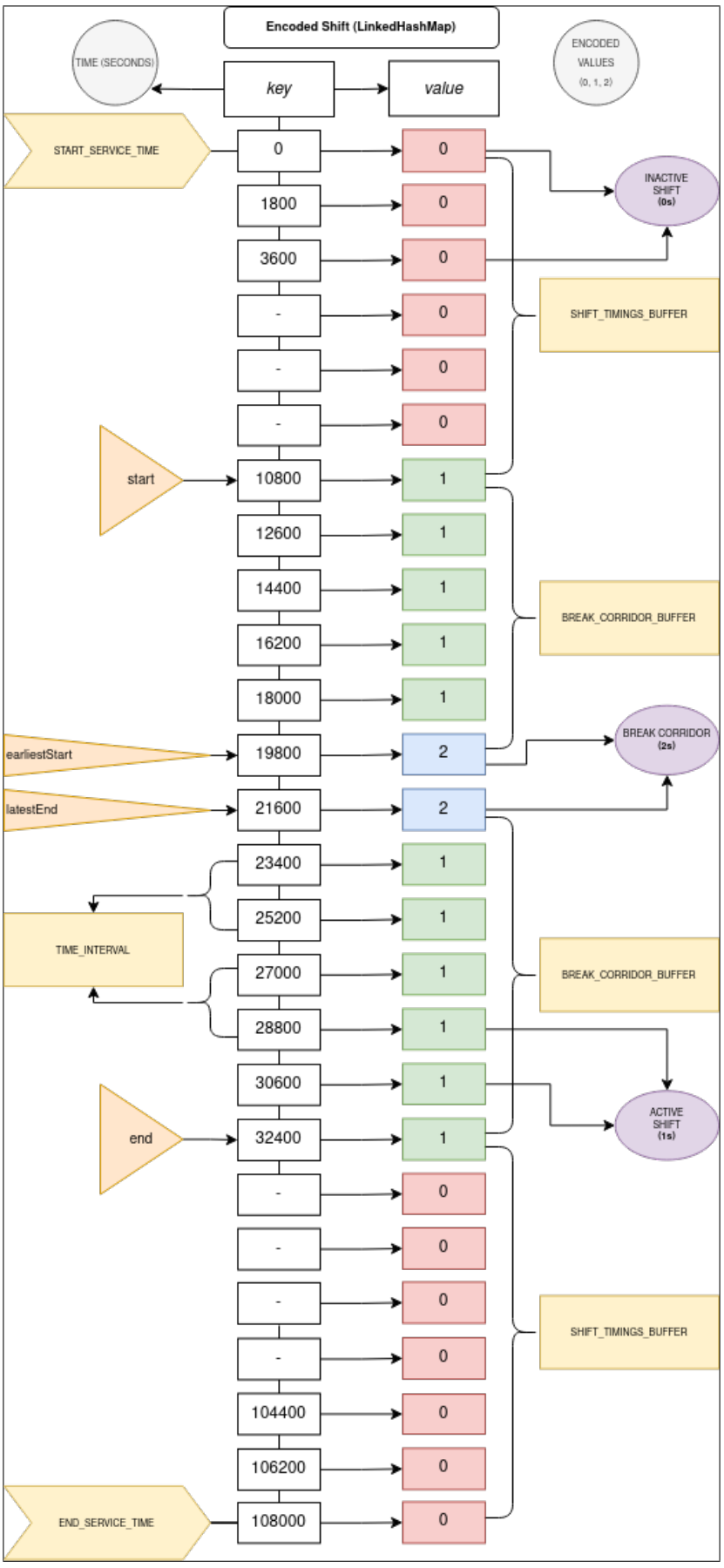


Figure 3.6: Encoded Shift - Shift with id: 5 in figure 3.3

3.8 Initial Solution

As SA is a trajectory-based algorithm applied to a single state optimization technique, as it aims to achieve single solution modification to reach global minima. In contrast to other trajectory-based algorithms, such as hill-climbing, neural network, or almost all greedy algorithms have high chances of getting stuck in the local optimum (Tovey, 1985; Schmitt and Wanka, 2015). An advantage of SA is that the fitness of an accepted neighbour solution can have both high and low costs, preventing the confinement in the local minima (Abramson, 1991). An initial solution is intended to provide a good starting point for future processing, influencing the potential processing time. When initializing shifts, it is imperative to ensure that all the unfeasible shift schedules, having no constraint violations, are omitted from the initial solution. It may cause the algorithm to fail in the future, as unexpected errors would occur due to this flawed initialization and confuse the algorithm on understanding the solution's quality.

As mentioned before, the model refers to the initial **shift plan** or **solution** as the ***DrtSolution*** object or the ***Individual*** object, a representative of java codes in the model. A ***DrtSolution*** is a collection of ***DrtShift*** objects. ***DrtShift***, representing a shift (and break) schedule, consists of their ***Id***, **start**, ***end***, and ***ShiftBreak***, coded by the developers (Kuehnel et al., 2021-05). ***ShiftBreak*** is also a java object used to represent the break corridor in the shift, containing ***earliestStartBreakTime***, ***latestEndBreakTime***, and **duration**. Similarly, the ***Individual*** object is a collection of ***SAShift*** objects which also contain the same variables defined in the ***DrtShift*** object and the ***SABreak*** object like the ***ShiftBreak*** object that the author of this thesis coded.

For the Holzkirchen scenario that is considered in the thesis, the authors manually and arbitrarily prepared an initial shift plan in (Zwick et al., 2021). These researchers wrote a shift plan with 30 number of driver (shift and break) schedule having various shift timings and break corridors. To test the shift optimizer model's robustness and efficacy, the model makes use of manually created initial solutions with 5 and 60 numbers of driver (shift and break) schedule. The results of these three scenarios have been classified into three categories, namely **5_shift_plan scenario**, **30_shift_plan scenario**, and **60_shift_plan scenario**, which are configured with different parameters settings internally.

3.9 Temperate and Cooling Schedules

3.9.1 Temperature

Temperature is used as a control parameter in the SA algorithm that decreases to simulate cooling. The cooling is usually done iteratively and at slow speeds. A slow-speed cooling ensures an extensive solution exploration in neighbourhood space. It is also possible to simulate heating in the algorithm, which rapidly increases the temperature at specific iterations to further the exploration stage. Nevertheless, the model does not need to implement this sudden heating behaviour as the search space is discrete and not too vast. However, it is imperative to set the initial temperature to a high value in the beginning and decide a gradual rate at which it should cool (Norgren and Jonasson, 2016).

3.9.2 Cooling Schedules

The specific temperature reduction over time defines a cooling schedule. It is a crucial process in determining the algorithm's speed (Bogdanov, 2015). Therefore, selecting an appropriate annealing schedule is one of the most critical aspects of the simulated annealing algorithm, and numerous efforts have been made in this direction.

A cooling schedule is needs the following parameters:

3.9.2.1 Initial Temperature

Because of Metropolis's acceptance probability equations' dependence on temperature, the solution space is more likely to be explored at high temperatures. The neighbour solution is expected to be exploited at lower temperatures. A high value should be set to the initial temperature to increase the likelihood of accepting a worse solution in the early stages. Too low of an initial temperature will lose the global minima of the neighbour space. However, a very high starting temperature will cause the algorithm to go back and forth within the investigation space repeatedly. The result of this high starting point is a waste of valuable computing time, and also, if there are no sufficient iterations, it may lead to the search failing (Banchs, 1997c). Thus, defining the initial temperature value is crucial so that any new neighbour solution is accepted only during a few early iterations of the cooling scheme. In practice, determining the initial temperature can be difficult, so an a priori understanding of the problem can help resolve it. The initial temperature can also use the procedure of (Nourani and Andresen, 1998), who suggested that the initial temperature should be such that at the temperature, the expected cost is within one standard deviation of the average cost. Keeping the procedure in mind and several preliminary test runs, 10000 was fixed in all configuration sub-models.

3.9.2.2 Decrementing temperature function or a cooling pattern

Concerning the shift optimizer model, a cooling pattern is outlined by how initial temperature reduction should take place with increasing MATSim iterations. The temperature reduction or cooling scheme is crucial for a successful search, as it influences the time the SA algorithm spends in the search space. A cooling schedule can be static or dynamic and fast or slow. An algorithm with a dynamic cooling schedule can adapt its temperature parameters. At the same time, it is being lowered, as opposed to a static cooling schedule which has a fixed reduction process (Aarts et al., 2005). The thesis only implements a fixed cooling schedule, which may make the algorithm too complex. Authors (Romeo and Sangiovanni-Vincentelli, 1991a) argued in the paper that implementing a cooling schedule effectively is imperative to speeding up the process to achieve an optimal solution. Hence, let's observe the various static cooling processes that are commonly used in research:

- **Linear**: Simulation annealing is commonly implemented using the linear approach to apply the temperature reduction cycle. With a linear cooling schedule, the temperature is reduced by the same amount during the annealing process instead of sudden exponential or logarithmic cooling reductions. Each time the annealing process is iterated, it is lowered by the linear_factor or α amount. It is generally considered to be a slow cooling process. Whose equation is given by:

$$T_i = T_0 + \frac{i}{\alpha} \times T_0 \tag{3.2}$$

  Where, $\mathbf{T_i}$ is the instantaneous temperature at $\mathbf{i}^{\text{th}}$ iteration, $\boldsymbol{\alpha}$ is the constant value that determines the cooling speed and $\mathbf{T_0}$ is the initial temperature

- **Exponential**: Pioneer researchers of SA in (Kirkpatrick et al., 1983) have created this cooling criterion, and it is used as an example for comparing the various cooling criteria. As the temperature gets lowered in each iteration, the factor reducing the temperature exponentially ($\boldsymbol{\alpha}$) is multiplied by the initial temperature $\mathbf{T_0}$. Cooling temperature using this method is undoubtedly the most common as it can be fast and slow depending on the $\boldsymbol{\alpha}$ value. Higher values of $\boldsymbol{\alpha}$ result in slowest cooling. Smaller values cause excessively rapid cooling, which may get stuck in the local optima because of no exploration time. Good $\boldsymbol{\alpha}$ values usually lie between 0.8 and 0.99 (Bohachevsky et al., 1986; Romeo and Sangiovanni-Vincentelli, 1991b). Which is given by:

$$T_i = \alpha^i \times T_0 \tag{3.3}$$

- **Logarithmic**: Given by:

$$T_i = \frac{c}{\ln(1+i)} = \frac{T_0}{1 + \alpha \times \ln(1+i)} \tag{3.4}$$

Introduced by (Geman and Geman, 1984), where **c** is an iteration-independent constant of the $\mathbf{i}^{th}$ iteration. Authors have claimed (Hajek, 1988) that this schedule guarantees convergence to the global optima in an infinite amount of time when **c** is greater than or equal to highest energy state or when $\boldsymbol{\alpha} = 1$. Thus, the schedule is entirely impractical due to the prolonged and asymptotically temperature reduction. Moreover, it does not fit the Metropolis acceptance function due to its prolonged speed. As a result, it is rarely used in practice. Nevertheless, a trade-off generally is seen between the cooling schedule rate and the quality of solutions explored. A slow reduction in temperature produces superior results but at the cost of longer computation time.

The linear approach equally favours exploration and exploitation time. Due to slow temperature reduction, exploration time is preferred to exploitation time in a logarithmic cooling scheme. In comparison, the exponential cooling schedule may change its exploration or exploitation time according to its speed. It forces a transition from exploration to exploitation much faster, which is mainly preferred by researchers (Nourani and Andresen, 1998; Peprah et al., 2017). Additionally, combinatorial optimization problems like the shift optimizer perform better with exponential cooling as they discrete neighbour solution space (Wegener, 2005). Figure 3.7 shows the graphical representation of the cooling schedules.

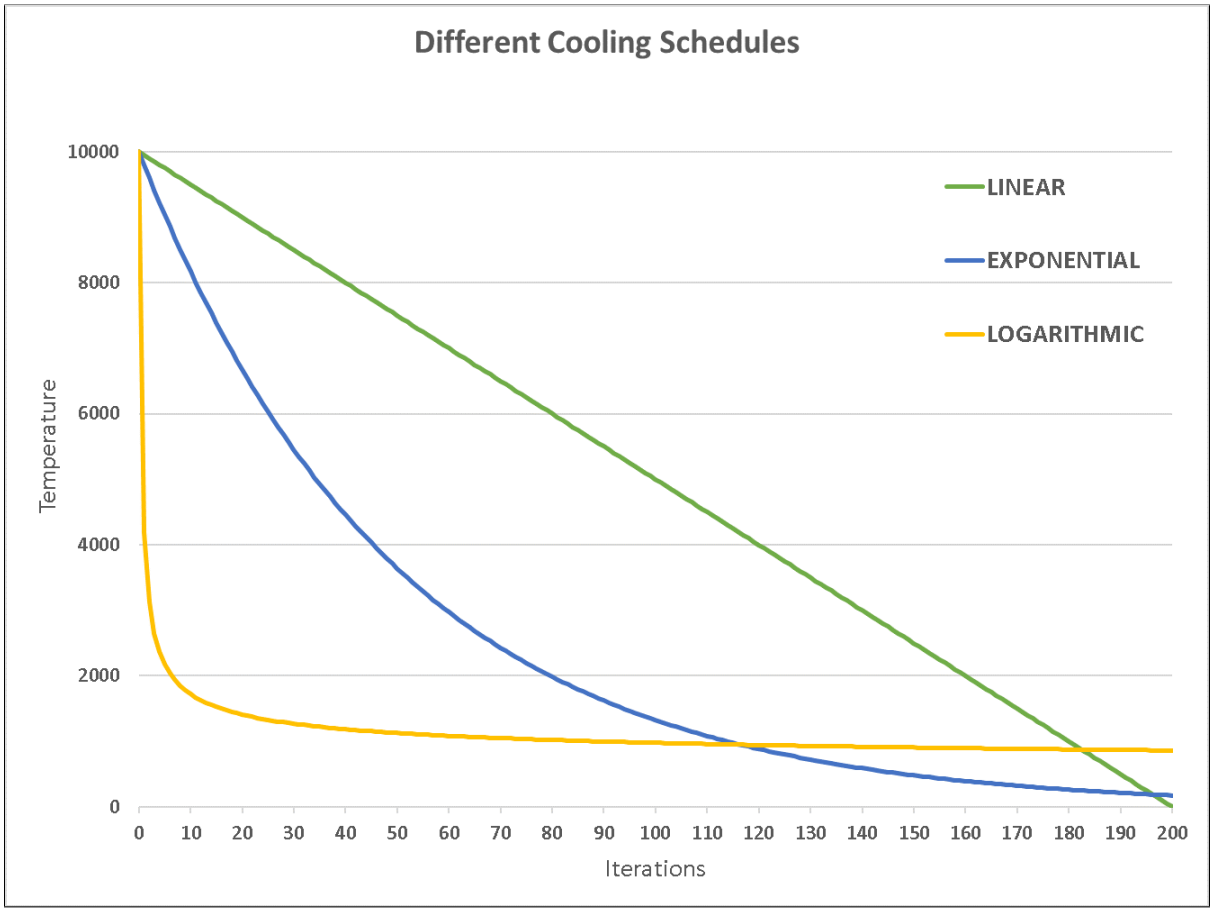


Figure 3.7: Graphical representation of different Cooling Schedules

3.9.2.3 Final temperature or Stopping condition

A final temperature is the last temperature that the cooling schedule needs to reach before the algorithm stops. A simulated annealing algorithm uses different stopping criteria based it's the neighbourhood space. A more expansive neighbourhood solution space lets the stopping criteria be more flexible. Flexibility means the stopping condition may become hybrid, changing over time. Whereas narrow and discrete spaces have a fixed termination criterion (Banchs, 1997c).

Since it has been established that the search space for a new driver shift (and break) schedules solution is discrete and less expansive, a fixed termination condition is suitable. The criteria for obtaining the desired accuracy must, however, be considered more carefully when doing a stand-alone global search (Banchs, 1997c). As a result, the number of iterations criterion was chosen as running simulations in MATSim iteratively was an essential process in the model. Therefore, setting the final temperature criterion was not done explicitly. The **ITERATIONS** parameter, which is a temperature-independent value, should be large enough not to stop the search (Banchs, 1997b) prematurely. Hence, many iteration values have been tested for this thesis to know which value yields complete results. As the run time of the Simulated Annealing algorithm is highly influenced by the number of simulation iterations, the small study service areas resulted in a run time of $^1/_2$ hour for 200 iterations. It is not high in itself, but because the model is not calibrated with the suitable parameters, it needs several independent runs, making the total run quite huge. Also, testing the model's performance on a more significant area will increase the run time.

3.10 Objective Function - cost of solution

The objective (cost) function is a crucial component in determining the size of the neighbourhood space of the solution and influencing the speed for a successful search (Moscato, 1993). It is defined as the fitness or quality of a solution. A neighbour solution space should resemble a stable topological structure with several small local optima as irregular search space with more minor deep local minima is not favourable to the SA algorithm according to (Eglese, 1990). In the context of the shift optimizer model, the neighbour solutions of the driver (shift and break) schedule should be close to each other in terms of their cost (defined later). New solutions with several high and low costs aren't beneficial for the algorithm efficiency. It requires that the new neighbour solutions either be similar in nature, i. e., having minor adjustments, or have an appropriate objective function definition to make them as such. Therefore, the model kept the goal of the optimization algorithm in mind to create the objective function definition and included the costs of:

1. active shifts per hour: It governs the travel supply of the system and is based on the number of hours drivers work. Since ride companies want a sufficient and efficient supply of drivers working for them, it was essential to include this term.

2. rejections per hour: It occurs when the values in submitted requests per hour are higher than the values active shifts per hour. Higher rejections would lead to higher costs. Therefore, it was included to bring it down ultimately.

3. rejection rate per hour or the maximum rejection rate: It is an indispensable measurement factor that needs to be brought down to improve the efficiency of the ride-pooling system. Therefore, it is included as a hard constraint.

Cost of solution is calculated using:

$$\begin{aligned} \min: \ \text{Cost} = & \sum Solution_{driver\_hours} \times \beta_{hourly\_rate} \\ & + \\ & \sum Rejections_{per\_hour} \times \beta_{hourly\_rejection\_cost} \\ & + \\ & \sum \rho_{per\_violation} \end{aligned} \tag{3.5}$$

Where, all these parameters are in a LinkedHashMap data structure for easy computation in the code:

- **Solution$_{\text{driver_hours}}$** is the number of hours all the drivers work on a given service day. The value is similar to the **active shifts per hour** parameter but not the same. The

reason they are different is that, break corridor is not considered in active shift per hour variable. Since it is an inactive shift (**0** in encoding), **active shifts per hour** only takes active shifts (**1** in encoding) into consideration. Whereas, $\textbf{Solution}_{\textbf{driver\_hours}}$ variable the whole duration of driver shift (and break) schedule. The summation of $\textbf{Solution}_{\textbf{driver\_hours}}$ gives the total hours all the drivers have worked on that particular day.

- $\textbf{Rejections}_{\textbf{per\_hour}}$ are the non-served rides summed up for each hour based on the ride-pooling simulation in MATSim. It is exactly the same as the **rejections per hour** value. The summation of this value gives the total number of rejections during the services day.

- rejection rate per hour value is not explicitly used in the objective function; it is instead added a penalty: $\boldsymbol{\rho}$ **which** $_{\textbf{per\_violation}}$ is a hard constraint added to increase the cost of solution drastically whenever the rejection rate exceeds the **DESIRED_REJECTION_RATE** set at 0.2 (or 80% acceptance of requests). The **PENALTY** value for such violations should be kept very large to increase the cost of the solution. Discussed later in the thesis is why 9999 was set. And, summation of such penalties gives the total number of violations exceeding the desired rejection rate value.

- $\beta_{\textbf{hourly\_rate}}$ or **DRIVER_COST_PER_HOUR**: It is a configurable constant set to 30 €/hr. Due to privacy and legal reasons, obtaining actual driver salaries was not possible. However, it was only possible to read from the advertisement (MOIA's Driver advertisement) that tells that driver's are paid 12 €/hr as their base salary. But then, considering the company's social security payments and other overhead costs, it was roughly estimated to be around 30 €/hr. It has been kept constant through the configuration settings.

- $\beta_{\textbf{hourly\_rejection\_cost}}$ or **COST_PER_REJECTION_PER_HOUR**: It is also a configurable constant value set to 10 €/hr. The value accounts for the missed rides revenue and some penalties due to reduced passenger satisfaction. It has also been kept constant through the configuration settings.

Note that the set parameter (driver cost, rejection cost or penalty) values are changeable and can be set to a different number. However, in the scope of this thesis, these numbers were selected and fixed for sub-models .

3.11 Constraints

It is impossible to assign employees to specific shift (and break) schedules across many occupations. It is not easy to satisfy all employees as different employees have different priorities and preferences. There are also legal, company-specific, and demand-specific requirements to consider in creating the schedules. Hence, it is a nightmare for the people who manually make employees' schedules. However, researchers overcame this problem mathematically; they classified such requirements as constraints in an equation and tried to solve them without violating these constraints (Wong et al., 2014). Not only do they constrain the perturbation strategy by narrowing the search space of neighbour solutions, but they also help in giving weights to independent variables in the objective function.

Similarly, these hard and soft constraints are adopted in shift and break scheduling (Kletzander and Musliu, 2019). The objective function must not violate hard constraints, but it is permissible to violate soft constraints at a cost (Chen et al., 2020). Since values of these constraints were not generalizable due to a lack of literature, the model had to be tested with many arbitrary constraint values. Later in the thesis, it has been discussed what these values were and how well they yielded results. The shift optimizer model took soft constraints to reduce rejections per hour and remove bad quality driver (shift and break) schedule. In comparison to hard constraints, that narrows the neighbourhood search space due to the perturbation parameters and improves the efficiency of the ride-pooling system.

3.11.1 Hard Constraints

3.11.1.1 Hard Constraints implicitly included in objective function but explicitly used in perturbations

1. **TIME_INTERVAL**: It is the time bin size that for important for encoding the shift.
2. **START_SERVICE_TIME**: It is the time when all drivers start working, or the ride-pooling company starts its operations. Used as the starting point for key of an encoded shift. The time helps in acting as a limiting (lower bound) the textbfmoveSAShiftTimings perturbation. Set at 0:00 or 0 seconds or midnight.
3. **END_SERVICE_TIME**: It is the time at which all drivers stop working or the time at which ride-pooling companies stop their service operation, such that no shift (and break) schedule can be assigned to a driver beyond this time. It is also used in determining the length of an encoded shift, and acts upper limiting bound for *moveSAShiftTimings* perturbation. Set at 30:00 or 108000 seconds or 6 AM the next day. This constraint has not been limited to 24:00 or midnight so that night shifts or trips during night times can be included.

4. **SHIFT_TIMINGS_MAXIMUM_LENGTH**: As the name suggests, the maximum length of shift timings (difference between end - start) of a legal shift. The parameter is set to keep all shift schedules for a length of Eight and a half hours (8.5 hr) as a legal requirement from the German Law. The extra half an hour includes the half-hour breaks in these shifts. The parameter is used in the *increaseSAShiftTimings* perturbation function as the upper bound for increasing the shift (and break) schedule's length if it is feasible. In reality, shifts may sometimes be bigger than 8.5 hours due to excessive demand, but they have not been considered in the shift optimizer model. It has been set to 30600 seconds (8.5 hr) in all configuration sub-models as a legal requirement by ride-pooling companies in Germany

5. **SHIFT_TIMINGS_MINIMUM_LENGTH**: The minimum length of a legal shift (and break) schedule is required by German Law. Drivers must work a minimum of five and half hours (5.5 hr), which is what was legally set by the German government (für justiz und verbraucherschutz, 2021). The Perturbation strategy *decreaseSAShiftTimings* uses the parameter as a lower cap for not decreasing the length of a shift (and break) schedule if feasible. It has been set to 19800 seconds (5.5 hr) in all configuration sub-models for the same reason.

6. **BREAK_CORRIDOR_LENGTH**: It is the relief-time length that is assigned within the break corridor. It is set to a fixed value of $\frac{1}{2}$ hr or 1800 seconds.

7. **BREAK_CORRIDOR_BUFFER**: It is the time from break corridor's *earliestStartBreakTime* and *lastestEndBreakTime* beyond which a perturbation of moving break corridors can not take place. The buffer parameter is set to 2 hr or 7200 seconds, as shifts cannot have breaks starting before 2 hours after start time or after 2 hrs before end time. This serves as a limiting bound for the *moveSABreakCorridor* perturbation as a break corridor's earliestStartBreakTime and lastestEndBreakTime values can not fall within these 2 hrs.

8. **SHIFT_TIMINGS_BUFFER**: It is the time from shift's start or end beyond which moving shift time perturbation cannot take place. It is set at 0. The parameter serves a limiting bound for *moveSAShiftTimings* perturbation as a shift's start and end times can not fall within this number. It has not been set to different number other than 0 because drivers at ride-pooling companies can start working exactly from the START_SERVICE_TIME and stop working exactly at the END_SERVICE_TIME.

3.11.1.2 Hard constraints explicitly utilized in the objective function

The **PENALTY** parameter is added to the cost function 3.5 whenever a violation exceeds the **DESIRED_REJECTION_RATE**. The model wants to search for an optimal solution where the rejection rate per hour, specifically maximum rejection rate, is below 0.2 (20%).

The value of the penalty is kept high enough to increase the cost of the solution whenever it occurs. High costs make the solution worse for the algorithm and reject it in the exploitative stage. If the penalty value is set too low, the algorithm will not distinguish between a good quality solution from a bad quality once, thereby not improving the system's efficiency. The objective function must realize the hard perturbation constraints to meet legal and workplace regulations.

3.11.2 Soft Constraints

3.11.2.1 Soft constraints implicitly kept in the objective function and explicitly used in perturbations

1. **SHIFTS_REMOVAL**: The maximum number of shift (and break) schedules a solution can lose to get a new perturbed solution. Removals are based on the random number between 0 and the set value.

2. **SHIFTS_INSERTION**: A parameter opposite to the removals parameter adds or inserts new driver shift (and break(schedules to the solution to produce a new neighbor solution. The number of insertions is based on the random number between 0 and the set value.

3. **SHIFTS_MAXIMUM**: The upper limit of the number of shift (and break) schedules that a solution (shift plan) can have. Used as an upper cap for *insertSAShifts* perturbation strategy. It should be based on the fleet size or the number of available drivers. Useful parameters to keep the neighbor solutions moving in a specific region may lead to faster convergence.

4. **SHIFTS_MINIMUM**: The number of shift (and break) schedules lower than this value cannot be permitted. The parameter is only useful when the optimal solution's number of shift (and break) schedules is known. Used an lower cap for the *removeSAShifts* perturbation function. The parameter should be set to a number that reduces the search space for optimal solutions, as very low values may lead to unnecessary processing time. A wide difference between SHIFTS_MINIMUM and SHIFTS_MAXIMUM may influence the exploration time.

3.11.2.2 Soft constraints explicitly utilized in the objective function

1. **DRIVER_COST_PER_HOUR**: The parameter denotes the hourly salary paid to the driver for his work defines this value. It is used in the objective function in 3.5 to reach a solution with fewer driver shift (and break) schedules by minimizing the operational costs of a ride polling company. It is multiplied by the sum of total driver hours on a working day which is, in fact, based on the **active shifts per hour**.

2. **COST_PER_REJECTION_PER_HOUR**: The parameter is defined as the cost of rejecting a request in a ride pooling simulation in the cost function in 3.5. It was introduced because a ride-pooling company incurred an indirect cost when the system rejected the passenger's request. After all, completing requests is what majorly generates revenue for the company. Since it is considered as a soft constraint, some rejections are allowed to occur, keeping in mind that **rejections per hour** doesn't exceed **DESIRED_REJECTION_RATE**, which is chosen to be 20%.

3.12 Perturbation Strategies

The term perturbation strategy or function is used to modify a solution containing all encoded driver shift (and break) schedules, with a slight manipulation in their encoded values leading to new encoded driver shift (and break) schedules. New neighbour solutions are generated due to the minor change. These manipulations are random but within the predefined constraints. The stochasticity occurs at two different levels during the perturbations. In the first level, a first random number is chosen between 0 and the number of shift (and break) schedules that determine the number and strategies of perturbations that will take place on the solution. Then, in the next level, a second random number decides the number of manipulations per strategy for each perturbation strategy. Please note that the perturbations don't produce many solutions simultaneously but instead create a single perturbed neighbour solution in every iteration.

To explain the levels of randomization, let take an example, a 30 shifts solution is to be perturbed. A first random number, let say 4, is taken. Then, 4 random perturbation functions are applied to the solution, let say, 1) removeShifts 2) moveShift 3) moveShift 4) decreaseShiftTimings. After that, for each function type, a second random number is selected, let us say 20, 6, 3, 15. Then 20 random shift (and break) schedules in the solution are manipulated with removeShifts, 6 random shift (and break) schedules are manipulated with moveShift perturbation, 3 random shift (and break) schedules are again manipulated with moveShift perturbation, and lastly, 15 random shift (and break) schedules are manipulated with decreaseShiftTimings. Note: The number of shift (and break) schedules is randomly selected in any order, and the perturbation strategies are replaceable; they can be selected more than once.

Perturbations occur at the end of every iteration so that the new solution perturbed on the current solution can be taken as the following input for the MATSim iteration. Only the current solution is perturbed and simulated. The accepted solution acts as a tracker to store only those solutions with low cost compared to the previously accepted solution.

Perturbations affect the performance of the algorithm because it searches for neighbour solutions. If perturbation strategies are not minute in nature, it may lead to a failed search. However, no changes or minor changes may only increase the computational time and effort. Therefore, these changes are problem-specific. Perturbation type is defined later in Section 4.1.1.

The following functions are the various strategies in pertubating a solution:

1. ***insertSAShifts***: It creates a new neighbour solution with an increased number of shifts. The new shift (and break) schedules inserted into the solution are the manipulated (with

other perturbations) shifts schedules already present in the solution. With an increase in shifts, the number of active shifts per hour will have a value increase, thereby increasing the cost of the solution; however, if a certain number of good shift (and break) schedules are added, keeping the demand static, the cost may also decrease. Since the algorithm is in its developing stage, shift (and break) schedule(s) inserted are flawed (as it is taken from the solution itself) shift (and break) schedules. One can later improve this technique. For example, in figure 3.8b the shift ***6*** and ***7*** (marked in red and randomly selected) are added to the Shift Plan in figure 3.8a.

2. ***removeSAShifts***: The function removes a random number of shift (and break) schedules from the solution by significantly affecting the **active shifts per hour**. As a result, the cost of a solution that depends on this value is susceptible to such manipulation. There are chances of removing good quality shift (and break) schedules from the solution, leading to increased cost. However, shift (and break) schedule removal generally leads to decreased cost because of its direct influence. For example, in figure 3.9b, shift ***3***, marked in green and randomly selected, is removed from the Shift Plan in figure 3.9a.

3. ***moveSABreakCorridor***: The function is intended to relocate the whole break corridor (from earliestStartBreakTime to lastestEndBreakTime) to shift to a new location and place it within the bounds of BREAK_CORRIDOR_BUFFER. With this perturbation, a minor change arises in the **active shifts per hour** thereby influencing the cost of the solution. This minority in change is because a break corridor (*2s* in the encoding shift) is smaller compared to the active shift. For example, in figure3.10b, the marked blue shifts ***2***, ***4***, ***5*** (randomly selected are perturbed by moving their break corridors to new timings (locations), concerning figure 3.10a, resulting in changes in the earliestStartBreakTime and lastestEndBreakTime values in them.

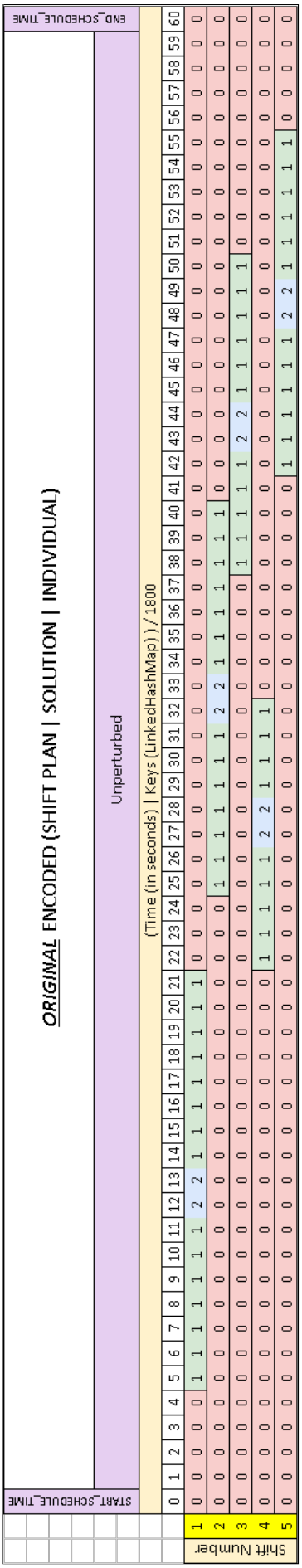
4. ***moveSAShiftTimings***: The function relocates the shift (and break) schedule (from start to end) to a new location, not violating SHIFT_TIMINGS_BUFFER or the service time (from START_SERVICE_TIME to END_SERVICE_TIME). Major changes occur in the **active shifts per hour** with this perturbation, directly influencing the number of active shifts (1s in the encodings). For example, in figure 3.11b, the marked blue shifts 4 are shifted to new timings (locations), concerning figure 3.11a, resulting in changes in the start and end times.

5. ***increaseSAShiftTimings***: This perturbation increases the shift (and break) schedule's length by elongating the duration of shift (and break) schedule times only if it is under the SHIFT_TIMINGS_MAXIMUM_LENGTH parameter. The elongation is done through bringing the shift's start time closer to START_SERVICE_TIME and shift's end time closer to END_SERVICE_TIME. Concerning the encoded shift, more number of *1s* are added to shift (and break) schedule SHIFT_TIMINGS_BUFFER. The elongation length

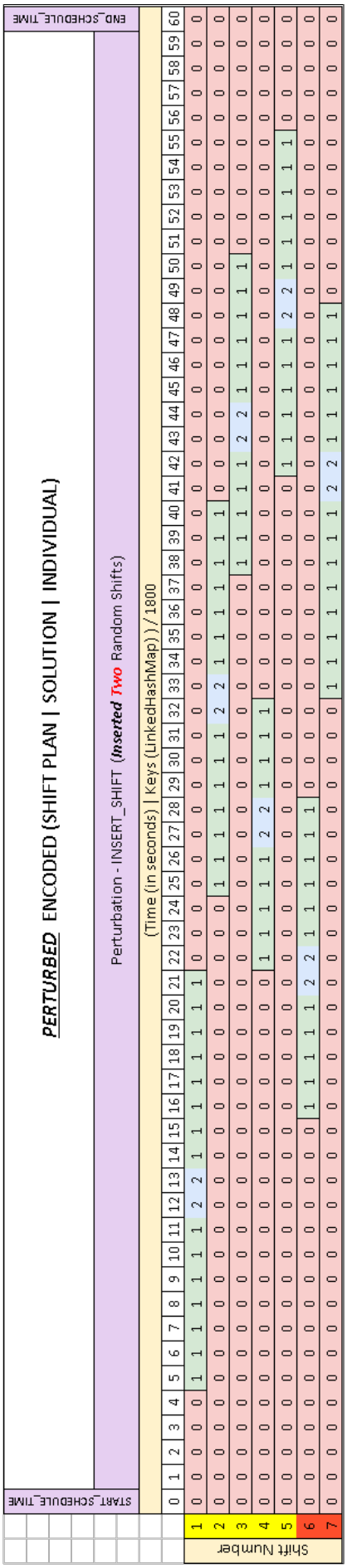
or the number of *1s* are also based on randomness. The influence on **active shifts per hour** is that it increases the number of active shifts on a few time bins indexes during the service time. For example, in figure 3.12b, all the blue-marked shifts from ***1*** to ***5*** are perturbed by elongating (if possible) their original timings (locations) in figure 3.12a,to new timings resulting in an increase in the duration of the shift (and break) schedules' timings

6. ***decreaseSAShiftTimings***: This perturbation is directly opposite to the perturbation mentioned above, where it randomly decreases the length of the shift (and break) schedule or removes number of random *1s* from the shift. The SHIFT_TIMINGS_MINIMUM_-LENGTH parameter is kept into consideration, as shift's duration cannot be lower than it. The effect on **active shifts per hour** is the opposite of what ***increaseSAShiftTimings*** perturbation has on it. For example, in figure 3.13b, the blue-marked shift (and break) schedules ***2*** and ***5*** are perturbed by shrinking (if possible) their original timings (locations) in figure 3.13a to new timings resulting in a decrease in the duration of the shift (and break) schedules.

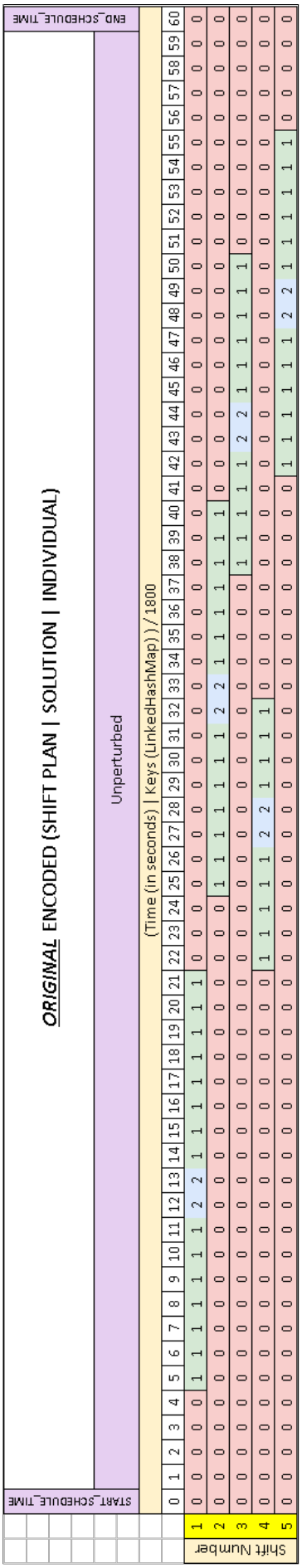
Note that colour schemes in the figures below are just for explanation and have no importance in the model.



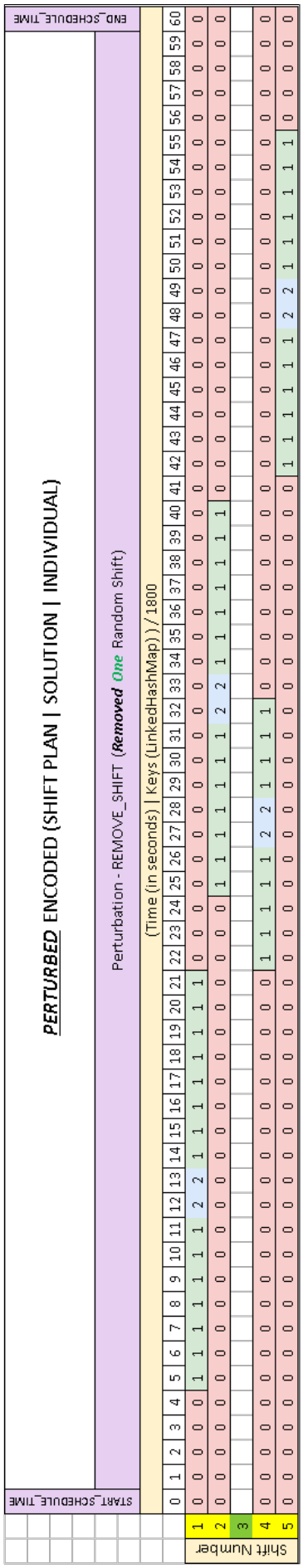
(a) Original Shift Plan - Unperturbed



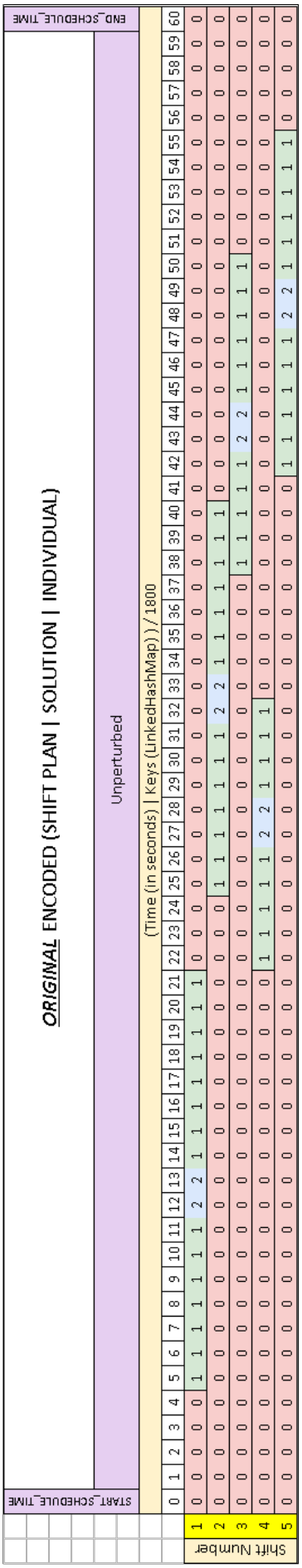
(b) Insert Shifts Perturbation on the above Original Shift Plan



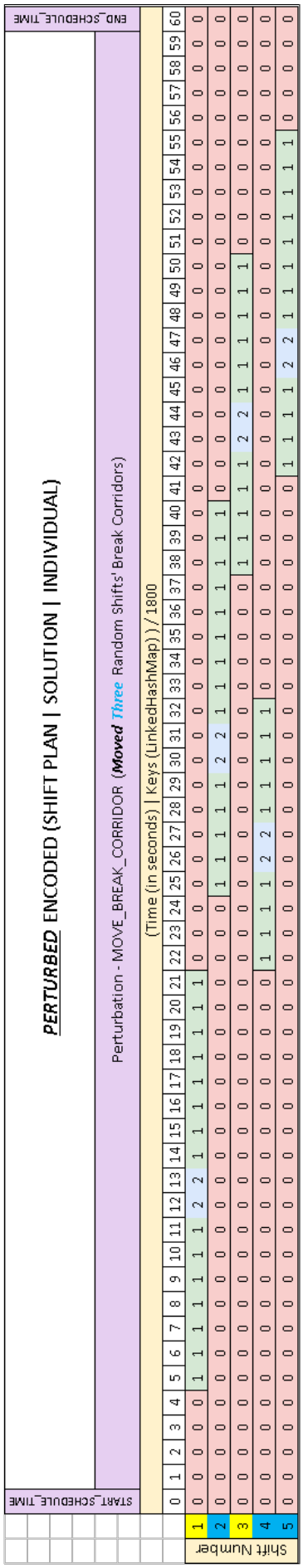
(a) Original Shift Plan - Unperturbed



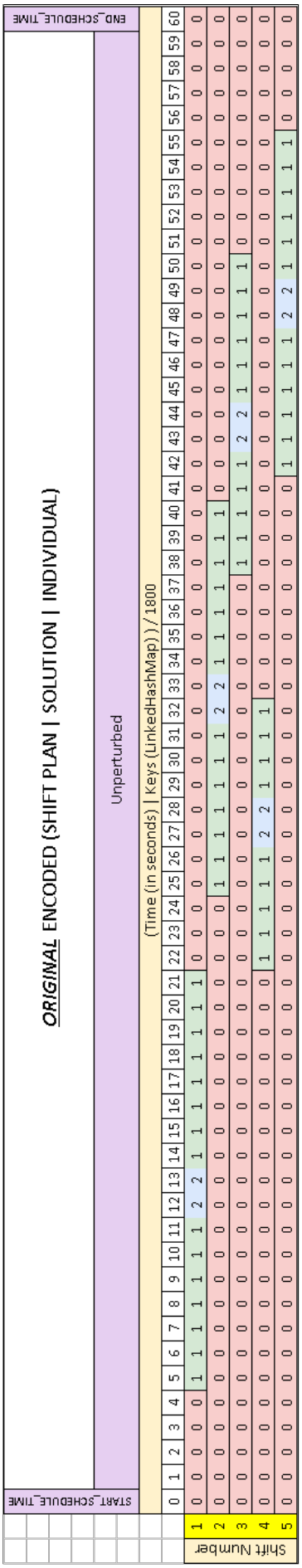
(b) Remove Shifts Perturbation on the above Original Shift Plan in



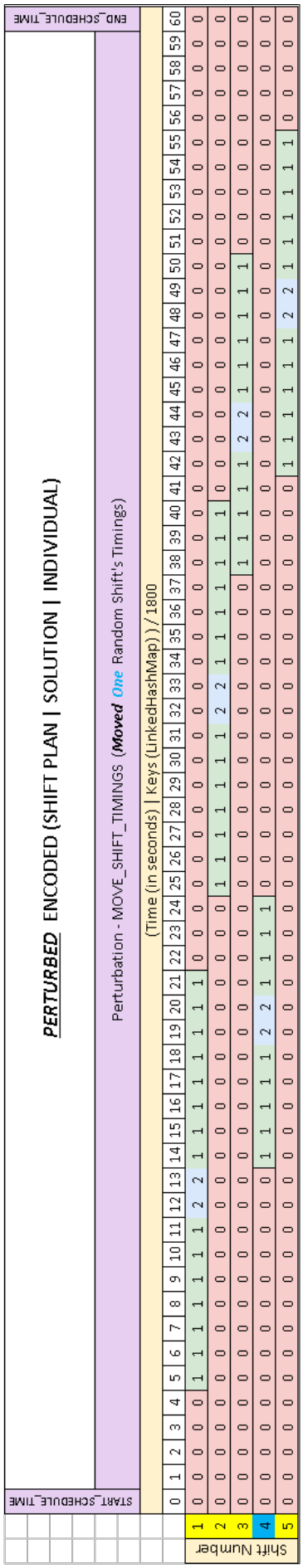
(a) Original Shift Plan - Unperturbed



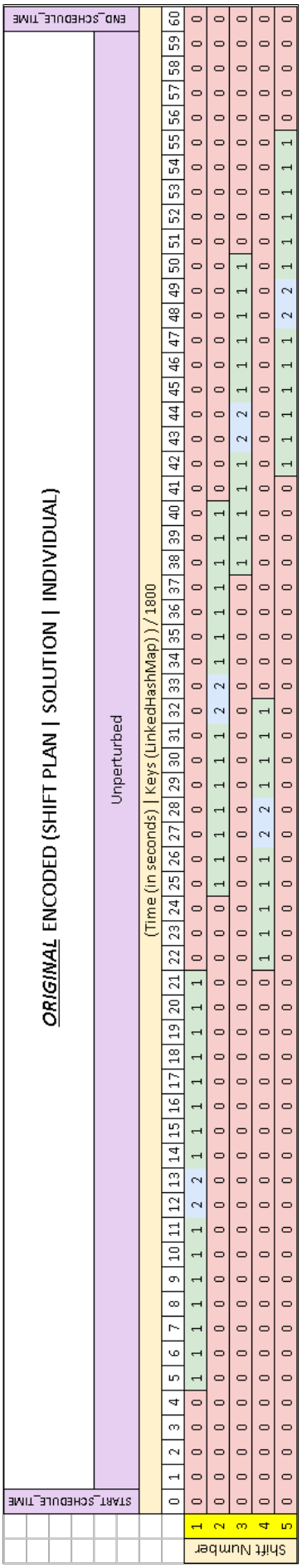
(b) Move Break Corridor Perturbation on the above Original Shift Plan



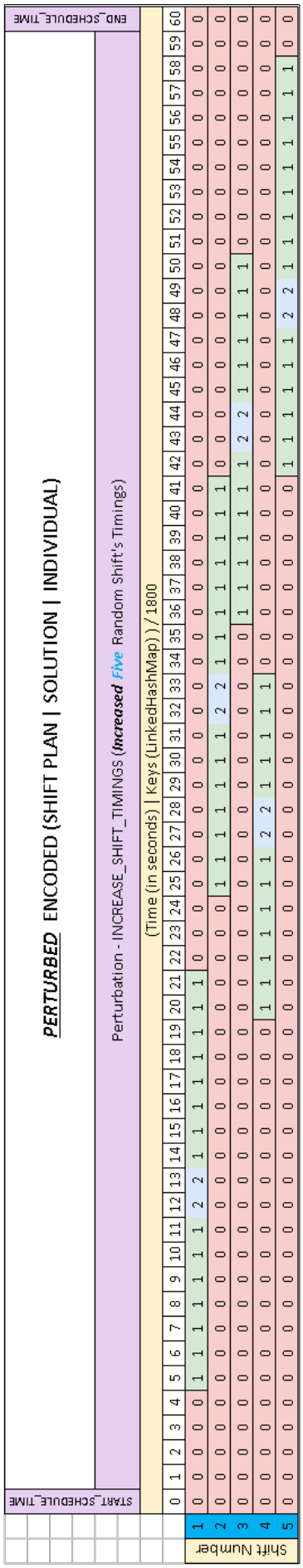
(a) Original Shift Plan - Unperturbed



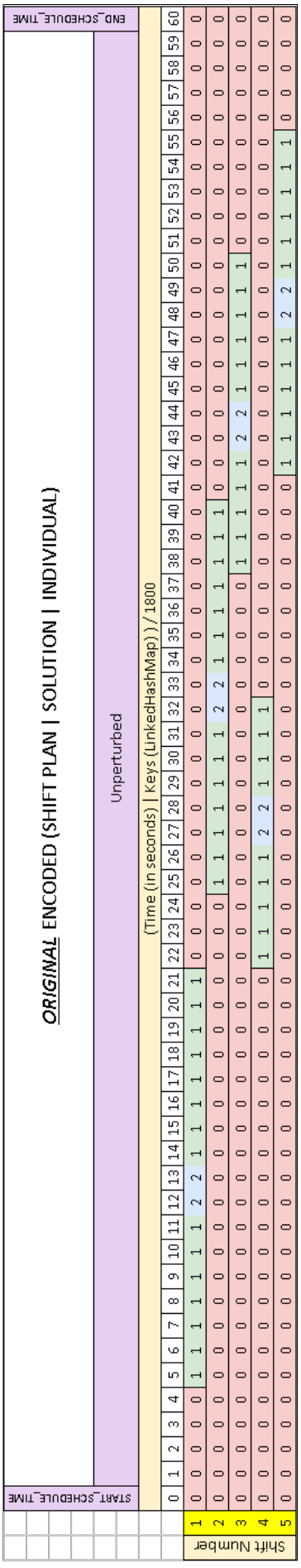
(b) Move Shift Timings Perturbation on the above Original Shift Plan



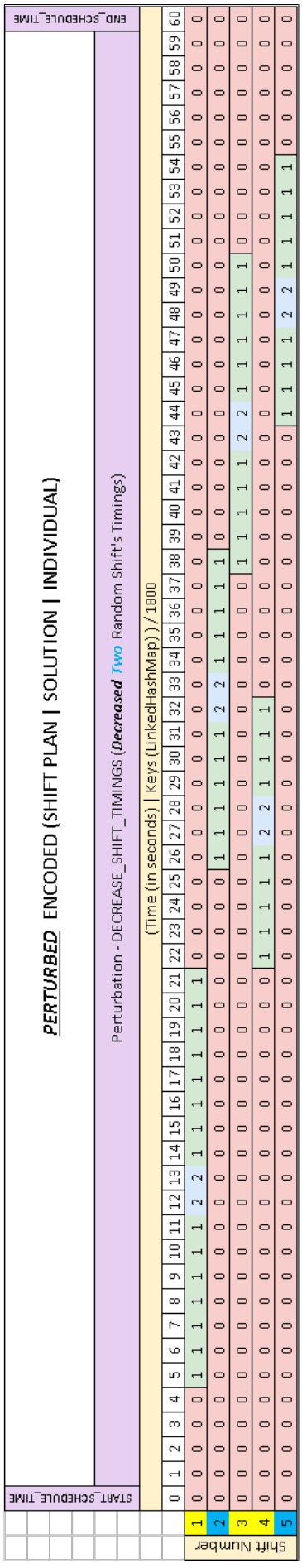
(a) Original Shift Plan - Unperturbed



(b) Insert Shift Timings Perturbation on the above Original Shift Plan



(a) Original Shift Plan - Unperturbed



(b) Decrease Shift Timings Perturbation on the above Original Shift Plan

Chapter 4

Results

This chapter presents the results of the **Shift Optimizer Model** that was specifically designed using a Simulated Annealing algorithm to optimize driver shift (and break) schedules in ride-pooling situations.

Since MATSim and the Shift and Break extension are written in Java, it was also used to generate output data. Meanwhile, (Python) was used to analyze the output results and plot them. Iterative simulations of MATSim and outcome analyses were executed on a 32 GB RAM, 3.00GHz Intel(R) Xeon(R) ES-1660v3 processor, where every iteration lasted for around 15 seconds. Thus, it meant that for 200 iterations, it took approximately $\frac{1}{2}$ hour, which was not too slow individually, but when combined with several other sub-model runs, it was.

The chapter also discusses the numerous configuration parameters necessary for the convergence rate, quality of solutions, and neighbourhood solution search space in the model. After that, the sub-model scenarios are described and compared to evaluate the model's performance and resiliency (ability to produce a similar final optimized solution from different starting solutions). Later, all the findings are presented.

4.1 Configuration Parameters

Several researchers from different fields have concluded that simulation annealing requires fine-tuning the various parameters involved in the algorithm to achieve good results (Banchs, 1997c; Park and Kim, 1998; Catoni, 1998; Kundu et al., 2008; Bellio et al., 2016; Jackson et al., 2017). As is true with any SA algorithm, the SA algorithm in this thesis uses numerous parameters that must be adjusted and set. It is imperative to set these simulated annealing parameters appropriately to reduce run times and reduce the neighbourhood search space. The various parameters can be categorized into two groups:

4.1.1 Algorithm Specific Parameters

Specific parameters, such as these, are common to all implementations, i.e., they are unique to SA algorithms and have a certain influence on the quality of the optimal solution (Arenas et al., 2010).

- **INITIAL_TEMPERATURE**: A temperature value equal to this parameter determines the system's thermal energy in annealing thermodynamics. As mentioned in the previous subsubsection 3.9.2.1 a relatively high value should be set (Alrefaei and Andradóttir, 1999). The temperature should be reduced at an appropriate rate in an adequately scheduled cooling process. To set this parameter, we fixed a random temperature value and checked the acceptance ratio for early iterations to determine if we should keep the randomly fixed temperature. The acceptance ratio was the number of accepted worse solutions. The SA algorithm needs to accept less optimal solutions in early iterations to escape local minima. For all initial shift plan scenarios and their sub-models, the parameter was set to a fixed value: **10000**.

- **COOLING_SCHEDULE**: This parameter indicates how gradual the temperature decrease should be at each simulation iteration. Thus, the temperature should be close to 0 or the predefined final temperature upon completing all simulation iterations. Since **Exponential** cooling schedule is a commonly chosen scheme because of its robustness, malleability, and fast performance (Gonzales et al., 2015; Catoni, 1998), it was also chosen for the shift optimizer model. The thesis study also acknowledges the exponential cooling scheme's capabilities and uses the scheme with varying cooling speeds. The speeds were altered by changing the **ALPHA** values, which were within the range 0.85 and 0.99. The lower the **ALPHA** value, the faster the cooling and vice versa. Therefore, **EXPONENTIAL** cooling schedule with different cooling speeds was chosen for the Shift Optimizer model. Furthermore, it was also assessed later in the chapter what value of **ALPHA** gave the best results.

- **PERTURBATION_TYPE**: Perturbation describes changing an existing solution by

moving within its physical system or changing its state. Since each movement was governed by stochasticity (randomness), a new solution could be optimal or less optimal based on the cost function. The parameter described here is a method of using several perturbation techniques simultaneously in a single iteration. The method types are described as the following:

- **WEIGHTED**: The probability of selecting a particular perturbation strategy is influenced by its weights. The weights are pre-assigned by the user so that this method can perturb the solution majorly with strategies that higher weights and minorly with lesser weighted strategies. Therefore, larger weights will increase the likelihood of a perturbation function being applied to the current solution in every iteration and vice versa. Because all weights have to be assigned to each Perturbation Strategies method as percentages, the sum of their cumulative weights has to be equal to 100. Several *WEIGHTED* sequences with varied weights are discussed later in the chapter. Example: 4.1, that the last weight is the cumulative weight of 100, telling us that the chances of ***removeSAShifts*** is 10%, ***insertSAShifts*** is 10%, ***moveSABreakCorridor*** is 25%, ***increaseSAShiftTimings*** is 25% and so on.

Table 4.1: Example of a weighted perturbation type

Perturbation Strategy	Weights
REMOVE_SHIFT_WEIGHT	10
INSERT_SHIFT_WEIGHT	20
MOVE_BREAK_CORRIDOR_WEIGHT	45
MOVE_SHIFT_TIMINGS_WEIGHT	70
INCREASE_SHIFT_TIMINGS_WEIGHT	85
DECREASE_SHIFT_TIMINGS_WEIGHT	100

- **RANDOM**: With this method type, perturbation techniques are applied randomly to the current solution. They do not have weights attached to any of the perturbation methods, as any strategy could perturb the solution. It would not be useful to use this method type as the many perturbations would result in useless neighbourhoods since they are influenced by specific perturbations techniques. It is only appropriate to use the method if the user is uncertain where the optimal solution lies in the neighbourhood.

(Hentschke and Reis, 2003) stated that the standard SA algorithms use random perturbations, but greedy perturbations were aimed at a particular search space and converge faster in them. Despite producing faster convergence, the results may worsen after getting stuck in a local minimum. **WEIGHTED** is a mix of both types, random and greedy. Perturbations weights in the sub-model were not derived from literature but

taken after trial and error. The WEIGHTED is also greedy in some sense because when the model is calibrated after several runs, appropriate weights can lead to faster convergence.

- **ITERATIONS**: This parameter specifies when should a new neighbour solution be generated to substitute the current solution. (Lundy and Mees, 1986) describes it as a state where it should be of some decreasing temperature value. The shift optimizer model in the thesis needs a relatively high number of iterations values to ensure convergence. Later in the chapter, the iteration values are analyzed for their importance in convergence. The number of iterations that yielded the best result is also identified.

4.1.2 Objective Function Parameters

The cost of solution is sensitive to the constraints discussed previously in the thesis in section 3.11 and has a significant influence in yielding optimal solutions. So, they also discussed here.

- **DRIVER_COST_PER_HOUR**: A soft constraint parameter of the objective function reflects the influence that the number of driver hours has on the solution cost. For this thesis model, 30 €/hr was considered.
- **COST_PER_REJECTION_PER_HOUR**: An additional soft constraint that contributes to keeping rejections per hour in check. As 10 €/hr is set for the model, it means that for every rejection in simulation, 10 will be added. The algorithm will therefore try to improve the solution as the cost rises.
- **PENALTY**: It refers to a hard constraint parameter that has been set to a high value to suddenly increase the solution's cost when a violation of the rejection rate occurs. The violation occurs when the rejection rate per hour consists of values that exceed the desired rate. The algorithm will always reject solutions with several violations. The rejection of such solutions leads to ultimately obtaining a solution with the least or no violations. The value is set to 9999 for all different sub-models and will be discussed for its significance.

4.1.3 Perturbation Strategy Parameters

Defining perturbation parameters specifies the specifications that control how an encoded individual (shift plan) is designed and the size of the neighbourhood search space for optimal solutions. New designs of an encoded solution lead to a new neighbour solution. The parameters are further divided into two groups, namely:

- Rigid Value Parameters: Although these parameters are changeable, they are fixed in the scope of the thesis. They will always be fixed to a constant value in all the simulation runs or sub-models. See table 4.2.
- Flexible Value Parameters: These parameters are also changeable and have been configured in the same way. The flexible parameters have a different value with different simulation runs or sub-models. See table 4.3.

The flexible parameter values in this table 4.3 are used in combination with the other. For example, the SHIFTS_INSERTION and SHIFTS_REMOVAL are simulated together with different configurations in the sub-models: [2, 2], [10, 10], [20, 20]. Here, a sub-model will be set to 2 for SHIFTS_REMOVAL and 2 for SHIFTS_INSERTION. They are not set to 2 removals with 10 insertions or 20 removals with 2 insertions. The values are kept the same for both removals and insertions. Even though like described before, the configuration parameters are **Fixed** in the algorithm-specific and objective-function parameters or **Rigid** in perturbation-function parameters, they are not always constant. These configuration parameters can be changed to other values for different sub-model scenarios, different study areas, different settings, or even different analyses. However, in the scope of this thesis, the parameters that have either been Fixed or Not Fixed are described below, showing an overview:

Table 4.2: Perturbation parameter descriptions and their **rigid** values

Perturbation Parameters	Description	Value
TIME_INTERVAL	Time bin size. Important for encoding shift and break schedule appropriately.	$1/2$ hr or 1800
START_SERVICE_TIME	Time at which DRT service starts its operations and first key of the value **0** in the encoded shift (Figure 3.6)	0
END_SCHEDULE_TIME	The time when DRT service stops it operations and last key of the value **0** in the encoded shift (Figure 3.6)	30 hr or 108000
SHIFT_TIMINGS_-MAXIMUM_LENGTH	Maximum duration of a shift. The difference between start and end times should always be smaller than this value	8.5 hr or 30600
SHIFT_TIMINGS_-MINIMUM_LENGTH	Minimum duration of a shift. The difference between start and end times should always be greater than this value	5.5 hr or 19800
BREAK_CORRIDOR_-LENGTH	Predefined fixed duration of a break within a shift	$1/2$ hr or 1800
BREAK_CORRIDOR_-BUFFER	The intermediate duration between: 1) start time and earliestStartBreakTime 2) end time and lastestEndBreakTime	2 hr or 7200
SHIFTS_MINIMUM	To maintain a size fluctuation above this value, a solution must maintain a minimum number of shifts	1
SHIFTS_MAXIMUM	To maintain a size fluctuation below this value, a solution must maintain a maximum number of shifts	100
SHIFT_TIMINGS_BUFFER	The intermediate duration between: 1) START_SERVICE_TIME and start time 2) END_SCHEDULE_TIME and end time	0

Table 4.3: Perturbation parameter descriptions and their **flexible** values

Perturbation Parameters	Description	Value Range
SHIFTS_INSERTION	A random integer number less than this value determines the number of shifts that will be inserted into the solution	[2, 10, 20]
SHIFTS_REMOVAL	A random integer number less than this value determines the number of shifts that will be removed from the solution	[2, 10, 20]

Table 4.4: Overview of Fixed or Not Fixed configuration parameters in the scope of the thesis

Configuration Parameters	Fixed or Not Fixed
INITIAL_TEMPERATURE	Fixed at 10000
COOLING_SCHEDULE	Fixed at EXPONENTIAL cooling system
PERTURBATION_TYPE	Fixed at WEIGHTED perturbation strategies, but Not Fixed with weights
ITERATIONS	Not Fixed
DRIVER_COST_PER_HOUR	Fixed at 30
COST_PER_REJECTION_PER_HOUR	Fixed at 10
PENALTY	Fixed at 9999
All *rigid* Perturbation Parameters	Fixed at their respective settings described in table 4.2
All *flexible* Perturbation Parameters	Not Fixed at their respective settings described in table 4.3

4.2 Model Evaluation Procedure

An empirical approach is used to assess the implementation of the **Simulated Annealing algorithm's** strength and performance in the **Shift Optimizer model**. The strength or resiliency of an SA algorithm is given by its ability to reach the same optimized solution from different input variables (initial solutions) while keeping all other input variables constant. On the other hand, the performance of an SA algorithm is based on its execution time, complexity, and computational effort (Akinwale et al., 2012). The performance of algorithms is typically evaluated by comparing these metrics with other algorithm's. However, since Simulated Annealing is the only algorithm implemented in this thesis, the factors that influence the measurements have been discussed.

To avoid confusion, a **sub-model** is different from a **shift optimizer model** as it has different configuration values and their specific initial solution. So, a sub-model is a copy of the general shift optimizer model with different parameters.

The strength of the SA algorithm is tested by using different initial shift plans (solutions) in each sub-model. It is measured based on the same shift size (number of shift (and break) schedules), reduced rejection rate under 0.2 and very low average rejection rate. Therefore, if a sub-model solves with similar numbers, the model is resilient or strong. Not many sub-models were required to be run for testing, but they were tested with only three sub-models for the three different starting solutions (5, 30, 60).

Creating a starting solution: Each shift plan is manually created with a different number of random driver shift and break schedules, then classified into three different scenarios, namely **5_shifts**, **30_shifts** and **60_shifts**. For example, the 5_shifts scenario, it means, the initial Shift Plan will randomly contain 5 shift (and break) schedules. As evident from the prefix value on "_shift", it represents the number of shift and break schedules created.

In contrast, the parameter's influence on the algorithm's performance is examined by running several sub-models. The influence of Fixed values (given in table 4.4) are not evaluated explicitly in this thesis but were discussed on their performance likelihood.

4.3 Sub-Model Configuration and Description

All of the sub-model combinations and their various parameter settings that were simulated and analyzed during the thesis are presented in table 4.5. Each sub-model combination was simulated for each initial (5, 30, 60) solution. Thus, 3 X 54 models were run in total during the thesis.

The general idea behind these configurations is that the **Not Fixed** parameters are different in different sub-model. All the values were arbitrarily chosen or decided after several preliminary test runs (hit and trial). They were not based on literature, as SA algorithms are usually problem-specific.
PS: the names of the parameters variables have been changed to lower case to fit the page. Here is the explanation of each parameter in the sub-model:

1. **configuration**: It denotes each combination of the configuration or the sub-model number.

2. **alpha**: It is a factor representing the speed at which the **Exponential** cooling occurs. Arbitrary numbers between [0.8, 0.99] were considered: **0.8, 0.85, 0.88, 0.9, 0.95, 0.99**. These small differences between the numbers hugely affect the temperature reduction graphs. Selecting an appropriate alpha value is important for the algorithm's convergence rate.

3. **rejection_cost**: Reason for choosing the cost has been described here in subsubsection 3.11.1.2

4. **penalty**: This parameter value influences the **maximum rejection rate** of the finally accepted solution. The previous discussion on PENALTY established that a high penalty value is good for the cost function; setting it to 1000 instead of 9999 gave poor maximum rejection rate values when tested in the preliminary test runs. The reason for choosing cost has been described in subsubsection 3.11.2.2.

5. **driver_cost**: Reason for choosing cost has been described in subsubsection 3.11.1.2.

6. **shift_min**: The parameter denotes the minimum number of driver shift (and break) schedules that need to be kept in the optimized solution. The value should be kept to a minimum as the number of shift (and break) schedules that improves the solution's cost is unknown. Naturally, setting the parameter set to the fleet size is beneficial in more extensive study areas. However, low numbers in such large scenarios will only increase the computational time as the algorithm will search in regions where it is not required.

7. **shift_min**: The parameter denotes the maximum number of driver shift (and break) schedules that need to be kept in the optimized solution. The parameter should be

set to a reasonably large number. Too small numbers will lead to failed searches as one cannot predict the number of good shift (and break) schedules in an optimized solution. Similarly, a very high value would increase the run time of algorithm searching in unwanted regions.

8. **shift_removal**: The value determines the random number of shift (and break) schedules removed from the solution during perturbation. In theory, the value should not be above the number of shifts **"n"** in the initial shift plan. So when a random number above **"n"** gets selected, it would manipulate the solution and waste run time. A too low number compared to the initial solution is also not good, as it will not make much of a difference to the solution. That is why (2, 10, 20) were chosen to check if 1) **10** or **20** are too big for **5_shift** scenario setting or **2** is too small for **60_shift** scenario setting.

9. **shift_insertion**: Similarly, in theory, the insertion parameter should also depend on the shift size of the initial shift plan (solution). The value determines the number of shift (and break) schedules inserted into the solution to find a new neighbour solution. Since the algorithm does not specifically insert good shift (and break) schedules into the solution, keeping them low in the shift optimizer model was better. Alternatively, it is better to keep the weights of insertion perturbation low.

10. **weights of perturbation strategies**: Various weighting patterns are tested in the model algorithm to search for new solutions. In theory, all perturbations strategies other than *remove_weight* and *insert_weight* perturbation techniques should have higher weights as they manipulate the already present driver shift (and break) schedules in the solution and are less sensitive to the solution's cost. In this sense, there is only a slight change in the cost. In contrast, inserting or removing shift (and break) schedules from the solution changes the derived active shifts per hour value drastically, causing major changes in the cost. The changes are not slight when using these perturbation techniques.

11. **iterations**: This parameter defines the algorithm's exploration and exploitation time. In theory, SA algorithms find the best solution in infinite time or iteration when that time is quite high. So the sub-models are tested with different iteration values ranging from 200 to 800.

There are three metrics that defined the shift optimizer model's strength (Please refer to the table 4.6 for results):

1. accepted_shift_size: It is the size or number of driver shift (and break) schedules in the final output solution (optimized).

2. accepted_cost: The final cost of the final optimized solution.

3. accepted_max_rejection_rate: The maximum rejection rate of the final optimized solution that denotes the largest value in the rejection rate per hour variable.

4. accepted_average_rejection_rate: The average rejection rate of the final optimized solution that denotes the mean value in the rejection rate per hour variable.

The figure in Appendix: 1a shows the iterative progression of the cost of the least-cost accepted solution with the cost of varying simulated current solutions. As one can see, it is the accepted solution that is becoming the current solution at iterations with a low cost. Similarly, the figure in Appendix: 1b shows hows the iterative progression of the least-cost accepted solution with its maximum rejection rate and average rejection rate values.

Table 4.5: All sub-models with the different *Fixed* and *Not Fixed* parameter settings

configuration	alpha	rejection_cost	penalty	driver_cost	shifts_min	shifts_max	shifts_removal	shifts_insertion	remove_weight	insert_weight	move_break_weight	move_shifts_weight	increase_weight	decrease_weight	iterations
1	0.8	10	9999	30	1	100	2	2	10	45	60	75	85	100	200
2	0.8	10	9999	30	1	100	10	10	10	45	60	75	85	100	200
3	0.8	10	9999	30	1	100	20	20	10	45	60	75	85	100	200
4	0.85	10	9999	30	1	100	2	2	10	45	60	75	85	100	200
5	0.85	10	9999	30	1	100	10	10	10	45	60	75	85	100	200
6	0.85	10	9999	30	1	100	20	20	10	45	60	75	85	100	200
7	0.88	10	9999	30	1	100	2	2	10	45	60	75	85	100	200
8	0.88	10	9999	30	1	100	10	10	10	45	60	75	85	100	200
9	0.88	10	9999	30	1	100	20	20	10	45	60	75	85	100	200
10	0.9	10	9999	30	1	100	2	2	10	45	60	75	85	100	200
11	0.9	10	9999	30	1	100	10	10	10	45	60	75	85	100	200
12	0.9	10	9999	30	1	100	20	20	10	45	60	75	85	100	200
13	0.95	10	9999	30	1	100	2	2	10	45	60	75	85	100	200
14	0.95	10	9999	30	1	100	10	10	10	45	60	75	85	100	200
15	0.95	10	9999	30	1	100	20	20	10	45	60	75	85	100	200
16	0.99	10	9999	30	1	100	2	2	10	45	60	75	85	100	200
17	0.99	10	9999	30	1	100	10	10	10	45	60	75	85	100	200
18	0.99	10	9999	30	1	100	20	20	10	45	60	75	85	100	200
19	0.8	10	1000	30	1	100	2	2	10	20	45	70	85	100	300
20	0.8	10	1000	30	1	100	10	10	10	20	45	70	85	100	300
21	0.8	10	1000	30	1	100	20	20	10	20	45	70	85	100	300
22	0.8	10	9999	30	1	100	2	2	10	20	45	70	85	100	300
23	0.8	10	9999	30	1	100	10	10	10	20	45	70	85	100	300
24	0.8	10	9999	30	1	100	20	20	10	20	45	70	85	100	300
25	0.8	10	1000	30	1	100	2	2	25	35	45	70	85	100	300
26	0.8	10	1000	30	1	100	10	10	25	35	45	70	85	100	300
27	0.8	10	1000	30	1	100	20	20	25	35	45	70	85	100	300
28	0.9	10	1000	30	1	100	2	2	10	20	45	70	85	100	300
29	0.9	10	1000	30	1	100	10	10	10	20	45	70	85	100	300
30	0.9	10	1000	30	1	100	20	20	10	20	45	70	85	100	300
31	0.9	10	9999	30	1	100	2	2	10	20	45	70	85	100	300
32	0.9	10	9999	30	1	100	10	10	10	20	45	70	85	100	300
33	0.9	10	9999	30	1	100	20	20	10	20	45	70	85	100	300
34	0.9	10	1000	30	1	100	2	2	25	35	45	70	85	100	300
35	0.9	10	1000	30	1	100	10	10	25	35	45	70	85	100	300
36	0.9	10	1000	30	1	100	20	20	25	35	45	70	85	100	300
37	0.88	10	9999	30	1	100	2	2	10	15	20	55	85	100	400
38	0.88	10	9999	30	1	100	10	10	10	15	20	55	85	100	400
39	0.88	10	9999	30	1	100	20	20	10	15	20	55	85	100	400
40	0.88	10	9999	30	1	100	2	2	10	15	20	55	85	100	800
41	0.88	10	9999	30	1	100	10	10	10	15	20	55	85	100	800
42	0.88	10	9999	30	1	100	20	20	10	15	20	55	85	100	800
43	0.88	10	9999	30	1	100	2	2	15	20	30	55	85	100	400
44	0.88	10	9999	30	1	100	10	10	15	20	30	55	85	100	400
45	0.88	10	9999	30	1	100	20	20	15	20	30	55	85	100	400
46	0.88	10	9999	30	1	100	2	2	15	20	30	55	85	100	800
47	0.88	10	9999	30	1	100	10	10	15	20	30	55	85	100	800
48	0.88	10	9999	30	1	100	20	20	15	20	30	55	85	100	800
49	0.88	10	9999	30	1	100	2	2	5	10	20	50	80	100	400
50	0.88	10	9999	30	1	100	10	10	5	10	20	50	80	100	400
51	0.88	10	9999	30	1	100	20	20	5	10	20	50	80	100	400
52	0.88	10	9999	30	1	100	2	2	5	10	20	50	80	100	800
53	0.88	10	9999	30	1	100	10	10	5	10	20	50	80	100	800
54	0.88	10	9999	30	1	100	20	20	5	10	20	50	80	100	800

Table 4.6: Model results of sub-models (37 to 54) with different initial solutions (5_shift, 30_shifts, 60_shifts)

configuration	initial_shift_size	accepted_shift_size	accepted_cost	accepted_max_rejection_rate	accepted_average_rejection_rate
37	5_shifts	8	2040	0.15	0.022079125
38	5_shifts	9	2295	0.142857143	0.01809604
39	5_shifts	7	1960	0.185185185	0.028849206
40	5_shifts	7	1975	0.19047619	0.034845343
41	5_shifts	9	12229	1	0.142906998
42	5_shifts	9	2445	0.185185185	0.049686949
43	5_shifts	8	2055	0.185185185	0.025374379
44	5_shifts	8	22458	1	0.081661898
45	5_shifts	7	1980	0.185185185	0.016071429
46	5_shifts	6	32087	1	0.085421724
47	5_shifts	8	1875	0.185185185	0.167321343
48	5_shifts	8	2035	0.1	0.0140384
49	5_shifts	14	2705	0.175	0.020981538
50	5_shifts	8	12274	1	0.046900871
51	5_shifts	8	1920	0.175	0.008227513
52	5_shifts	8	1925	0.185185185	0.015113035
53	5_shifts	8	11904	1	0.11622575
54	5_shifts	8	32722	1	0.094772727
37	30_shifts	7	1755	0.178571429	0.051989071
38	30_shifts	8	1970	0.1	0.004567901
39	30_shifts	9	1985	0.148148148	0.017896825
40	30_shifts	8	2055	0.185185185	0.021038961
41	30_shifts	7	1700	0.111111111	0.013910935
42	30_shifts	8	1870	0.148148148	0.016748838
43	30_shifts	8	2170	0.148148148	0.040116615
44	30_shifts	7	1905	0.185185185	0.128086045
45	30_shifts	7	11819	1	0.299374633
46	30_shifts	8	1650	0.175	0.176078976
47	30_shifts	8	1995	0.175	0.033787478
48	30_shifts	8	2070	0.090909091	0.013422625
49	30_shifts	40	7350	0.148148148	0.019382716
50	30_shifts	11	2540	0.15	0.012006173
51	30_shifts	11	2445	0.185185185	0.025955726
52	30_shifts	24	14504	1	0.044029982
53	30_shifts	9	2340	0.185185185	0.041541842
54	30_shifts	9	2070	0.175	0.0314903
37	60_shifts	10	2070	0.142857143	0.03218201
38	60_shifts	8	1865	0.1	0.010808551
39	60_shifts	9	2295	0.185185185	0.078225349
40	60_shifts	8	1970	0.175	0.029655516
41	60_shifts	7	1600	0.175	0.238902801
42	60_shifts	8	1950	0.125	0.012407407
43	60_shifts	8	1905	0.148148148	0.024049291
44	60_shifts	9	1835	0.185185185	0.015656966
45	60_shifts	9	2015	0.111111111	0.017418202
46	60_shifts	8	2105	0.185185185	0.021210089
47	60_shifts	8	1955	0.185185185	0.036550964
48	60_shifts	11	2510	0.148148148	0.545852
49	60_shifts	55	9960	0.185185185	0.009907407
50	60_shifts	11	2115	0.185185185	0.014475309
51	60_shifts	11	2085	0.125	0.015019309
52	60_shifts	49	8960	0.175	0.010771605
53	60_shifts	8	1665	0.125	0.003530378
54	60_shifts	10	2030	0.185185185	0.02799807

4.4 Findings

The table 4.6 shows the aggregated results of the final outputs generated from the sub-models: 37 to 54. As mentioned previously, these sub-models produced good results. Here it has been discussed on why they were good. The good results are in terms of:

- Since the model's objective was to reduce the driver hours, the algorithm reduced the number of shift (and break) schedules in the optimized solution for almost all sub-models. Not only that, most of the sub-models have arrived at more or less the same shift size solution. This same search indicates the shift optimizer model's resiliency to initial solutions. Because of static demand, they should reach the same optimized solution.
- The other goal of the SA algorithm was to bring down the rejections per hour under the **DESIRED_REJECTION_RATE** value. It also did for most cases, except in cases where it became 1, which may be because the neighbour solutions were not the best in the algorithm's space. It may have been stuck in a local minimum. Further analysis is still needed to reveal the truth.
- The other implicit aim was also realized after reducing the average rejection rate due to the optimized solution. That is why the model's objective function focused on reducing the maximum rejection rate by placing the penalty factor in the function, as the average rejection rate would have easily reduced simultaneously.

Therefore, the model works for most sub-models but is unsuccessful on a few of them. Let's discuss the observations that were seen after running all sub-models:

- The number of MATSim **iterations** definitely played a huge role in determining the run time of the models. As literature also claims that SA guarantees a converges in infinite time. It was also observed that a higher number of iterations lead to better results, as seen in the table 4.6. The top three optimal cost solutions were found in the sub-models with 800 iterations. Not only that, the least average rejection rate and maximum rejection rate were also in sub-models configured with 800 iterations.
- One major reason why these 18 (37 to 54) sub-models shown in the table 4.4 yielded superior results was because the perturbation **weights** were kept higher for ***removeSAShifts***, ***increaseSAShiftTimings*** and *decreaseSAShiftTimings* techniques. The algorithm spent more time perturbing these techniques. These perturbations tend to manipulate the active shifts per hour value very easily. These then slightly affected the cost solution, keeping the neighbours' solution close. However, removing and inserting techniques were very sensitive to the cost of the solution; for example, even a single insertion of driver shift could have quickly increased the cost of the solution due to its direct effect on the cost. This happens because the driver_cost acts as a weight and multiplication factor

to the additional driver shift insertions resulting in higher costs. One can argue that the cost will reduce if good driver schedules are added, which is also true. Nevertheless, the insertion technique was not the main focus during the study, so it was not developed appropriately to add suitable shifts into the solution. Thus, slight perturbations are best for the algorithm.

- The **number of insertions and removals** also affected the solution's quality and algorithm's convergence rate. After careful analysis of sub-model results, it is seen that this number should be based on the size of the initial solution of the sub-model. A number larger than the size of the initial solution will not perturb the solution, or a number too small will have no or negligible effect on the solution. As seen from the table 4.6, for 5_shifts sub-model, 53^{th} and 54^{th} configurations produced the worst results in terms of shift size while the 52^{th} configuration with 2 shift removals and insertions was superior. It may be because the initial solutions were never perturbed gradually but suddenly during the iterations. Alternatively, the 60_shifts sub-model did not see this problem as the number of insertions and removals were well within 60. Even though, 53^{th} and 54^{th} configurations had high shift size, their cost was low. The large shift size may be because of several small-length driver shift (and break) schedules present in the solution. However, it is known for sure why this would have occurred.

- HHigher **alpha** values (like 0.99) did not produce a low-cost solution, as proclaimed by (Peprah et al., 2017), other than a gradually cooling the temperature value. The slow cooling schemes are similar to a linear decrease in temperature, which, as we know (from subsubsection 3.9.2.2), will lead to a slow convergence rate due to increased exploration time. Moreover, fast cooling schedules are not good for the algorithm as it needs high-temperature values to accept a bad solution in early iteration. Thus, 0.88, that is within 0.8 and 0.99, was chosen in the sub-models in table 4.6 which produced promising results. Although the table shows positive outcomes, it is still not sufficient to claim that the cooling rate increased the performance.

- Since the **driver_cost** and **rejection_costs** were not altered during any of the sub-models, it is not known how sensitive these values are to the solution. However, the **penalty** value was changed and tested in the preliminary runs. It was noticed that high penalty values result in reduced maximum rejection rate. The reason is that high penalties means high costs per violation in rejection rate per hour. Aditionally, since the violation is to exceed 0.2 (DESIRED_REJECTION_RATE), solutions with many violations will have a very high cost, thereby forcing the algorithm to look for better solutions with no or few violations.

- It is also seen that sub-models concerning 5_shift scenarios in the table 4.6, resulted in a very high costs and low shift sizes (41^{th}, 44^{th}, 46^{th}, 50^{th}, 52^{th}, 53^{th}, sub-models). It

is the expected behaviour, as small shift sizes will produce high rejections, increasing the cost. However, what is concerning is that the *max rejection rate* isn't under 0.2 for given sub-models. It may have happened either because the algorithm was getting stuck in local minima or because the perturbation techniques did not work well in the sub-models. A major limitation is the low test analysis of such concerning sub-models. The sub-models needs to be rerun and tested to understand such discrepancies.

Chapter 5

Conclusion

To conclude the thesis, that begets the research question: Can a heuristics algorithm like Simulated Annealing optimize shift and break schedules of drivers in ride-pooling services? Based on the promising results of a few specific models presented in the thesis, it is safe to assume that Simulated Annealing can optimize drivers' schedules.

The model was simulated in MATSim several times, whose analyzed results indicated the model's robustness and performance. The simulations were carried out on the Study Area of Holzkirchen town in Germany. The final solution of a few calibrated sub-models was quite promising in having similar yet low shift sizes and low rejection rates for different starting solutions. The maximum rejection rate in these sub-models was also under the predefined desired rejection rate value of 0.2. Furthermore, the average rejection was also relatively low. Reaching a similar optimized solution after changing the initial solutions indicates that these sub-models or the model itself were resilient and robust. The other observations were that these sub-models performed well with specific perturbation techniques when the solution was perturbed majorly with them. Higher penalty values had helped in the reduction of maximum rejection rate significantly, rapidly fast cooling schedules were not suitable for the algorithm, the number of iteration was directly proportional to the run time of the model, and the number of removals and insertion parameters should be well under the initial solution's shift size.

Since the few sub-models that were calibrated on preliminary test runs had performed well, it is safe to assume that the shift optimizer model will get better with more model tests runs. Moreover, more test analyses are needed to confidently estimate the models' parameters.

The final conclusion is that even though it showed a positive outcome, the model comes with a set of its own limitations and limitations of the Simulated Annealing algorithm that it employs. They have been discussed later in the chapter with the suggestions for future work.

5.1 Model Limitations

Like most models, this shift optimizer model is not without its limitations which are discussed below. The model's disadvantage is mainly due to the Simulated Annealing algorithm involved in it and partially due to the primary development of it.
These are by:

- The Stimulated Annealing algorithm used in the model itself has several disadvantages such as:

    1. **Run time**: As cited in (Abramson, 1991; Banchs, 1997c; Catoni, 1998; Aarts et al., 2005)], the algorithm requires much computational time to explore and exploit the search space. In this case, the MATSim iterations need to be high for MATSim to co-evolve with the shift schedules, undermining MATSim's benefits as a fast and dynamic simulation system. The run times of SA are high because of the algorithm's stochastic nature. The heuristic needs to be tailored to a specific problem to solve the problem. The processing time is significantly reduced when Simulated Annealing algorithms are combined with other problem-specific greedy heuristics. It is also possible to reduce the burden on computational resources by using parallel computing methods (Chu et al., 1999).
    2. **Tuning multiple parameters**: SA involves numerous parameters, as we have seen in section 4.1, and many authors also mention it in (Arenas et al., 2010; Jackson et al., 2017). Fine-tuning these parameters happen to be another problem. Furthermore, not tuning the parameters leads to a failed convergence in a limited time. Moreover, the cooling scheduling needs to be an appropriate reduction scheme as it dictates the algorithm's convergence rate for finding the optimal solution. Dynamic cooling schedules could yield better results in lesser times (Aarts and Van Laarhoven, 1985). Setting suitable perturbation strategies and their weights seems crucial in finding good results like in the table 4.6. Perturbation functions create a valuable neighbourhood space for a combinatorial optimization problem like the shift optimization model.
    3. **Initializing temperature and solution**: The initial temperature is crucial to the algorithm's ability to avoid local minima by accepting worse solutions in early iterations (Kirkpatrick et al., 1983). (Ben-Ameur, 2004) proposes a method to calculate the initial temperature that is consistent with the algorithm's acceptance ratio while accounting for all cooling rates. Single solution-based techniques like SA are difficult to use without a good starting point, and an unfeasible starting point will lead to failed searches (Shojaee. et al., 2010).

- There are also three major disadvantages that addresses the issues with the proposed model are:

1. **Small scenario**: The optimizer model has been tested on a small network with positive results in most cases; however, it cannot be guaranteed that the model will yield the same positive results for a more extensive network. Additionally, since the model involves tuning many configuration parameters for its performance, its parameters do not need to work with an extensive network. In that case, it is necessary to re-calibrate the model.
2. **Static demand**: As the model was built on static demand, it can be enhanced further by including a realistic dynamic travel demand. Since this model uses the (Kuehnel et al., 2021-05) Shift and Break extension to simulate shift and break schedules, the limitations of that extension apply to this model as well.
3. **Initial version**: TThe model is in its early stage, so several areas need improvement, including the *insertSAShifts* perturbation technique that inserts random driver schedules that are already present in the solution, which may not be the best approach. Since most of the thesis time was devoted to developing and coding the model, not many runs were conducted. Further testing and analysis will be needed.

5.2 Future Work

- **Larger scenario**: In order to increase the model's confidence, it needs to be applied and tested on a larger study area. There is also a higher possibility that the network service area and the number of ride-pooling trip requests (demand) will increase in larger areas. It will be interesting to see how well the model performs in such a situation.
- **Addition of realistic elements**:Since static travel demand was used in developing, the model is not as accurate as it should be, so a dynamic demand is needed. In addition to this change, the simulation could include several other parameters such as the vehicle's energy consumption, infrastructure in the area, other DRT modes (electric cars or buses), and a combination of DRT modes and other modes.
- **Combination with other heuristics**: The shift optimizer model can be studied by implementing a hybrid algorithm that was studied in (Bailey et al., 1997; Widl and Musliu, 2010; Norgren and Jonasson, 2016) however, in the context of a ride-pool. As an alternative, apply a regression model to estimate rejection rates and improve the model's performance. The optimizer model uses the estimated rejection rate to predict the cost of solutions without actually simulating them. Multiple initial iterations can train the regression model to predict solutions without simulating and reducing run time gradually. Although this study was done during the thesis, it was not incorporated into the current optimizer model. Here is the example of the estimated rejection rate per hour in figure 3 and the actual rejection rate per hour in figure 3. The estimated rejection rate was based on the formula in (Militão and Tirachini, 2021).
- **Parametric sensitivity analysis**: Due to the model's algorithm's use of multiple configuration parameters, a wide range of studies can be conducted to study how sensitive these parameters are to the quality of the solution and the algorithm's convergence rate. During the thesis, only many parameters were Fixed, and only a few were Not Fixed. It would be beneficial to explore a model with more flexibility in parameter configurations.
- **Improving objective function**: A further improvement to the objective function (cost) could be achieved by adding variables such as VKT, detour time, or waiting time. All of these variables contribute to improving the efficiency of the ride-pooling system.

This thesis' research represents the first attempt to develop a heuristic algorithm to solve combinatorial scheduling problems in a ride-pooling situation. It is my hope that the thesis will establish the groundwork for future state-of-the-art research in the ride-pooling area.

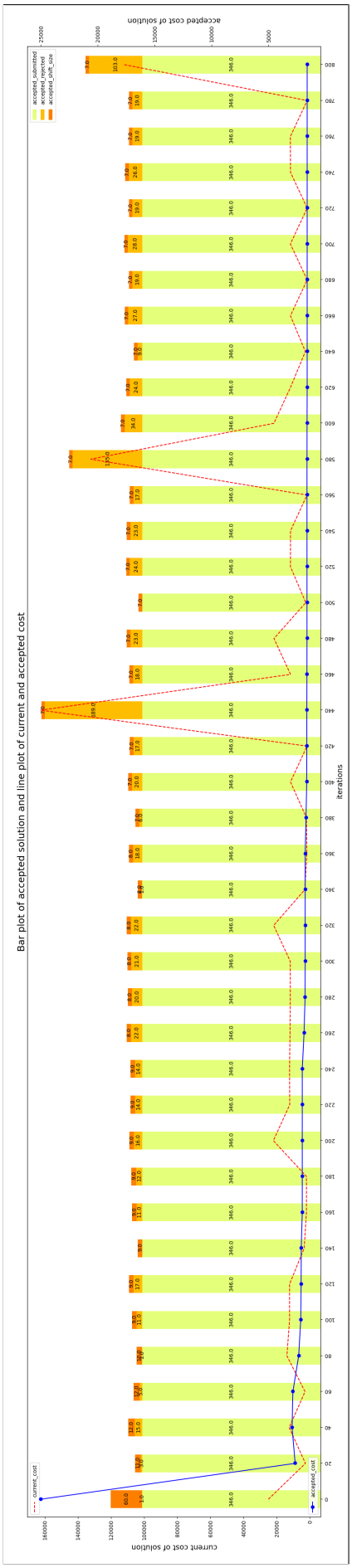
Statement of Independent Work

I hereby certify that this thesis was written independently by myself without using any other sources than those cited, and all passages and ideas taken from other sources are properly cited. No earlier versions of this thesis have been submitted for assessment elsewhere.

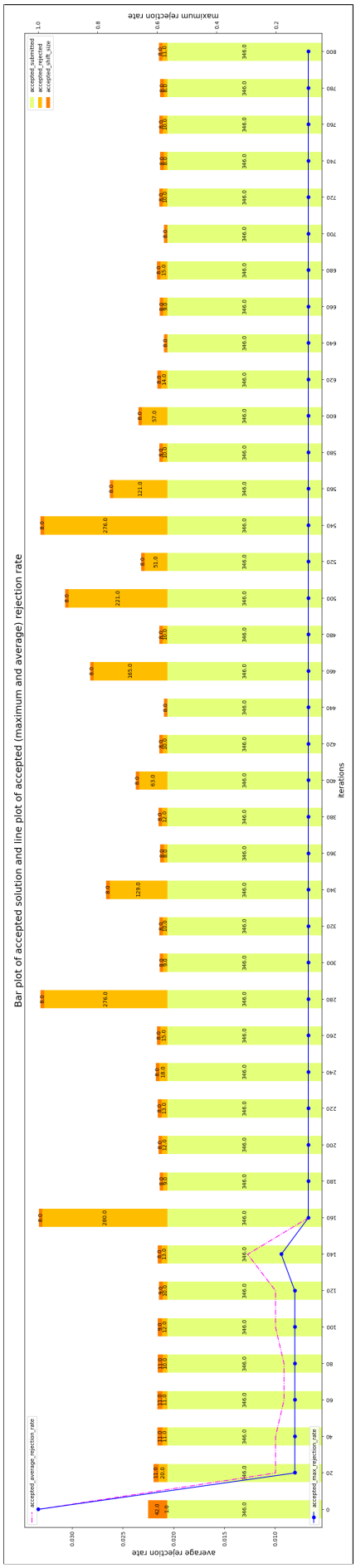
Shivam Arora
Technical University of Munich

Qin Zhang
Professorship for Modelling Spatial Mobility,
Technical University of Munich

Nico Kühnel
MOIA GmbH, Hamburg, Germany



(a) Bar plot of accepted solution through iterations and the costs of current and accepted solution - Based on the results of the configuration **12** of **60_shift** scenario setting in table 4.6



(b) Bar plot of accepted solution through iterations and the average and maximum rejection rate of the accepted solution - Based on the results of the configuration **5** of **30_shift** scenario setting in table 4.6

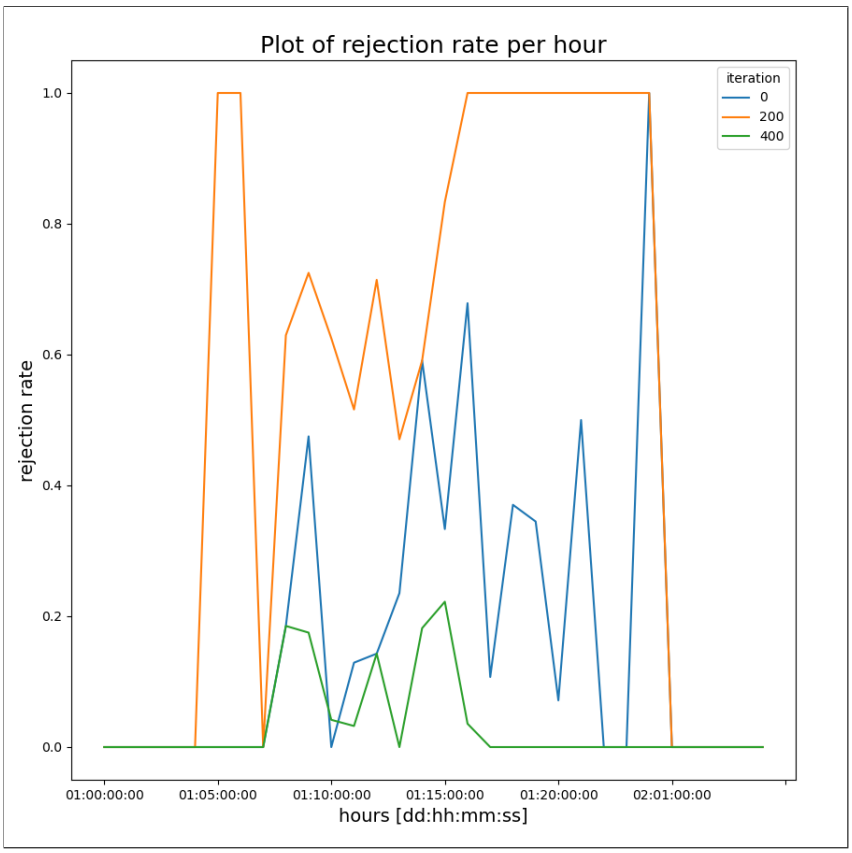


Figure 2: Plot of rejected rate per hour

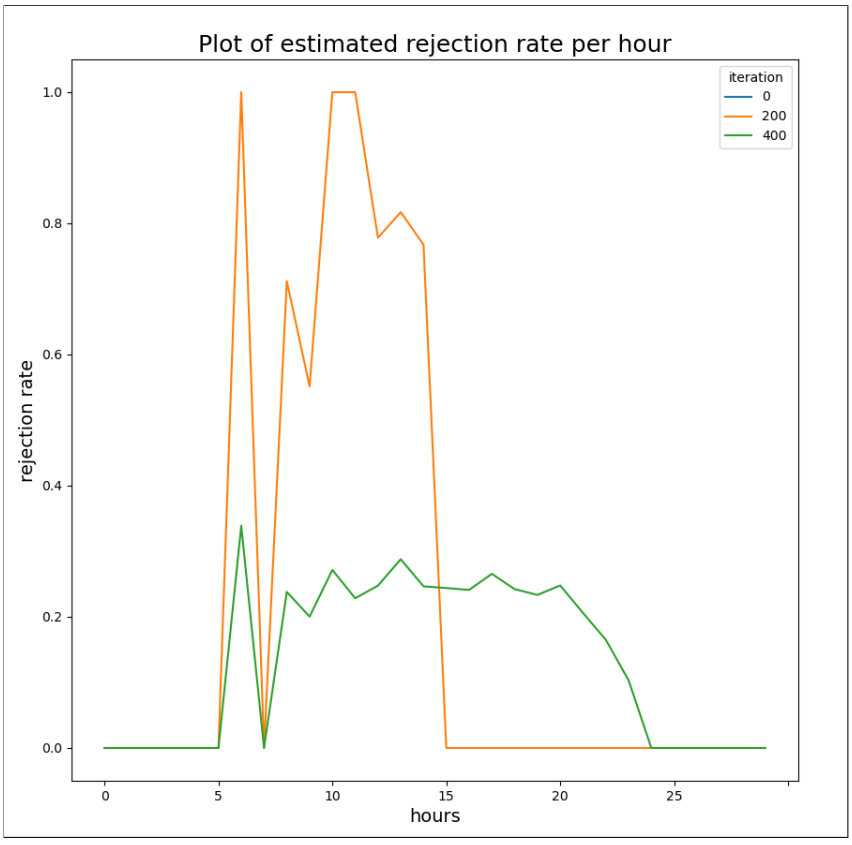


Figure 3: Plot of estimated rejection rate per hour

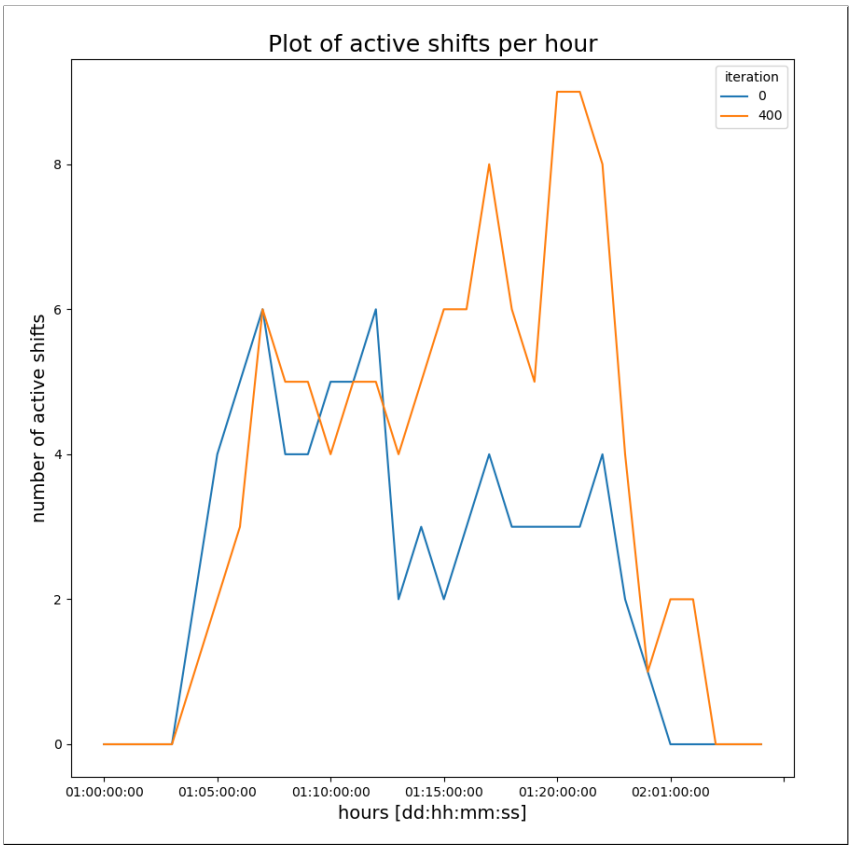


Figure 4: Plot of active shifts per hour

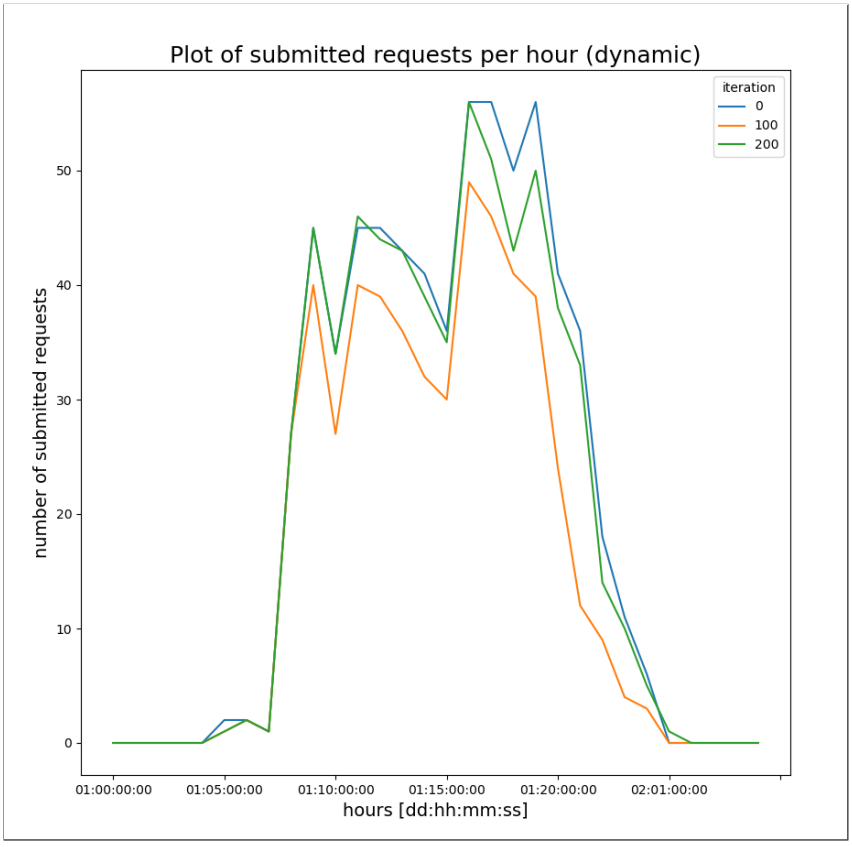


Figure 5: Plot of dynamic submitted requests per hour

References

E.H.L. Aarts, J.H.M. Korst, and W.P.A.J. Michiels. *Simulated annealing*, pages 187–210. Springer, Germany, 2005. ISBN 0-387-23460-8. doi: 10.1007/0-387-28356-0_7.

Emile HL Aarts and Peter JM Van Laarhoven. Statistical cooling: A general approach to combinatorial optimization problems. *Philips Journal of research*, 40(4):193–226, 1985.

David Abramson. Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management science*, 37(1):98–113, 1991.

Mauricio Acuna and J. Sessions. A simulated annealing algorithm to solve the log-truck scheduling problem. *Mathematical Research Summaries*, pages 87–88, 01 2017.

C Akinwale, S Olatunde, E Olusayo, J Babalola, et al. Performance evaluation of simulated annealing and genetic algorithm in solving examination timetabling problem. *Scientific Research and Essays*, 7(17):1727–1733, 2012.

Arjan Akkermans, Gerhard Post, and Marc Uetz. Solving the shift and break design problem using integer linear programming. *Annals of operations research*, pages 1–22, 2019.

Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114:201611675, 01 2017. doi: 10.1073/pnas.1611675114.

Mahmoud H Alrefaei and Sigrún Andradóttir. A simulated annealing algorithm with constant temperature for discrete stochastic optimization. *Management science*, 45(5):748–764, 1999.

M.G. Arenas, Juan Luis Laredo, Pedro Castillo, Pablo García-Sánchez, Antonio Mora, Alberto Prieto, and Juan Merelo Guervós. Statistical analysis of the parameters of the simulated annealing algorithm. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 07 2010. doi: 10.1109/CEC.2010.5586160.

Turgut Aykin. Optimal shift scheduling with multiple break windows. *Management Science*, 42 (4):591–602, 1996. URL `https://EconPapers.repec.org/RePEc:inm:ormnsc:v:42:y:1996:i:4:p:591-602`.

RN Bailey, KM Garner, and MF Hobbs. Using simulated annealing and genetic algorithms to solve staff-scheduling problems. *Asia-Pacific Journal of Operational Research*, 14(2):27, 1997.

Rafael E Banchs. "genetic algorithms. *Research progress report, on Time Harmonic Field Electric Logging Austin, University of Texas at Austin*, 1997a. URL `https://rbanchs.com/publications/reports.html`.

Rafael E Banchs. Gradient methods. *Research progress report, on Time Harmonic Field Electric Logging Austin, University of Texas at Austin*, 1997b. URL https://rbanchs.com/publications/reports.html.

Rafael E Banchs. Simulated annealing. *Research progress report, on Time Harmonic Field Electric Logging Austin, University of Texas at Austin*, 1997c. URL https://rbanchs.com/publications/reports.html.

Stephen E. Bechtold and Larry W. Jacobs. Implicit modeling of flexible break assignments in optimal shift scheduling. *Management Science*, 36(11):1339–1351, 1990. ISSN 00251909, 15265501.

Andreas Beer, Johannes Gärtner, Nysret Musliu, Werner Schafhauser, and Wolfgang Slany. Scheduling breaks in shift plans for call centers. In *Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling, Montréal, Canada*, 2008. doi: 10.1007/978-3-642-39304-4_5.

Andreas Beer, Johannes Gaertner, Nysret Musliu, Werner Schaffhauser, and Wolfgang Slany. An ai-based break-scheduling system for supervisory personnel. *IEEE Intelligent Systems*, 25 (2):60–73, 2010. ISSN 1541-1672. doi: 10.1109/MIS.2010.40.

Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, Andrea Schaerf, and Tommaso Urli. Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers & Operations Research*, 65:83–92, 2016.

Walid Ben-Ameur. Computing the initial temperature of simulated annealing. *Computational Optimization and Applications*, 29(3):369–385, 2004.

Mauro Birattari, Luis Paquete, Thomas Stützle, and Klaus Varrentrapp. Classification of metaheuristics and design of experiments for the analysis of components. *Teknik Rapor, AIDA-01-05*, 2001.

Joschka Bischoff and Michal Maciejewski. Proactive empty vehicle rebalancing for demand responsive transport services. *Procedia Computer Science*, 170:739–744, 2020.

Joschka Bischoff, Ninja Soeffker, and Michał Maciejewski. A framework for agent based simulation of demand responsive transport systems. Technical report, Technische Universität Berlin, 2016. URL http://dx.doi.org/10.14279/depositonce-5760.

Joschka Bischoff, Michal Maciejewski, and Kai Nagel. City-wide shared taxis: A simulation study in berlin. In *2017 IEEE 20th international conference on intelligent transportation systems (ITSC)*, pages 275–280. IEEE, 2017. doi: 10.1109/ITSC.2017.8317926.

Joschka Bischoff, Karoline Führer, and Michal Maciejewski. Impact assessment of autonomous drt systems. *Transportation Research Procedia*, 41:440–446, 2019.

Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

Daniil Bogdanov. A comparative evaluation of metaheuristic approaches to the problem of curriculum-based course timetabling, 2015.

Ihor O Bohachevsky, Mark E Johnson, and Myron L Stein. Generalized simulated annealing for function optimization. *Technometrics*, 28(3):209–217, 1986.

Alex Bonutti, Sara Ceschia, Fabio De Cesco, Nysret Musliu, and Andrea Schaerf. Modeling and solving a real-life multi-skill shift design problem. *Annals of Operations Research*, 252 (2):365–382, 2017.

Philippe Bruecker, Jorne Van den Bergh, Jeroen Beliën, and Erik Demeulemeester. Workforce planning incorporating skills: State of the art. *European Journal of Operational Research*, 243, 05 2015. doi: 10.1016/j.ejor.2014.10.038.

Edmund K Burke, Jingpeng Li, and Rong Qu. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, 203(2):484–493, 2010.

Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013. URL https://doi.org/10.1057/jors.2013.71.

Patrick M. Bösch, Felix Becker, Henrik Becker, and Kay W. Axhausen. Cost-based analysis of autonomous mobility services. *Transport Policy*, 64:76 – 91, 2018-05. ISSN 0967-070X. doi: 10.3929/ethz-b-000184754.

Olivier Catoni. Solving scheduling problems by simulated annealing. *SIAM Journal on Control and Optimization*, 36(5):1539–1575, 1998.

Vladimír Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985.

Nelson D Chan and Susan A Shaheen. Ridesharing in north america: Past, present, and future. *Transport reviews*, 32(1):93–112, 2012.

Ziyi Chen, Patrick De Causmaecker, and Yajie Dou. Neural networked assisted tree search for the personnel rostering problem. *CoRR*, abs/2010.14252, 2020. URL https://arxiv.org/abs/2010.14252.

King-Wai Chu, Yuefan Deng, and John Reinitz. Parallel simulated annealing by mixing of states. *Journal of Computational Physics*, 148(2):646–662, 1999.

Claudio Ciancio, Demetrio Laganà, Roberto Musmanno, and Francesco Santoro. An integrated algorithm for shift scheduling problems for local public transport companies. *Omega*, 75: 139–153, 2018.

Clevershuttle (2021). https://www.clevershuttle.de/, 2021. Accessed: 2021-10-05.

George B. Dantzig. A comment on edie's "traffic delays at toll booths". *Journal of the Operations Research Society of America*, 2(3):339–341, 1954. ISSN 00963984. URL http://www.jstor.org/stable/166648.

Demand Responsive Transit (DRT). https://matsim.org/apidocs/drt/0.9.0/, 2021. Accessed: 2021-05-02.

Luca Di Gaspero, Johannes Gärtner, Guy Kortsarz, Nysret Musliu, Andrea Schaerf, and Wolfgang Slany. The minimum shift design problem. *Annals of operations research*, 155(1): 79–105, 2007. doi: 10.1007/s10479-007-0221-1.

Luca Di Gaspero, Johannes Gärtner, Nysret Musliu, Andrea Schaerf, Werner Schafhauser, and Wolfgang Slany. Automated shift design and break scheduling. In *Automated scheduling and planning*, pages 109–127. Springer, 2013. doi: 10.1007/978-3-642-39304-4_5.

Leslie C. Edie. Traffic delays at toll booths. *Journal of the Operations Research Society of America*, 2(2):107–138, 1954. ISSN 00963984. URL http://www.jstor.org/stable/166599.

Richard W Eglese. Simulated annealing: a tool for operational research. *European journal of operational research*, 46(3):271–281, 1990.

MA Saleh Elmohamed, Paul Coddington, and Geoffrey Fox. A comparison of annealing techniques for academic course scheduling. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 92–112. Springer, 1997.

David Ennen and Thorsten Heilker. Ride-hailing services in germany: Potential impacts on public transport, motorized traffic, and social welfare. *Institute of Transport Economics Münster Working Paper*, 29, 2020.

A.T Ernst, H Jiang, M Krishnamoorthy, and D Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1): 3–27, 2004. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(03)00095-X. Timetabling and Rostering.

Javed Farhan and T. Donna Chen. Impact of ridesharing on operational efficiency of shared autonomous electric vehicle fleet. *Transportation Research Part C: Emerging Technologies*, 2018.

BR Fox and MB McMahon. Genetic operators for sequencing problems. In *Foundations of genetic algorithms*, volume 1, pages 284–300. Elsevier, 1991.

Bayrisches Landesamt für Statistik. Einwohnerzahlen am 31. Technical report, März 2019. Technical Report. Fürth. URL: https://www. statistik. bayern. de . . . , 2019.

Bundesministerium für justiz und verbraucherschutz. Arbeitszeitgesetz (arbzg, 2021. URL https://www.gesetze-im-internet.de/arbzg/BJNR117100994.html.

Johannes Gaertner, Nysret Musliu, and Wolfgang Slany. Rota: A research project on algorithms for workforce scheduling and shift design optimization. *AI Commun.*, 14:83–92, 07 2001.

Michael R Garey. A guide to the theory of np-completeness. *Computers and intractability*, 1979.

Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990. ISBN 0716710455.

Stuart Geman and Donald Geman. Geman, d.: Stochastic relaxation, gibbs distribution, and the bayesian restoration of images. ieee trans. pattern anal. mach. intell. pami-6(6), 721-741. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6:721–741, 11 1984. doi: 10.1109/TPAMI.1984.4767596.

Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986. ISSN 0305-0548. doi: https://doi.org/10.1016/0305-0548(86)90048-1. URL https://www.sciencedirect.com/science/article/pii/0305054886900481. Applications of Integer Programming.

Fred Glover and Claude McMillan. The general employee scheduling problem. an integration of ms and ai. *Computers & Operations Research*, 13(5):563–573, 1986. ISSN 0305-0548. doi: https://doi.org/10.1016/0305-0548(86)90050-X. Applications of Integer Programming.

Gill Velleda Gonzales, Elizaldo Domingues dos Santos, Leonardo Ramos Emmendorfer, Liércio André Isoldi, Luiz Alberto Oliveira Rocha, and Emanuel da Silva Diaz Estrada. A comparative study of simulated annealing with different cooling schedules for geometric optimization of a heat transfer problem according to constructal design. *Scientia Plena*, 11 (8), 2015.

Shivapratap Gopakumar, Sunil Gupta, Santu Rana, Vu Nguyen, and Svetha Venkatesh. Algorithmic assurance: An active approach to algorithmic testing using bayesian optimisation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 5470–5478, 2018.

Bruce Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988. ISSN 0364-765X. doi: 10.1287/moor.13.2.311.

Alejandro Henao and Wesley E Marshall. The impact of ride-hailing on vehicle miles traveled. *Transportation*, 46(6):2173–2194, 2019.

Renato Fernandes Hentschke and RADL Reis. Improving simulated annealing placement by applying random and greedy mixed perturbations [ic layout]. In *16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings.*, pages 267–272. IEEE, 2003.

ioki (2021). https://ioki.com/en/home/, 2021. Accessed: 2021-09-12.

Warren G Jackson, Ender Özcan, and Robert I John. Tuning a simulated annealing metaheuristic for cross-domain search. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1055–1062. IEEE, 2017.

L. W. Jacobs and S. Bechtold. Labor utilization effects of labor scheduling flexibility alternatives in a tour scheduling environment. *Decision Sciences*, 24:148–166, 1993.

Java Library. https://docs.oracle.com/javase/8/docs/api/java/util/Random.html, 2021. Accessed: 2021-07-22.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671. URL https://www.science.org/doi/abs/10.1126/science.220.4598.671.

Lucas Kletzander. *A Heuristic solver framework for the general employee scheduling problem*. PhD thesis, Wien, 2018.

Lucas Kletzander and Nysret Musliu. Solving the general employee scheduling problem. *Computers & Operations Research*, 113:104794, 08 2019. doi: 10.1016/j.cor.2019.104794.

Lucas Kletzander and Nysret Musliu. Scheduling bus drivers in real-life multi-objective scenarios with break constraints. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling-PATAT*, volume 1, 2021.

Deniz Kocabas. *Exact methods for shift design and break scheduling*. PhD thesis, TU Wien, 2015.

John R Koza and John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

Nico Kuehnel, Felix Zwick, and Sebastian Hörl. Shifts in perspective. operational aspects in (non-) autonomous ride-pooling simulations. "[Online; accessed 08-November-2021]", 2021-05.

Anit Kumar. Encoding schemes in genetic algorithm. *International Journal of Advanced Research in IT and Engineering*, 2(3):1–7, 2013.

S Kundu, M Mahato, B Mahanty, and S Acharyya. Comparative performance of simulated annealing and genetic algorithm in solving nurse scheduling problem. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, pages 96–100. Citeseer, 2008.

P.J.M. Laarhoven, Van and E.H.L. Aarts. *Simulated annealing : theory and applications*. Mathematics and its applications. Reidel, 1987. ISBN 90-277-2513-6.

Ziru Li, Yili Hong, and Zhongju Zhang. Do ride-sharing services affect traffic congestion? an empirical study of uber entry. *Social Science Research Network*, 2002:1–29, 2016.

Miranda Lundy and Alistair Mees. Convergence of an annealing algorithm. *Mathematical programming*, 34(1):111–124, 1986.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

Microscopic Transportation Orchestrator (MITO). `https://github.com/msmobility/mito`, 2021. Accessed: 2021-11-20.

Aitan M Militão and Alejandro Tirachini. Optimal fleet size for a shared demand-responsive transport system with human-driven vs automated vehicles: A total cost minimization approach. *Transportation research part A: policy and practice*, 151:52–80, 2021.

MIT. Rideshare history and statistics. `https://ridesharechoices.scripts.mit.edu/home/histstats/`, 2009. [Online; accessed 08-November-2021].

Rolf Moeckel, Nico Kuehnel, Carlos Llorca, Ana Tsui Moreno, and Hema Rayaprolu. Agent-based simulation to improve policy sensitivity of trip-based models. *Journal of Advanced Transportation*, 2020, 2020. doi: https://doi.org/10.1155/2020/1902162.

MOIA (2021). `https://www.moia.io/`, 2021. Accessed: 2021-12-13.

MOIA's Driver advertisement. `https://www.moia.io/de-DE/fahrer`, 2021. Accessed: 2021-12-20.

Shyan L Moondra. An lp model for work force scheduling for banks. *Journal of Bank Research*, 7(4):299–301, 1976.

Ana Tsui Moreno and Rolf Moeckel. Population synthesis handling three geographical resolutions. *ISPRS International Journal of Geo-Information*, 7(5):174, 2018.

Pablo Moscato. An introduction to population approaches for optimization and hierarchical objective functions: A discussion on the role of tabu search. *Annals of Operations Research*, 41:85–121, 1993.

Multi-Agent Transport Simulation (MATSim). `https://matsim.org/`, 2021. Accessed: 2021-05-14.

Nysret Musliu, Andrea Schaerf, and Wolfgang Slany. Local search for shift design. *European Journal of Operational Research*, 153:51–64, 02 2004. doi: 10.1016/S0377-2217(03)00098-5.

Münchner Verkehrsgesellschaft GmbH (2021). `https://www.mvg.de/services/mobile-services/mvg-sod/isartiger.html{#}intro`, 2021. Accessed: 2021-11-17.

A.G. Nikolaev and Sheldon Jacobson. Simulated annealing. *Handbook of Metaheuristics*, 146: 1–39, 09 2010a. doi: 10.1007/978-1-4419-1665-5_1.

Alexander G Nikolaev and Sheldon H Jacobson. Simulated annealing. In *Handbook of metaheuristics*, pages 1–39. Springer, 2010b.

Eric Norgren and Johan Jonasson. Investigating a genetic algorithm-simulated annealing hybrid applied to university course timetabling problem: A comparative study between simulated annealing initialized with genetic algorithm, genetic algorithm and simulated annealing, 2016.

Yaghout Nourani and Bjarne Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373, 1998.

Ibrahim H Osman and Gilbert Laporte. Metaheuristics: A bibliography, 1996.

Moon-Won Park and Yeong-Dae Kim. A systematic procedure for setting parameters in simulated annealing algorithms. *Computers & Operations Research*, 25(3):207–217, 1998.

Alex Kwaku Peprah, Simon Kojo Appiah, Samuel Kwame Amponsah, et al. An optimal cooling schedule using a simulated annealing based approach. *Applied Mathematics*, 8(08):1195, 2017.

Python. `https://www.python.org/`, 2021. Accessed: 2021-12-13.

Monia Rekik, Jean-François Cordeau, and François Soumis. Implicit shift scheduling with multiple breaks and work stretch duration restrictions. *J. Scheduling*, 13:49–75, 02 2010. doi: 10.1007/s10951-009-0114-z.

Caroline Rodier. The Effects of Ride Hailing Services on Travel and Associated Greenhouse Gas Emissions. Institute of Transportation Studies, Working Paper Series qt2rv570tt, Institute of Transportation Studies, UC Davis, April 2018. URL `https://ideas.repec.org/p/cdl/itsdav/qt2rv570tt.html`.

Fabio Romeo and Alberto Sangiovanni-Vincentelli. A theoretical framework for simulated annealing. *Algorithmica*, 6(1):302–345, 1991a.

Fabio Romeo and Alberto Sangiovanni-Vincentelli. A theoretical framework for simulated annealing. *Algorithmica*, 6(1):302–345, 1991b.

Claudio Ruch, ChengQi Lu, Lukas Sieber, and Emilio Frazzoli. Quantifying the efficiency of ride sharing. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

Bruce Schaller. The new automobility: Lyft, uber and the future of american cities, 2018. URL `http://www.schallerconsult.com/rideservices/automobility.pdf`.

Manuel Schmitt and Rolf Wanka. Particle swarm optimization almost surely finds local optima. *Theoretical Computer Science*, 561:57–72, 2015.

Susan Shaheen. Shared mobility: the potential of ridehailing and pooling. In *Three revolutions*, pages 55–76. Springer, 2018.

Susan Shaheen and Adam Cohen. Shared ride services in north america: definitions, impacts, and the future of pooling. *Transport reviews*, 39(4):427–442, 2019.

Susan Shaheen, Adam Cohen, Ismail Zohdy, et al. Shared mobility: current practices and guiding principles. Technical report, United States. Federal Highway Administration, 2016.

Kambiz Shojaee., Hamed Shakouri G., and Mojtaba Behnam Taghadosi. Importance of the initial conditions and the time schedule in the simulated annealing. In Rui Chibante, editor, *Simulated Annealing*, chapter 12. IntechOpen, Rijeka, 2010. doi: $10.5772/\text{intechopen.}83946$. URL `https://doi.org/10.5772/intechopen.83946`.

Marius Sinclair. Comparison of the performance of modern heuristics for combinatorial optimization on real data. *Computers & Operations Research*, 20(7):687–695, 1993.

El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

Craig A Tovey. Hill climbing with multiple local optima. *SIAM Journal on Algebraic Discrete Methods*, 6(3):384–393, 1985.

Uber Pool (2021). `https://www.uber.com/de/en/ride/uberpool/`, 2021. Accessed: 2021-10-24.

Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, 2013. URL `https://EconPapers.repec.org/RePEc:eee:ejores:v:226:y:2013:i:3:p:367-385`.

Kay W Axhausen, Andreas Horni, and Kai Nagel. *The multi-agent transport simulation MATSim*. Ubiquity Press, 2016.

Ingo Wegener. Simulated annealing beats metropolis in combinatorial optimization. In *International Colloquium on Automata, Languages, and Programming*, pages 589–601. Springer, 2005.

Magdalena Widl and Nysret Musliu. An improved memetic algorithm for break scheduling. In María J. Blesa, Christian Blum, Günther Raidl, Andrea Roli, and Michael Sampels, editors, *Hybrid Metaheuristics*, pages 133–147, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-16054-7.

Magdalena Widl and Nysret Musliu. The break scheduling problem: complexity results and practical algorithms. *Memetic Computing*, 6:97–112, 2014.

Gabriel Wilkes, Roman Engelhardt, Lars Briem, Florian Dandl, Peter Vortisch, Klaus Bogenberger, and Martin Kagerbauer. Self-regulating demand and supply equilibrium in joint simulation of travel demand and a ride-pooling service. *Transportation Research Record*, 2675(8):226–239, 2021. URL `https://doi.org/10.1177/0361198121997140`.

Tse-Chiu Wong, Mai Xu, and Kwai-Sang Chin. A two-stage heuristic approach for nurse scheduling problem: A case study in an emergency department. *Computers & Operations Research*, 51:99–110, 2014.

Anthony Wren and David O Wren. A genetic algorithm for public transport driver scheduling. *Computers & Operations Research*, 22(1):101–110, 1995.

Lin Xie, M Merschformann, N Kliewer, and L Suhl. Metaheuristics approach for solving multi-objective crew rostering problem in public transit. Technical report, Working paper. University of Paderborn, Germany, 2013.

Wenwen Zhang, Subhrajit Guhathakurta, Jinqi Fang, and Ge Zhang. Exploring the impact of shared autonomous vehicles on urban parking demand: An agent-based simulation approach. *Sustainable Cities and Society*, 19:34–45, 2015.

Felix Zwick and Kay W Axhausen. Analysis of ridepooling strategies with matsim. In *20th Swiss Transport Research Conference (STRC 2020)(virtual)*. IVT, ETH Zurich, 2020.

Felix Zwick, Nico Kuehnel, Rolf Moeckel, and Kay Axhausen. Ride-pooling efficiency in large, medium-sized and small towns - simulation assessment in the munich metropolitan region. *Procedia Computer Science*, 184:662–667, 01 2021. doi: 10.1016/j.procs.2021.03.083.