TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

Dynamic Vehicle Routing with Time Windows for Police Patrolling

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

066 394 Masterstudium Technische Mathematik

eingereicht von

Michael Langschwert, BSc BSc
Matrikelnummer 00925644

ausgeführt am Institut für Stochastik und Wirtschaftsmathematik
der Fakultät für Mathematik und Geoinformation
der Technischen Universität Wien

Betreuung
Betreuer: DI Dr.techn. Nikolas Popper
Mitwirkung: DI Dr.techn. Martin Bicher

Wien, 21.10.2024 (Unterschrift Verfasser) (Unterschrift Betreuer)

Dedicated to my parents and my wife.
For their everlasting love and support.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources and resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Date, Signature

Abstract

Patrolling with police vehicles is a common strategy for crime prevention. During a patrol, so called points of interest are visited to show police presence and discourage potential criminals. Those are locations like parks, shops, bars and drug hotspots. However, patrols need to be interrupted eventually if police presence is needed more urgently elsewhere.

The influence of interruptions on patrolling vehicles is determined to optimise the assignment of police resources and the execution of patrols. The aim of this thesis is to target this Dynamic Vehicle Routing Problem with time windows. For this purpose, quality criteria for a patrol are defined, Monte Carlo Simulations are performed and the quality criteria are evaluated.

Based on real street data from OpenStreetMap, a network graph is created. Nodes of this network graph are defined as points of interest which should be visited in certain time windows during patrols. The routes are planned using the metaheuristic algorithms "Guided Local Search" and "Simulated Annealing". Patrol interruptions are introduced to simulate the unpredictable occurrence of police operations. Finally, the influence on quality criteria for patrols is evaluated statistically.

Zusammenfassung

Das Patrouillieren mit Polizeifahrzeugen ist eine gängige Strategie zur Verbrechensprävention. Während einer Patrouille werden sogenannte „Points of Interest“ besucht, um Polizeipräsenz zu zeigen und potenzielle Kriminelle abzuschrecken. Das sind Orte, die besondere Aufmerksamkeit der Polizei bedürfen, wie zum Beispiel Parks, Geschäfte, Lokale und Drogen-Hotspots. Die Patrouillen müssen allerdings unterbrochen werden, wenn die Polizei an anderen Orten dringender eingesetzt werden muss.

Der Einfluss von Unterbrechungen auf die patrouillierenden Fahrzeuge wird untersucht, um den Einsatz von Polizei-Ressourcen und die Durchführung von Patrouillen zu optimieren. Ziel dieser Arbeit ist die Behandlung dieses dynamischen Fahrzeugrouten-Problems mit Zeitfenstern. Hierfür werden Qualitätskriterien für eine Patrouille definiert, Monte-Carlo-Simulationen durchgeführt und die Qualitätskriterien abgeschätzt.

Basierend auf realen Straßendaten von OpenStreetMap wird ein Netzwerkgraph erstellt. Knoten dieses Punktenetzes werden als „Points of Interest“ definiert und mit Zeitfenstern versehen, die im Rahmen einer Patrouille besucht werden sollen. Die Routenplanung erfolgt unter Verwendung von den metaheuristischen Algorithmen „Guided Local Search“ und „Simulated Annealing“ . Es werden Patrouillen-Unterbrechungen angenommen, um Polizeieinsätze zu simulieren. Abschließend wird der Einfluss auf die Qualitätskriterien für Patrouillen statistisch ausgewertet.

Contents

List of Figures

List of Tables

1 Introduction

Police work includes many different tasks. One of them is patrolling, which aims to convey a feeling of security, prevent crime and to reduce emergency response times. Patrolling is considered as an important part of police work [1]. Crime is not uniformly distributed, as it is more frequent at certain times and certain locations [2]. A trial in Minneapolis showed that patrolling at crime hotspots could reduce the number of crimes by about 6 % to 13 % [3]. However, measuring crime can be difficult and there can always be unrecorded cases, see for example [4].
As a general rule, [2] points out that opportunities to commit crime should be reduced. Even displaced crimes can be beneficial, as criminals cannot attack their preferred targets. However, patrolling for too long at the same hotspot yields less additional deterring effects [5] and police resources could therefore be used more efficiently elsewhere.

The Police Patrol Routing Problem is a Dynamic Vehicle Routing Problem [6], [7]. Vehicle Routing Problems are most commonly researched for vehicles with capacity constraints [8]. Dynamic Vehicle Routing Problems are researched less frequently [8], especially those related to police patrols [6].
The Vehicle Routing Problem can be seen as a generalisation of the Travelling Salesperson Problem [9]. The Travelling Salesperson Problem, arguably the simplest variant of the Vehicle Routing Problems, is an NP-hard problem [10]. Therefore, a polynomial time algorithm to solve the problem does not exist. The most common solution methods for Vehicle Routing Problems can be categorised in exact methods, heuristic methods and metaheuristic methods, with the latter two being the most common [8]. Heuristic and metaheuristic methods are widely used because they are less computational expensive [8], [11]. This is especially beneficial for dynamic routing, which requires frequent recalculation of the routes.

As analysed by [6], the Police Patrol Routing Problem differs from most other Dynamic Vehicle Routing Problems: In case of an emergency, an instant response is needed. As response times should be minimised [1], the routes are changed immediately, which makes real time location information essential.

To gain insights into the Police Patrol Routing Problem, quality criteria will be defined to quantify the influence of stochastic emergencies on patrolling vehicles.
Real street data will be used to create a network graph and emergencies will be simulated to create interruptions for patrolling vehicles. Points of interest will be defined as potential patrol locations. The routes will be planned using metaheuristics. A Monte Carlo Simulation will be performed to evaluate the defined quality criteria statistically.

The upcoming chapters are structured as follows: In Chapter 2, the Police Patrol Routing Problem is introduced in a mathematical context. Also, characteristics of the problem are described. Solving techniques are discussed in Chapter 3. This chapter subdivides in exact, heuristic and metaheuristic methods. The source of the data, OpenStreetMap, the used software, OSMnx and OR-Tools, and Monte Carlo Simulations are discussed in Chapter 4. The simulation model is explained in Chapter 5. This includes simplifications and assumptions, data preparation, the structure of the actual simulation, model validation and the definition of quality criteria. In Chapter 6, the results of the simulations are discussed and quality criteria are compared for different simulation settings with randomized emergencies. Finally, the main results and an outlook are given in Chapter 7.

2 Dynamic Vehicle Routing Problem for Police Patrolling

The Travelling Salesperson Problem is described in Chapter 2.1, which serves as a basis for routing problems. As generalisation of the Travelling Salesperson Problem, the Vehicle Routing Problem is discussed in Chapter 2.2 and a brief overview of the many variants is given. In Chapter 2.3, the characteristic features of the Police Patrol Routing Problem are discussed, which can be seen as a dynamic version of the Vehicle Routing Problem.

2.1 Travelling Salesperson Problem

The Travelling Salesperson Problem, or Travelling Salesman Problem, can be described as combinatorial optimisation problem, in which a hypothetical salesperson wants to visit each city from a given list. All travel distances between those cities are known. The shortest path to visit all cities once and then return to the starting city needs to be found [9].

A mathematical formulation of the Travelling Salesperson Problem is given by [9] and summarised in the following. Weights w_{ij} are defined as the distance from the i^{th} to the j^{th} node. Each node represents a city. x_{ij} is used as an indicator if the salesperson takes a certain route and is defined as

$$x_{ij} := \begin{cases} 1, & \text{the path from the } i^{\text{th}} \text{ to the } j^{\text{th}} \text{ node is on the route} \\ 0, & \text{otherwise} \end{cases} . \tag{1}$$

The problem can be written as minimisation problem of the total distance of the route:

$$\min_{x_{ij}} \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} w_{ij} x_{ij} \tag{2}$$

The constraints

$$\sum_{i=1,i\neq j}^{n} x_{ij} = 1 \qquad \text{for } j = 1, \dots, n \tag{3}$$

and

$$\sum_{j=1,i\neq j}^{n} x_{ij} = 1 \qquad \text{for } i = 1, \dots, n \tag{4}$$

ensure that the salesperson arrives in each city and leaves each city exactly one time.

The problem can be displayed as a graph, where cities are represented as nodes and roads are displayed as edges, which connect the nodes. An example is visualised in Figure 1. However, when only using the Equations 2, 3 and 4, loops in the route are still possible. These loops are also called subtours or subcycles. One possible solution allowed by the equations above is marked in red in Figure 1.

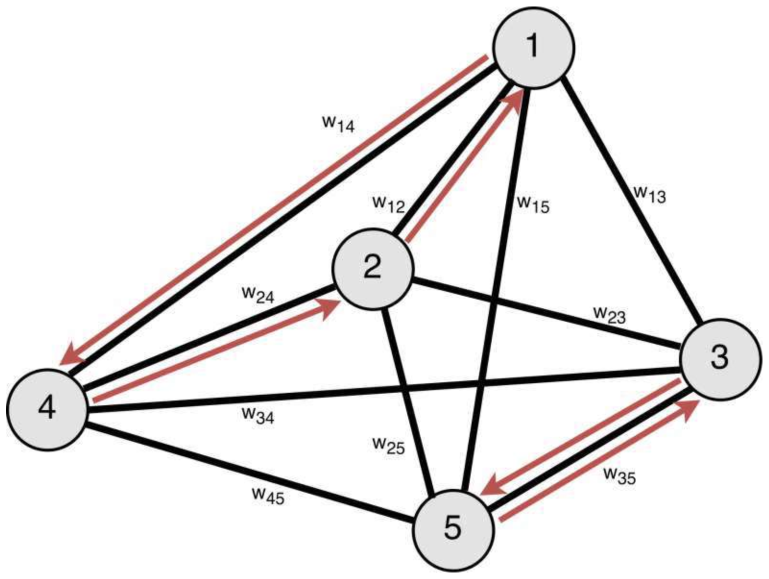


Figure 1: Example network graph with 5 nodes, connecting edges and weights w_{ij}. The routes in red are allowed if Equation 5 is not used.

To solve this issue, additional equations are needed. For this purpose, [9] uses proper subsets of all cities $S \subsetneq \{1, \ldots, n\}$ and defines their associated cardinalities $\widetilde{n} := |S|$. Notably, using a proper subset yields $\widetilde{n} < n$. To ensure that no loops exist, the following equations must be fulfilled:

$$\sum_{i \in S} \sum_{j \in S, i \neq j} x_{ij} \leq \widetilde{n} - 1 \qquad \forall\ S \text{ with } \widetilde{n} \geq 2 \tag{5}$$

For each subset S, all edges which are part of the route and which connect two nodes of the subset are counted. The equations enforce that at least one edge needs to connect with a node outside of the subset.

As mentioned in [9], if there are n cities including the starting city, there are $n-1$ possibilities to choose the second city on the route. To connect all n cities, there are $(n-1)!$ possible routes. If w_{ij} is equal to w_{ji} for all i and j, there are half as many possible solutions $\frac{(n-1)!}{2}$. It is easy to find an arbitrary route that connects all cities. However, it becomes increasingly difficult to find the optimal solution, due to increasing computational cost. The Travelling Salesperson Problem is an example for an NP-hard problem [10]. The class of NP-hard problems contains problems that cannot be solved in polynomial time.

The Travelling Salesperson Problem can be varied by changing the equations above or by adding additional constraints. To give a few examples, the survey [12] classifies the variants as follows:

- Profit oriented: Each visit yields a profit, but not every city needs to be visited.
- Time windows: Cities can only be visited in certain time windows.
- Maximum oriented: To get the maximum possible distance travelled, Equation 2 is maximised instead of minimised.
- Kinetic oriented: Linear movement of the target locations.

2.2 Vehicle Routing Problem

The Travelling Salesperson Problem can be generalised by replacing the traveller with multiple vehicles and by replacing the starting city with a depot. The cities can for example be replaced by customer locations. This generalised problem is commonly referred to as Vehicle Routing Problem. It was first discussed in 1959 under the name "The Truck Dispatching Problem" to find the best routes for delivery trucks [13] and is NP-hard [14]. In Figure 2, possible routes for three trucks, which start and return to a depot, are illustrated.

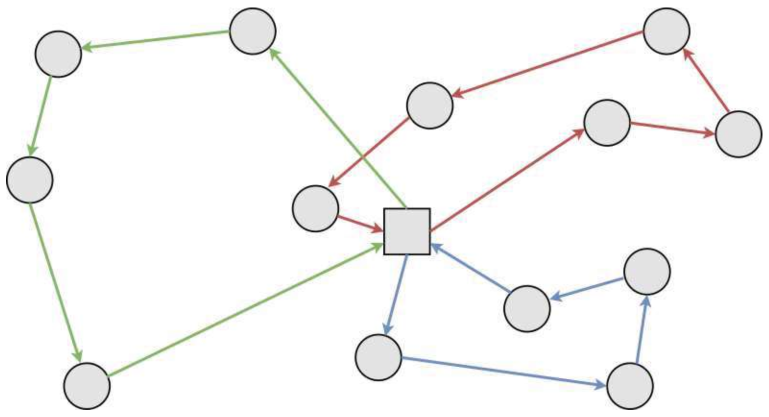


Figure 2: Example routes for three vehicles. Each colour represents the route of a vehicle. All vehicles start from and return to the square, which represents a depot.

Similar to the Travelling Salesperson Problem, there are many different variants of the Vehicle Routing Problem. The classification and review paper [8] analysed 277 articles, including 327 variants from the years 2009 to mid 2015. Out of 16 characteristics, the most popular models included the following:

- Capacitated vehicles 90.52 %
- Time windows 37.92 % (about 30.58 % strict time windows)
- Backhauls 18.65 %

- Heterogeneous vehicles 16.51 %
- Multiple depots 11.01 %

Dynamic requests were only considered in 2.45 % of all Vehicle Routing Problems.

An example for a possible mathematical formulation for a Vehicle Routing Problem with time windows and capacity constraints [15] is given in the following. Variables for this example are defined in Table 1.

C	Set of customer nodes $\{1, \ldots, n\}$
d_i	Demand of the i^{th} customer
N	Set of all nodes $\{0, \ldots, n+1\}$
V	Set of vehicles
q	Vehicle capacity
w_{ij}	Travel cost for travelling from the i^{th} to the j^{th} node
t_{ij}	Travel time for travelling from the i^{th} to the j^{th} node
s_{ik}	Time when vehicle k starts service at the i^{th} customer
$[a_i, b_i]$	Time window in which the service at the i^{th} customer can start

Table 1: Variables for a Vehicle Routing Problem with time windows.

The starting node is labelled as 0 and the final node as $n+1$. Despite having different indices, they can represent the same depot. Time windows $[a_0, b_0]$ and $[a_{n+1}, b_{n+1}]$ for the depot define the delivery time frame of the vehicles. The requirement that each vehicle leaves the starting node and arrives at the final node is expressed in the following formulas:

$$\sum_{j \in N} x_{0jk} = 1 \qquad \forall\, k \in V \tag{6}$$

$$\sum_{i \in N} x_{i,n+1,k} = 1 \qquad \forall\, k \in V \tag{7}$$

The indicator function in Equation 1 from the previous chapter can be adapted by adding an index k, which represents a vehicle:

$$x_{ijk} := \begin{cases} 1, & \text{the path from the } i^{\text{th}} \text{ to the } j^{\text{th}} \text{ node} \\ & \text{is on the route of vehicle } k \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

To calculate the total travel cost of all vehicles, the sum over all vehicles needs to be added to Equation 2:

$$\min \sum_{k\in V} \sum_{i\in N} \sum_{j\in N} w_{ij} x_{ijk}. \tag{9}$$

The following equation ensures that each customer node is visited exactly one time:

$$\sum_{k\in V} \sum_{j\in N} x_{ijk} = 1 \qquad \forall\ i \in C \tag{10}$$

The equation

$$\sum_{i\in N} x_{ihk} - \sum_{j\in N} x_{hjk} = 0 \qquad \forall\ h \in C, \forall\ k \in V \tag{11}$$

counts if a vehicles arrives at a customer node and enforces that it leaves the node again.

The service start time, or arrival time s_{jk}, cannot be earlier than the service start time of the previous customer s_{ik}, which is added to the travel time t_{ij}. If a time duration for the service at the i^{th} customer is needed, it can be included in t_{ij}.

$$x_{ijk}(s_{ik} + t_{ij} - s_{jk}) \leq 0 \qquad \forall\ i, j \in N, \forall\ k \in V \tag{12}$$

Using the inequality in Equation 12 allows vehicle waiting times until the time window of the j^{th} customer opens. The service start time needs to be in the time windows

$$a_i \leq s_{ik} \leq b_i \qquad \forall\ i \in N, \forall\ k \in V \tag{13}$$

Moreover, capacity constraints for each vehicle can be implemented. The demand of all customers of a vehicle cannot exceed the vehicle's capacity.

$$\sum_{i \in C} d_i \sum_{j \in N} x_{ijk} \leq q \qquad \forall\ k \in V \tag{14}$$

2.3 Police Patrol Routing Problem

The Police Patrol Routing Problem is a Dynamic Vehicle Routing Problem [6] in which routes are changed while vehicles are still travelling [7]. For example, a new customer request could require another stop, which is added to an existing route. It is important to reduce response times in police patrolling [1] and, therefore, the routes of police vehicles need to be modified immediately after an emergency occurs. For this process, real time location information of police vehicles is crucial.

Opportunities for committing crimes should be reduced by the police [2]. While patrolling is considered as a fundamental part of police work [1], patrolling policemen cannot work on other tasks at the same time. The balance between showing presence at the most impactful locations and reducing unnecessary travel times is important to save personnel costs.

Most crimes happen in certain time windows [2] and in specific small areas [16] and are not uniformly distributed on a map. These areas are called hotspots and need to be defined as potential patrol locations. To maximise the impact of police patrols, hotspots should be visited in suitable time windows. However, police patrols should be unpredictable [17], which implies that the time windows should not be too strict.

A study by [3] in Minneapolis showed that patrolling at hotspots could reduce crime by 6 % to 13 %. The optimal patrol duration per hotspot was determined to be about 12 minutes. Another study by [5] found that a patrol duration of 14 to 15 minutes is optimal, as additional patrolling yields less benefits.

As mathematical problem description, the formulation from Chapter 2.2 can be used. Equation 14 can be ignored for patrolling police vehicles, as no goods are transported.

3 Solving Techniques

To solve Dynamic Vehicle Routing Problems, routes of vehicles can be recalculated periodically or updated using information from previous calculations [7]. The latter approach results in more complex implementations. This work focuses on the first approach: Vehicle routes are recalculated if data changes or if fixed time intervals pass. Thus, the Dynamic Vehicle Routing Problem is reduced to many Vehicle Routing Problems.
There are various approaches to solve Vehicle Routing Problems. Many of them do not aim to provide an optimal, but a feasible solution. In [8], the most popular solution methods are split into exact, heuristic and metaheuristic methods. The following chapters are each dedicated to one of the above mentioned categories. However, due to the large scope of this topic and the incredible amount of ways to find solutions, the following can only be considered as an excerpt.

3.1 Exact Methods

Vehicle Routing Problems can be solved with exact methods. While exact methods can yield the optimal solution to a problem, they can also take a long time to calculate for non-trivial problems [11].

Brute Force

The most basic approach to solve Vehicle Routing Problems is brute force. Each route is calculated and the best, for example the shortest, is taken. The simplest Vehicle Routing Problem with only one vehicle, the Travelling Salesperson Problem, is already an NP-hard problem [10]. Thus, this approach is limited by problem size and computational power.

There are alternative exact methods to brute force, which attempt to speed up the calculations. However, due to the NP-hardness of the problem, the issue with potential long computation times persists.

The study [11] classifies exact methods as in the following:

- Constraint programming
- Dynamic programming
- Branch and bound/price/cut

These methods can be considered programming paradigms, as they are no simple rules, which guarantee a good result, but rather techniques or patterns, which can be used to approach a problem. Application examples can be found in the study [11].

Constraint Programming

With constraint programming, variable constraints for a problem are defined and the solution space is explored. Commonly, search strategies like backtracking and constraint propagation [18] are used for exploring the solution space. The approach is therefore well suited to situations in which the generation of solutions is of greater importance than the identification of an optimal solution. As constraint programming can include methods from many fields in computer science, it is a very broad topic. Additional information can be found for example in [18] or [19].

Dynamic Programming

Dynamic programming aims to find partial solutions of a problem and expand these. If the problem structure does not provide the opportunities to solve parts of the problem, this approach might not be very efficient. In general, the number of partial solutions which need to be saved to memory should be low for the approach to be efficient [20]. Further information can be found in [20] or [21].

Branch and Bound/Price/Cut

Branch and bound techniques and many of their variants are often used for NP-hard optimisation problems [22]. The techniques can be interpreted as the search in a decision tree, which describes the whole solution space. Possible solutions or partial solutions are enumerated and the search space is reduced by removing parts of the decision tree, which cannot yield better solutions. More information on the topic can be found for example in [22].

For many problems, it is not necessary to find the optimal solution regarding a predefined criterion. Often, it is sufficient to find a feasible solution in a reasonable time. Thus, exact methods are less popular compared to heuristics and metaheuristics [8], which are discussed in the following chapters.

3.2 Heuristic Methods

Heuristic methods can be distinguished into construction and improvement heuristics [23]. While metaheuristic methods are sometimes considered as third category (for example in [24]), they will be discussed separately in Chapter 3.3. This chapter focuses on classic heuristic methods, which are generally considered greedy [23].
Construction heuristics are used to build a solution from scratch. Improvement heuristics are used to improve a solution, which might have been obtained by using a construction heuristic. In many cases, the rules to construct or improve a solution are relatively simple and yield the same result every time they are applied to a problem.

3.2.1 Construction Heuristics

Construction heuristics can be viewed as rules or decisions for finding a solution to a problem [23]. These rules can be very simple and, because of the simplicity of many heuristics, solutions can often be built very fast. For many applications, these solutions can be feasible, but depending on the problem and the chosen heuristics, the found solution might not be optimal. In fact, it might not even be a good solution.

Solutions constructed using heuristic methods are often used as starting solutions for further improvement with other heuristics or metaheuristics [25]. Also, solutions found by constraint programming methods, as discussed in Chapter 3.1, can be used as starting solutions.
A few examples for construction heuristic algorithms are given in the following. More examples can be found for example in [23], [24] and [25].

Nearest Neighbour Addition

The Nearest Neighbour Addition method describes an algorithm for building routes. A brief description for this method is given in [24]:
A route starts at a depot. The route is expanded by appending the closest node if the addition does not violate any constraints. This procedure is repeated until all nodes are added. For problems with more than one vehicle, the routes can be built one after another or simultaneously.
In the first case, nodes are always added to the same route until no more additions are possible. Only then, a second vehicle and, therefore, a second route is introduced. In the second case, the number of routes is predefined and all routes are constructed at the same time. The node which is the closest node to any of the routes is added if the constructed route is feasible.

Insertion Method

Using the previous method, a node is always appended at the end of a route. An insertion method [24] inserts a node to any feasible position of the partially constructed routes. Naturally, by adding more possible positions, the heuristic becomes more computational expensive, but possibly also more successful in satisfying constraints.

Clarke and Wright's Savings Algorithm

The Clarke and Wright's Savings algorithm [26] starts by defining separate routes for each node. Then, the routes are merged to lower the overall cost.

To decide which routes to merge, potential savings S_{ij} for connecting the i^{th} and the j^{th} node are calculated using

$$S_{ij} := w_{i0} + w_{0j} - w_{ij}, \tag{15}$$

with the weights of the edges w. Index 0 marks the starting and end node of all routes. All savings are sorted in descending order and the routes are merged accordingly, starting with the largest saving. Notably, the nodes of a route, which is going to be merged into another, are only inserted next to node 0 and, therefore, the savings only need to be calculated once with Equation 15.

Christofides Algorithm

The Christofides algorithm [27] approaches the problem from a graph theoretical and more technical point of view.
First, the minimum spanning tree of the graph is calculated. The minimum spanning tree is the smallest possible subgraph regarding the total edge cost, which still connects all nodes. In a finite graph, there is always an even number of nodes with odd degree. The degree of a node is defined as the number of connected edges.
In the next step, a perfect matching is found for the set of nodes with odd degree in the minimum spanning tree subgraph. In other words, a subset of edges is found in a way that each of these nodes is connected to exactly one edge. The new edges are added to the previously calculated minimum spanning tree. In this multigraph, two nodes can be connected by multiple edges.
Now, each node has even degree, which guarantees the existence of an Euler Circle, a tour which uses all edges of a graph exactly once and subsequently returns to the starting node. If duplicated nodes are removed from the tour, a Hamiltonian Circle is received. A Hamiltonian Circle is a route which visits all nodes exactly once before returning to the starting node and, therefore, is a constructed solution to the routing problem. [27] provides proofs and additional information.

3.2.2 Improvement Heuristics

An existing solution can be improved by so called improvement heuristics, also known as classical improvement heuristics. For example, solutions can be partly deconstructed and rebuilt again using a construction heuristic [23]. Broadly speaking, improvement heuristics are used for finding the best neighbour solution of the current solution [24]. The neighbourhood $N(S)$ of a solution S is defined as the set of all solutions, which can be reached by changing the solution S according to one or more rules. Rules, also called moves, are changes to a solution and can be combined or restricted to define different neighbourhoods.

For Vehicle Routing Problems with more than one vehicle, improvement heuristics can be distinguished into intra-route and inter-route methods [24], [25]. While intra-route methods only change the route of one vehicle, inter-route methods change multiple routes at the same time.

A few examples are provided in the following. Additional information and examples can be found for example in [24].

Relocating Nodes

For each node of the existing routes, the improvement is calculated if it was relocated to another position. The best found move which still yields a feasible solution is carried out. This can be repeated until no more improving moves can be found.

Swapping of Nodes

Two nodes of the solution are swapped. This improvement heuristic can be viewed as a subset of a heuristic which relocates two nodes at once, however, with the restriction that one node's starting position needs to be the target position of the other.

λ-Opt

The λ-opt improvement method [28] describes the generalised method of replacing edges to gain better solutions. The amount of λ edges are removed from the graph and the potential cost reduction is calculated if λ new edges are added. The edges which yield the best saving are changed.

Additional information on popular improvement heuristics can be found in [24] and [25]. As an alternative to classical improvement heuristics, metaheuristics can be used to improve solutions of the Vehicle Routing Problem.

3.3 Metaheuristic Methods

While many algorithms target a specific problem, metaheuristic methods can be used for a wide range of problems [23]. Metaheuristics are generalised methods or strategies that focus on parameters and not on the function F, which should be optimised. The balance between exploring the solution space and locally improving a solution is a key element of metaheuristic methods [29].

Despite often missing theoretical foundation [29], metaheuristic methods are widely used and many variants exist. Compared to exact methods, they are often less computationally expensive [8], [11]. Because of this and because of their broad range of use cases, metaheuristic methods were the most popular techniques to approximate solutions of Vehicle Routing Problems in 2021 [11].

A metaheuristic algorithm can be applied to a random generated solution or to a solution that was constructed using a heuristic. While there is no guarantee to find an optimal solution, feasible solutions can be found for many complex problems [23]. In the following, a selection of metaheuristic methods is discussed. For each of the methods, many variants exist. More information and historical context can be found in [23].

3.3.1 Local Search

Simple Local Search algorithms are comparable with improvement heuristics in Chapter 3.2.2. Similar to the previous chapter, Local Search algorithms try to improve a given solution S by searching a local neighbourhood $N(S)$ of S [23]. The neighbourhood is defined as the set of all solutions that differ from S by one or more moves. A move can, for example, be a simple permutation, an insertion of a node into an existing route or a swap of nodes. In contrast to classical improvement heuristics, the perspective changes for metaheuristics, as the focus moves from problem specific neighbourhood operators to generalised algorithms.
A very simple Local Search algorithm variant is described in Algorithm 1.

Algorithm 1 Local Search

```
 1: Choose starting solution $S$
 2: $improving \leftarrow True$
 3:
 4: while $improving$ do
 5:     Select the best neighbour solution $S_{new} \in N(S)$
 6:     if $F(S_{new}) < F(S)$ then
 7:         $S \leftarrow S_{new}$
 8:     else
 9:         $improving \leftarrow False$
10:     end if
11: end while
12:
13: return $S$
```

First, a starting solution S is chosen, for example with a heuristic construction method or at random. In each iteration of the algorithm, the neighbourhood of the current best solution S is searched for better performing solutions. In this example, the best neighbour solution S_{new} is selected to be solution S in the next iteration. As an alternative, the first better solution - compared to S - can be taken. These steps are repeated until no better solution can be found that further lowers the cost $F(S_{new})$. The last found solution, which is the best encountered, is returned.

Obviously, Algorithm 1 is a greedy algorithm and might get stuck in a local minimum. A simple attempt to combat this issue is to repeat the procedure multiple times for different starting solutions. [30] proves that S converges against the global minimum, if the number of algorithm runs with uniformly distributed starting solutions converges to infinity.
It is important to keep in mind that Local Search algorithms are searching for better solutions in specific neighbourhoods of the current solution. A found minimum regarding one neighbourhood and its set of allowed moves might not be the global minimum. On the other hand, the global minimum is a minimum in all neighbourhoods [23].
A strategy that utilises a set of different neighbourhoods is the Variable Neighbourhood Search.

Variable Neighbourhood Search

In a Variable Neighbourhood Search [31], multiple different neighbourhoods $N_i(S)$, $0 < i \leq n$, are defined. The pseudocode is presented in Algorithm 2.

Beginning with a starting solution S, a random neighbour of the solution is improved regarding a local neighbourhood, using Local Search. The random choice, also called shaking [29], is added to avoid cycling solutions. If no better solutions are found, the next neighbourhood is tried. If the current solution is improved, the whole procedure starts at the first neighbourhood again. The algorithm can be stopped if the improvement of the solution is insufficient, a certain number of calculation cycles or a certain amount of time has passed.
More information regarding the Variable Neighbourhood Search can be found in [31] and [29].

Algorithm 2 Variable Neighbourhood Search

```
1: Choose starting solution $S$
2:
3: while no stopping criteria is satisfied do
4:     $i \leftarrow 1$
5:     while $i \leq n$ do
6:         Select random solution $S'$ from $N_i(S)$
7:         $S_{new} \leftarrow LocalSearch(S', N_i(S'))$
8:         if $F(S'') < F(S)$ then
9:             $S \leftarrow S''$
10:            $i \leftarrow 1$
11:        else
12:            $i \leftarrow i + 1$
13:        end if
14:    end while
15: end while
16:
17: return $S$
```

3.3.2 Tabu Search

Tabu Search was created by Fred Glover in the 1980s [32] and is one of the most popular metaheuristics [33]. In its basic form, it utilises memory but no randomness. It is based on Local Search, but introduces a tabu list. The purpose of a tabu list is to attempt escaping local extrema and to avoid cyclic choosing of solutions. Depending on the implementation, the tabu list temporarily logs former solutions or moves. A pseudocode for this metaheuristic method is given in Algorithm 3.

Algorithm 3 Tabu Search

```
1:  Choose starting solution S
2:  Initialise empty tabu list $TabuList \leftarrow \{\}$
3:
4:  while no stopping criteria is satisfied do
5:      Select the best neighbour solution $S_{new} \in N(S)$
6:      if $S_{new} \notin TabuList$ then
7:          $S \leftarrow S_{new}$
8:      end if
9:      Update TabuList
10: end while
11:
12: return the best found solution
```

In each iteration, a Tabu Search algorithm chooses the best neighbour solution that is not logged already in the tabu list. In some variants, reusing solutions is punished but not totally forbidden. In these cases, some aspiration criteria can be used to allow the reuse of solution despite being on the tabu list.

Additional information for Tabu Search for Vehicle Routing Problems with time windows can be found in [33].

3.3.3 Guided Local Search

Guided Local Search [34] is another adaptation of the Local Search algorithm, which is usable for many optimisation problems. Guided Local Search uses information to guide the search in the solution space. For this purpose, the method makes use of penalties, which contrasts to the strict tabu list in Tabu Search that entirely forbids solutions.
A set of n problem dependent features f_i, with $1 \leq i \leq n$, needs to be defined. As long as it is possible for solutions to have a feature or to not have it, a feature can be almost anything. An example for a well defined feature is the following: A solution includes a certain edge of the graph. The feature ”starting at the starting node” is not allowed, as all routes need to fulfill this criterion anyways.

An indicator function is defined as

$$I_i(S) := \begin{cases} 1, & \text{if the solution } S \text{ has feature } f_i \\ 0, & \text{otherwise} \end{cases} . \tag{16}$$

For each feature f_i, a penalty parameter p_i is defined. Initially, all p_i will be set to 0. Later, some of them will be incremented by 1 to count the times a features gets punished.
The cost function is adapted to the so called augmented cost function

$$\widetilde{F}(S) := F(S) + \lambda \cdot \sum_{i=1}^{n} I_i(S) \cdot p_i. \tag{17}$$

The parameter λ controls the influence of the second term in Equation 17 on $\widetilde{F}$. If λ is too small, the algorithm loses its ability to escape local minima. If λ is too large, the penalty of a feature outweighs the original cost function F.

To compare the cost c_i of a feature to the penalty parameter p_i, an utility function is defined as

$$u(S, i) := I_i(S) \frac{c_i}{1 + p_i}. \tag{18}$$

Algorithm 4 Guided Local Search

```
1:  Choose starting solution S
2:  Set penalties $p_i = 0 \; \forall i : 1 \leq i \leq n$
3:
4:  while no stopping criteria is satisfied do
5:      $S_{new} \leftarrow LocalSearch(S)$ using the augmented cost function $\widetilde{F}$
6:
7:      for $i = 1, ..., n$ do
8:          Calculate $u(S_{new}, i)$
9:      end for
10:     $\tilde{i} \leftarrow \underset{i, 1 \leq i \leq n}{\arg\max}\, u(S_{new}, i)$
11:     $p_{\tilde{i}} \leftarrow p_{\tilde{i}} + 1$
12: end while
13:
14: return the best found solution
```

The pseudocode for the algorithm is given in Algorithm 4.
In each iteration of the algorithm, the feature with the highest utility gets punished by incrementing p_i, which increases $\widetilde{F}(S)$. Initially, costly features are punished, but with increasing penalty parameter p_i, the utility value $u(S, i)$ decreases and a high cost feature might get punished less often. The chances of another feature being punished increases. More details can be found in [34] and [29].

3.3.4 Simulated Annealing

Simulated Annealing uses a statistical mechanics approach to try to find a good minimum, ideally the global minimum, of an optimisation problem. The optimisation technique is described in [35] and a brief summary is given in the following.

Simulated Annealing is inspired by statistical mechanics. If a system is in an equilibrium with a heat bath at a certain temperature T, the probability to find the system in a certain state i is given by the Boltzmann distribution

$$P_i \propto e^{-\frac{E_i}{kT}}, \tag{19}$$

with the Boltzmann constant k, often written as k_B, and the systems energy E_i.

Defining $\Delta E := E_i - E_j$ and dividing P_i by P_j yields

$$\frac{P_i}{P_j} = e^{-\frac{\Delta E}{kT}}. \tag{20}$$

The quotient of the probability of two states at a certain temperature depends on the difference of energies.

The Simulated Annealing algorithm uses a variation of Equation 20. The Boltzmann constant and the temperature can be summarised as new parameter T, which will still be called temperature, but is measured in the unit of the cost function. The energy E is replaced by a cost function F, which depends on a solution S, for example a set of nodes to describe a route.

The new formula

$$P(\Delta F, T) = e^{-\frac{\Delta F}{T}} \tag{21}$$

is used to compare two solutions. It provides a guideline for accepting new solutions for the Simulated Annealing algorithm: If $\Delta F = F(S_{\text{new}}) - F(S_{\text{old}}) \leq 0$, and, therefore, the new solution is better than the old one, the new solution is taken. For the second case $\Delta F > 0$, [35] uses the new solution if $P(\Delta F, T)$ is larger than a random number between 0 and 1. This approach allows a Simulated Annealing algorithm to sometimes escape local minima by selecting worse solutions. The probability $P(\Delta F, T)$ depends on ΔF and T. If ΔF is large or T is small, the absolute value of the exponent in Equation 21 becomes large. Therefore, the algorithm becomes less likely to accept a worse solution if the temperature T decreases.

A pseudocode for a Simulated Annealing algorithm is given in Algorithm 5. In this example, the temperature is scaled by r at the end of the while-loop. This cooling schedule simulates the system cooling down. However, other cooling schedules can be used as well. In analogy to statistical mechanics, the for-loop is supposed to allow the system to reach an equilibrium with the heat bath.
Additional information can be found in [35]. An example for the Simulated Annealing algorithm for Vehicle Routing Problem with time windows can be found in [36].

Algorithm 5 Simulated Annealing

```
 1: Choose starting solution S
 2: Initialise $T \leftarrow T_0$
 3:
 4: while no stopping criteria is satisfied do
 5:     for i = 1, . . . , n do
 6:         Select a random neighbour $S_{new} \in N(S)$
 7:
 8:         if $F(S_{new}) \leq F(S_{new})$ then
 9:             $S \leftarrow S_{new}$
10:         else if $P(\Delta F, T) > \text{random}(0, 1)$ then
11:             $S \leftarrow S_{new}$
12:         end if
13:     end for
14:
15:     $T \leftarrow r \cdot T$
16: end while
17:
18: return the best found solution
```

3.3.5 Other Metaheuristics

The methods discussed so far are trajectory based metaheuristics. This means, one solution is adapted iteratively [23]. Alternatively, population based methods can be used. In the following, a few examples are given.

Evolutionary Algorithms

Evolutionary Algorithms [37] are inspired by nature. Similar to the theory of evolution, solution candidates are evolved. These methods simulate trial and error and natural selection by using probabilistic parameters.
A popular example for Evolutionary Algorithms is the Genetic Algorithm [37]. In analogy to genes, possible solutions are encoded, often as bit strings. A set of candidate solutions is rated by their fitness, which is their ability to solve a problem. A higher fitness value yields a higher chance to reproduce and pass on their genes. To create a new set of solutions, the reproduction process uses the concepts of mutation, recombination and selection. Probabilities are used in each step of the reproduction process, which allows more diversity in the next generations.
More information on Evolutionary Algorithms can be found in [37].

Swarm Intelligence

Swarm Intelligence algorithms are inspired by animal swarms and flocks. In swarms and flocks, the knowledge of one individual influences the behaviour of others. In the following, two examples are given.
The Ant Colony Optimisation algorithm [38] simulates ants on a network graph. Pheromone values are assigned to each edge. A set of ants walks on the graph and every single one of them builds a solution. Each ant decides its route based on a probability function. The probability depends on the pheromone values of the edges. Also, repeated nodes in their routes are prohibited. This procedure is repeated, and after each iteration, the pheromone values are adjusted.

A set of solutions can be viewed as a Particle Swarm [39]. In each iteration, the location for each particle in the solution space is updated. To update all particle locations, the particles use the knowledge of their current location, their current velocity, their personal best solution and its respective location, as well as the global best solution and its position.
An example for the Travelling Salesperson Problem is given in [40]. In the example, swap operators are defined for the nodes of a solution. These permutations are used to move in the solution space.

Also, hybridisations of solution methods are possible. Additional information can be found in [25]. More information and additional examples for metaheuristics can be found for instance in [29], [24] and [23].

4 Methods and Materials

In this chapter, the source of geographical data, the used software frameworks and the used methods are discussed. OpenStreetMap, a project which provides free map data, is explained in Chapter 4.1. OSMnx, a software to work with OpenStreetMap data and street network graphs, is discussed in Chapter 4.2. A tool to solve the Vehicle Routing Problem, OR-Tools, is discussed in Chapter 4.4. In Chapter 4.5, the Monte Carlo Simulations and stopping rules are explained.

4.1 OpenStreetMap

OpenStreetMap [41] was founded in 2004 and is supported by the OpenStreetMap Foundation [42]. The foundation aims to make geographical data accessible for everyone. To do so, it supports OpenStreetMap as an intermediary for fundraising, provides infrastructure, supports working groups and functions as a legal entity.
OpenStreetMap is organised as a community project. The OpenStreetMap community collects geographical data from all over the world and maintains it. The project has a remarkable user base, with 11 million registered users in September 2023 [43]. Geographical data are provided for free as open data, using the Open Data Commons Open Database License [44].

OpenStreetMap provides road map information, building information, points of interest and associated metadata. In January 2022, the database contained about 7.4 billion nodes [43].
OpenStreetMap data includes road connections, distances and speed limits. Also, amenity data such as information about points of interest are available, for example schools, restaurants, businesses and public transport.

4.2 OSMnx

OSMnx [45] is a free software for working with street networks in Python. The open source module is built on top of GeoPandas [46], Matplotlib [47] and NetworkX [48]. It aims to make working with and analysing street networks easy. Thus, many useful tools are included. For example, map data and maps can easily be downloaded from OpenStreetMap by using city names, bounding boxes, polygons, the distance around an address or coordinates that define the area of the desired map. Downloaded networks are automatically processed to simplify the topology of the network graphs.
OSMnx allows saving and loading several file formats and provides many functions for working with street networks. Also, NetworkX's functions can be used directly. This includes algorithms and functions for network analysis, network manipulation, shortest path algorithms and distances from coordinates to edges or nodes. The used algorithm for finding the optimal path with given weights in an network graph, Dijkstra's Algorithm [49], is also easily accessible through OSMnx and will be explained in Chapter 4.3.
Functions for visualising maps are already implemented in OSMnx and the included Matplotlib package. They include marking nodes, edges and routes, which makes visualisations easy.

4.3 Dijkstra's Algorithm

Dijkstra's Algorithm [49] is used for finding optimal paths in a network graph with non-negative weights. Weights can for example be travel costs, distances or travel times. The basic idea of the algorithm is to expand paths beginning at a start node and to update the distances to other nodes.
A pseudocode is given in Algorithm 6. First, two dictionaries are initialised. One for saving the distances from the start node to any other node, and a second one for saving the previous node in the shortest known path. The queue is used to keep track of nodes that need to be processed. The distance to the start node is set to 0.
In the while-loop, the node with minimum distance is chosen from all nodes in the queue. As the distance is already minimal, no shorter path to this node

can be found. Two cases can occur: In the first case, the node is the end node. The algorithm terminates and returns the distance and predecessor dictionaries. In the second case, the node with minimum distance is not the end node. Paths are expanded by exploring neighbours of the node and the dictionaries are updated.
Alternatively, to find all distances and optimal paths from a given start node, the return-statement can be moved to the end of the algorithm and the if-statement in line twelve can be removed.

Algorithm 6 Dijkstra's Algorithm

```
 1: Searching for the optimal path from a $start$ to an $end$ node in a $graph$
 2:
 3: for $node$ in $graph$ do
 4:     $distance[node] \leftarrow \infty$
 5:     $predecessor[node] \leftarrow None$
 6:     add $node$ to $queue$
 7: end for
 8: $distance[start] \leftarrow 0$
 9:
10: while $queue$ is not empty do
11:     $min\_node \leftarrow node$ in $queue$ with minimum $distance[node]$
12:     if $min\_node$ equals $end$ then
13:         return $distance, predecessor$
14:     end if
15:     Remove $min\_node$ from $queue$
16:
17:     for $neighbour$ of $min\_node$ in $queue$ do
18:         $new\_dist \leftarrow distance[min\_node] + weight(min\_node, neighbour)$
19:         if $new\_dist < distance[neighbour]$ then
20:             $distance[neighbour] \leftarrow new\_dist$
21:             $predecessor[neighbour] \leftarrow min\_node$
22:         end if
23:     end for
24: end while
```

4.4 OR-Tools

Operations Research Tools, or short OR-Tools [50], is developed by Google for combinatorial optimisation. OR-Tools is an open-source software package and distributed under the Apache License 2.0.
The software suite includes solvers for constraint programming, linear programming, packing and Vehicle Routing Problems. Also, wrappers for both open source and commercial solvers are available. Graph algorithms are included for flow problems.
The software is written in C++, but can also be used through wrappers in Python, Java and C#. Additional information can be found in [50].

For vehicle routing, many options are available: The classic Travelling Salesperson Problem, Vehicle Routing Problems with capacity and time window constraints and also deliveries and pickups. It allows optimisation of different target dimensions, for example travel time, travel distance and transported weight. Options for vehicles to wait at a location, which could be needed if time windows are restrictive, and a penalty for missed stops are also available. Calculation time limits and solution limits can be set to control computational effort while searching for a solution.

OR-Tools provides various heuristics to create initial solutions and metaheuristic methods as improvement strategies. The heuristic methods available in OR-Tools include but are not limited to:

- Route expansion methods
- Insertion methods
- Clarke and Wright's Savings Algorithm
- Sweep Algorithm
- Christofides Algorithm

Some of the heuristic methods above were previously explained in Chapter 3.2.1.

Using the OR-Tools software, solutions found with heuristic methods can be improved with following metaheuristics algorithms, which were discussed in Chapter 3.3:

- Greedy Local Search
- Tabu Search
- Guided Local Search
- Simulated Annealing

A complete list of available options can be found in [50].

The metaheuristic improvement methods available in OR-Tools are adapted for Vehicle Routing Problems. Also, parameters are already set to default values that are suitable for many problems.
For example, for the Guided Local Search algorithm, the features are already predefined as edges that a solution might include. In the following, the indexing for the features f_i, the respective penalty parameter p_i and the travel cost per edge c_{ij} is changed, compared to Chapter 3.3.3. To better associate the indices with the nodes and edges, the index i is replaced by two node indices i and j.

The second term of the augmented cost function in Equation 17 is changed for the use in the OR-Tools software. The adapted formula includes the travel cost per edge c_{ij}, as in

$$\widetilde{F}(S) := F(S) + \lambda \cdot \sum_{i,j=1}^{n} I_{ij}(S) \cdot p_{ij} \cdot c_{ij}(S), \tag{22}$$

with a default value $\lambda = 0.1$.

4.5 Monte Carlo Simulations and Stopping Rules

The optimisation of a problem, for example a Vehicle Routing Problem, is usually carried out on static input. Events like emergencies, which are interruptions of patrolling police vehicles, are not predictable and, therefore, cannot be assumed to be static. Probability distributions can be used to model stochastic emergencies, for example with the Gaussian distribution. In Chapter 5.2.2, modelling the emergencies will be explained in more detail. Monte Carlo Simulations [51] are widely used for calculations that include aspects of randomness. One simulation run can be very dependent on random variables. With a computer, the simulation can be easily repeated a lot of times to sample results. These results can be statistically evaluated to gain insights into a problem.
As summarised in [51], Monte Carlo Simulations bring many benefits. They can be applied easily and their use cases are very flexible, as simulation runs can be conducted simultaneously with parallelisation.
Monte Carlo Simulations have many areas of application, which include for example finance, physics, operations research and statistics.

As examined in [52], one of the challenges when using Monte Carlo Simulations is the question: How many simulation runs are sufficient? Too many simulation runs can be a waste of time and resources, while a too small sample size could yield unreliable results. Therefore, a stopping rule can be an useful guideline to help deciding when to stop simulating.

For a random variable X, the expected value $\mu := \mathbb{E}(X)$, the variance $\sigma^2 := \mathbb{V}(X)$ and the empiric mean $\overline{X}_M := \sum_{i=1}^{M} X_i$ are defined. X_i, $i \in \mathbb{N}^+$, are considered to be independent and identically distributed random variables.
A stopping rule can be written as a function

$$f : \mathbb{R}^+ \times (0,1) \times \mathbb{N} \to \mathbb{R} : (\delta_{abs}, p, M) \mapsto f(\delta_{abs}, p, M, \cdot), \tag{23}$$

for which the implication

$$f(\delta_{abs}, p, M, \cdot) \geq 0 \Rightarrow P(\mu \in [\overline{X}_M - \delta_{abs}, \overline{X}_M + \delta_{abs}]) \geq p \tag{24}$$

holds. In other words, if $f(\delta_{abs}, p, M, \cdot)$ is 0 or positive, the probability to find the expected value in the confidence interval defined by δ_{abs} is greater than the confidence level p.
The stopping index M_{stop} is defined as the smallest $M \in \mathbb{N}^+$ with $f(\delta_{abs}, p, M, \cdot) \geq 0$. M_{stop} can depend on other parameters.

Two rules are discussed in [52]: The Chebyshev stopping rule and the Gaussian stopping rule. Both require bounded first and second moments for the random variable X. The Chebyshev stopping rule is defined as

$$f(\delta_{abs}, p, M, \sigma^2) = (1-p) - \frac{\sigma^2}{M \cdot \delta_{abs}^2} \tag{25}$$

and the second stopping rule, the Gaussian stopping rule, is defined as

$$f(\delta_{abs}, p, M, \sigma^2) = (1-p) - 2 \cdot \Phi\left(-\frac{\sqrt{M}\delta_{abs}}{\sqrt{\sigma^2}}\right), \tag{26}$$

with the standard normal distribution Φ and a sufficiently large M. While p and δ_{abs} can be chosen, the variance σ^2 is still unknown. In [52], two algorithms are discussed for finding M_{stop}. For one of them, a pseudocode is given in Algorithm 7.

In the algorithm, the sample variance is used as an estimation for the real variance and updated periodically. For this purpose, the sample variance is transformed as follows:

$$s_N^2 := \frac{1}{N-1} \cdot \sum_{i=1}^{N} (X_i - \overline{X}_N)^2 = \frac{1}{N-1} \cdot \sum_{i=1}^{N} (X_i^2 - 2X_i\overline{X}_N + \overline{X}_N^2) \tag{27}$$

$$\overset{*}{=} \frac{1}{N-1}(N\overline{X^2}_N - 2N\overline{X}_N^2 + N\overline{X}_N^2) = \frac{N}{N-1} \cdot (\overline{X^2}_N - \overline{X}_N^2) \tag{28}$$

At the marked equal sign, the formula

$$\overline{X^2}_N = \frac{1}{N} \cdot \sum_{i=1}^{N} X_i^2 \tag{29}$$

is used.

To update the sample variance, the sample means $\overline{X}_i$ and $\overline{X^2}_i$ are updated periodically with the formulas

$$\overline{X}_i = \frac{i-1}{i} \cdot \overline{X}_{i-1} + \frac{1}{i} \cdot X_i \tag{30}$$

and

$$\overline{X^2}_i = \frac{i-1}{i} \cdot \overline{X^2}_{i-1} + \frac{1}{i} \cdot X_i^2. \tag{31}$$

As mentioned in [52], the estimation for the sample variance bears the risk of underestimating M_{stop}. Additional information and derivations can be found in [52].

Algorithm 7 Stopping Rule Algorithm

```
 1: Choose a confidence level $p \in (0,1)$ and a confidence interval $\delta_{abs} > 0$
 2: Choose $M_0$ large enough to avoid early termination of the algorithm
 3:
 4: $\overline{X}_0 \leftarrow 0$
 5: $\overline{X^2}_0 \leftarrow 0$
 6: for $i \in N$ do
 7:     $X_i \leftarrow$ Simulation()
 8:     $\overline{X}_i \leftarrow \frac{i-1}{i} \cdot \overline{X}_{i-1} + \frac{1}{i} \cdot X_i$
 9:     $\overline{X^2}_i \leftarrow \frac{i-1}{i} \cdot \overline{X^2}_{i-1} + \frac{1}{i} \cdot X_i^2$
10:
11:     if $i > M_0$ then
12:         $s_i^2 \leftarrow \frac{i}{i-1} \cdot (\overline{X^2}_i - \overline{X}_i^2)$
13:         if $f(\delta_{abs}, p, i, s_i^2) \geq 0$ then
14:             $M_{stop} \leftarrow i$
15:             return $M_{stop}, \overline{X}_i$
16:         end if
17:     end if
18: end for
```

5 Experimental Chapters

The process of preparing data, the implementation of the simulation and the optimisations are lined out in Chapter 5.1 and Chapter 5.2. The tests for validating the model are explained in Chapter 5.3. In Chapter 5.4.2, quality criteria are introduced and their calculations are discussed. The code written for this thesis is available on GitHub [53].

5.1 Data Preparation

The process of how map data for the simulation is retrieved and prepared is explained in Chapter 5.1.1. Points of interests are defined in Chapter 5.1.2. Both are based on data from OpenStreetMap, which was discussed in Chapter 4.1.

5.1.1 Map Data

For this simulation, the map of the district Favoriten in Vienna is used. Out of all of Vienna's 23 districts, Favoriten is the district with the highest population, with about 212 000 residents in the year 2022 [54].
Compared to the other districts in Vienna, Favoriten has the highest absolute number of registered crimes [55]. In 2021, about 17 500 crimes were registered, which corresponds to about 12.2 % of all registered crimes in Vienna. The average number of registered crimes per 100 000 residents was 7529 in Vienna in the year 2021. In Favoriten, the crime rate was 8424, which was higher than Vienna's average.
Regarding the structure of Favoriten, the north can be described as densely populated, while the south can be characterised as a wider and greener area [54]. This structural difference is visible on the map, which is visualised in Figure 3.

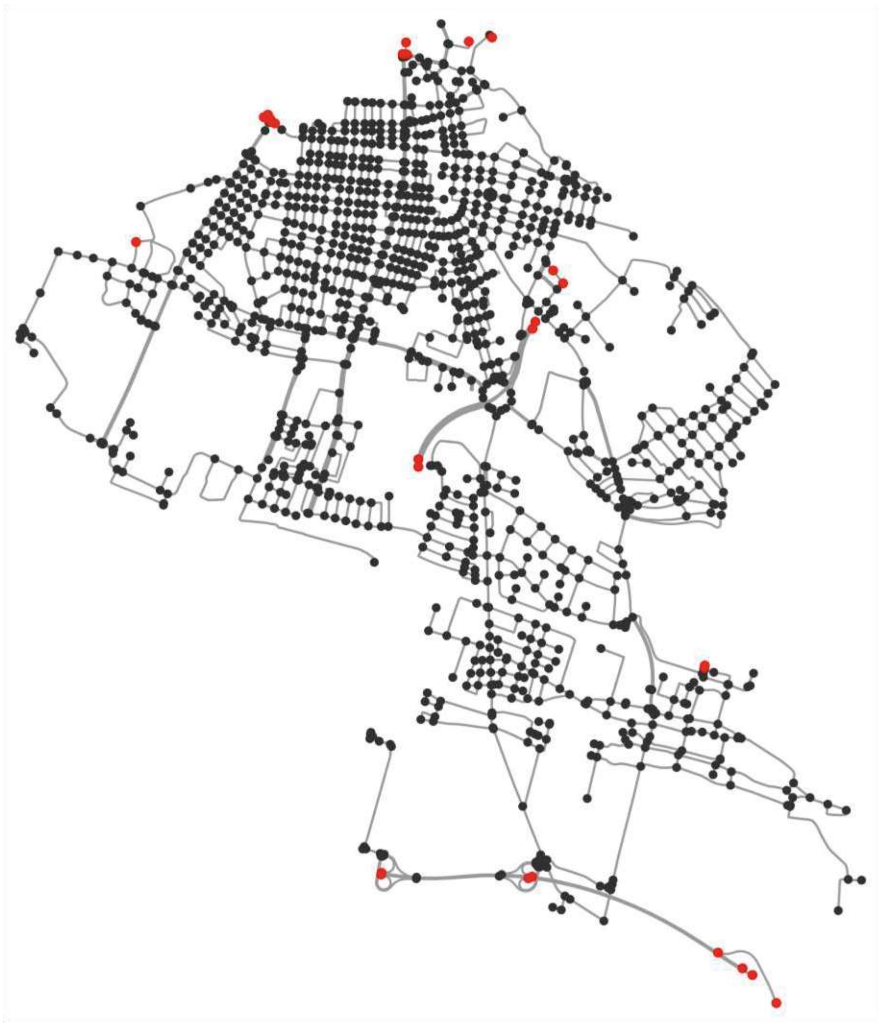


Figure 3: Network graph of the district Favoriten in Vienna, with removed nodes marked in red. (Map data © OpenStreetMap)

Real street data are retrieved from OpenStreetMap using the OSMnx Python module and saved as an OSM file. Coordinates and distances for each edge in the street network are already available, provided by OpenStreetMap. In most cases, speed limits are given as single values. For some edges in the network graph they are missing. This is quite common for very short edges, which for example occur in roundabouts. In these instances, speed limits of $30\,\mathrm{km\,h^{-1}}$ are assumed. If more than one speed limit is given, for example if the speed limit changes somewhere on the edge, the lowest is used. In the street map data of the Viennese district Favoriten, speed limits were given as single values 2638 times, 17 times as multiple values and 131 times no values were assigned.

As approximation for the travel speed, the speed limit is used. The travel time t in seconds for each edge is calculated using the travel speed v in $\mathrm{km\,h^{-1}}$ and the travel distance d in meters with the formula

$$t = d/(v/3.6)\ . \tag{32}$$

Travel times need to be rounded to integers to satisfy the OR-Tools' requirements and are added to the network graph. The average edge length is about 117.5 m and the average travel time is about 12.1 s.

The travel time matrix T consists of elements t_{ij} which are equal to the time needed to travel from the i^{th} node to the j^{th} node. In analogy, the distance matrix D is made up of distances d_{ij}. To obtain both matrices, the fastest route from the i^{th} node to the j^{th} node is calculated with Dijkstra's Algorithm, as explained in Chapter 4.3. The algorithm is already implemented in NetworkX and can be used through OSMnx. Travel times are used as weights. Using the fastest routes, travel times and travel distances are calculated by summing up the respective values for the edges on the route. These values are saved in the matrices T and D. Due to one-way streets, it is possible that $t_{ij} \neq t_{ji}$ and $d_{ij} \neq d_{ji}$, meaning that the matrices are asymmetrical.

In some cases, it is not possible to find a route. This is because of one-way streets on the fringe of the map area. To guarantee for every node to be reachable from any other node, some nodes are removed. Out of the original 1272 nodes in the street map of the district Favoriten, 28 are removed, which are about 2.2 % of total nodes. The removed nodes are visualised in Figure 3. The final dimensions of both matrices for the Favoriten map are 1244×1244. The travel time matrix T, the distance matrix D and mappings of the indices, linking the OpenStreetMap data and the matrices, are saved.

5.1.2 Points of Interest

To identify points of interest, amenity data are downloaded from OpenStreetMap and saved as a CSV file for further processing. Points of interest can be used to define police patrol locations. The full list of all amenity tags in the map of Vienna's district Favoriten and their associated frequencies are displayed in Appendix A in the Tables 10 and 11. Amenity tags, which are associated with a higher fluctuation of people or more nightlife, are chosen as potential points of interest and are displayed in Table 2.

Amenity Tag	Frequency
restaurant	151
fast_food	113
cafe	97
place_of_worship	41
toilets	30
atm	28
bank	27
pub	20
community_centre	12
social_facility	12
bar	10

Amenity Tag	Frequency
gambling	9
brothel	6
cinema	2
theatre	2
events_venue	2
marketplace	2
biergarten	2
casino	1
food_court	1
money_transfer	1
internet_cafe	1

Table 2: Used amenity tags of the Favoriten map with their associated frequencies.

In general, the found amenity coordinates are not located directly on nodes of the street network graph. Also, the amenity coordinates are not always given as single points, but also as coordinates of vertices of line segments, polygons or multiple polygons. As an approximation for the location of the amenity object, average longitude and latitude values of the vertices are calculated. An example for an amenity object with multiple polygons is the layout of the Church of St. Francis de Sales, which has an inner yard and a separated church tower. The church's layout and the average coordinates are visualised in Figure 4.

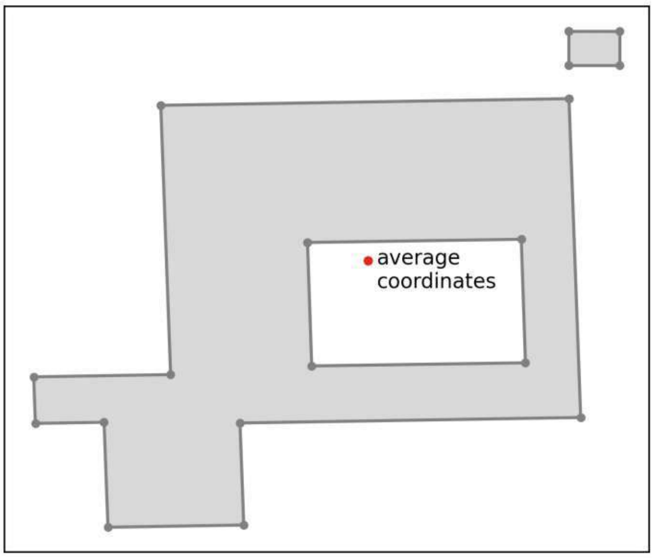


Figure 4: Example for multiple polygons: Church of St. Francis de Sales in the district Favoriten in Vienna with inner yard and separated church tower. The average coordinates are marked in red. (Map data © OpenStreetMap)

OSMnx is used to calculate the closest network nodes for all average amenity coordinates. In the following chapters, network nodes are used as mapping for amenities. All average amenity coordinates and the resulting 314 network nodes with mapped amenities are shown in Figure 5. Out of these network nodes, a fixed random sample of 30 distinct network nodes is chosen as patrol locations.

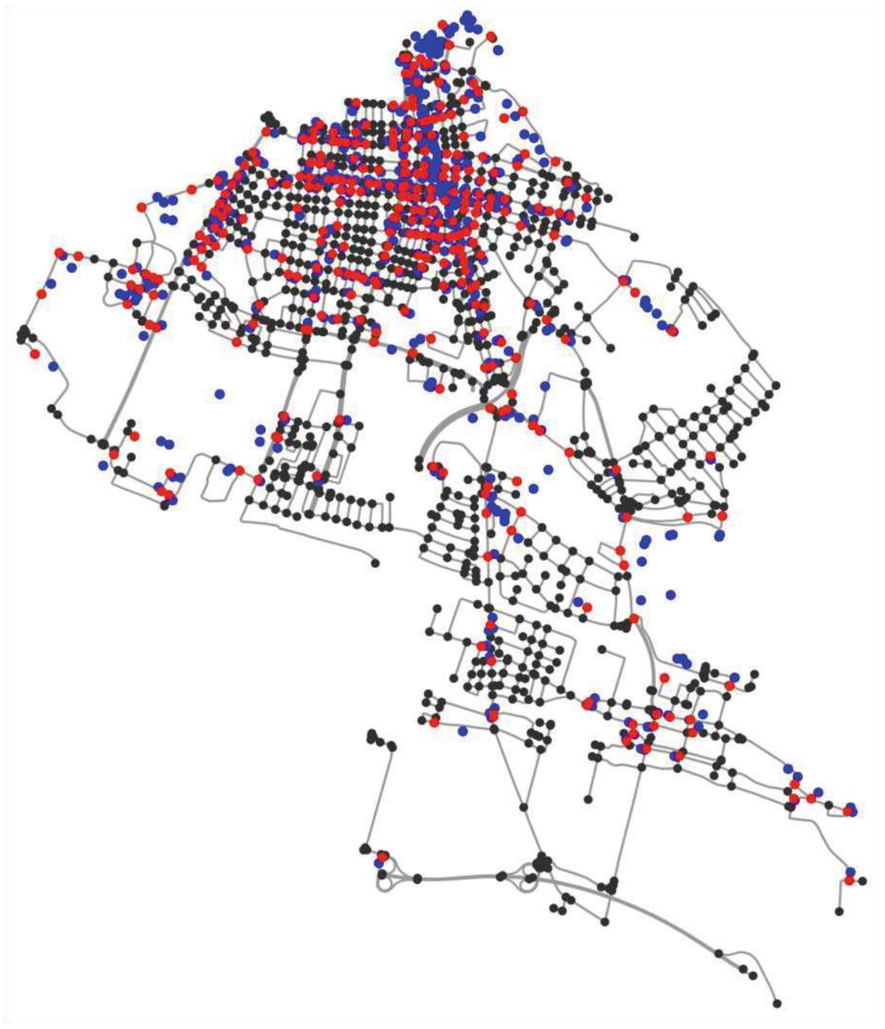


Figure 5: Network graph of the district Favoriten in Vienna with relevant amenity locations in blue and mapped network nodes marked in red. (Map data © OpenStreetMap)

There are seven police stations in the district Favoriten. Their locations on the map can be found in Appendix A in Figure 24. To simplify the simulation, the police station located at Van-der-Nüll-Gasse 11 is chosen as the only start and end location for all patrolling police vehicles in the district.

5.2 Simulation Model

The simulation can be understood as agent-based model [56]. Decisions are made using simple rules. The sum of all decisions results in the state of the simulation. A centralised approach with a mediator agent is used, as described in [57]. The mediator agent coordinates all police vehicles.
In the following chapters, the central parts of the simulation are lined out. Restrictions and assumptions are discussed in Chapter 5.2.1. In Chapter 5.2.2, the calculation of emergency data is explained. The importance and usage of time windows and dummy nodes is discussed in Chapter 5.2.3. In Chapter 5.2.4, the utilisation of OR-Tools is addressed. An overview over the simulation algorithm is given in Chapter 5.2.5. Finally, in Chapter 5.2.6, the acquired data are discussed. The full code written for the simulation can be found on GitHub [53].

5.2.1 Restrictions and Assumptions

The restrictions and assumptions of this simulation can be split in four categories: police vehicles, patrols, emergencies and travel.

Police Vehicles

For simplicity, actual police jurisdictions are ignored and only one police station is used for the whole district Favoriten. It is assumed that no vehicles or teams of policemen or policewomen are assigned to active emergencies at the start of the simulation. Thus, they are available to patrol. All vehicles start at the police station and have to return to it at the end of the simulation.

Patrols

The chosen patrol locations are distinct and the police station cannot be a patrol location. The patrol duration is assumed to be constant for all locations. If a patrol location is visited for the defined patrolling time, it is marked as visited. It is also marked as visited if a police vehicle is called to an emergency at this very location or if the location is visited for a patrol and, because of an emergency, the vehicle needs to leave. Just passing the node is not enough. If the patrol location is not visited in the planned time window, it will be saved as missed location.

Emergencies

As a simplification, emergencies only occur at network nodes. Network nodes are used as representative of their proximity. Notably, an emergency can also happen at the node of the police station. The emergency duration is assumed to be independent from the arrival time. Each emergency is assumed to only require one police vehicle, which resumes patrolling immediately after. Out of all vehicles which are currently not assigned to an emergency, the vehicle with the fastest response time is sent to the emergency location. If no patrolling vehicles are available, the emergency is saved for later analyses. This data could be used for taking extra vehicles from other police stations or extra teams into account, which are not assigned to patrol duty.

Travel

As discussed in in Chapter 5.1.1, the travel speed for each edge is assumed to be the speed limit. The travel time matrix is assumed to be constant. Changes in the travel speed caused by accidents, traffic and time to accelerate or break are not taken into account. Notably, the travel speed to an emergency location is also the same as the values for the edges of the network graph. It is also assumed that the police vehicles always take the fastest route, which is calculated with Dijkstra's Algorithm. Vehicles are only allowed to change their direction at nodes. This assumption is practical for dealing with one-way streets, but also realistic in a dense city setting, where space to turn the vehicle could be sparse.

5.2.2 Emergency Data

As detailed historical emergency data are not accessible to the public, assumptions need to be made for emergencies in the simulations. To create an emergency, a start time, an emergency duration and a location are determined. As location, a node is chosen out of all network nodes with equal probability. The node functions as representative for the proximity of the node itself. The start time of emergencies is assumed to be uniformly distributed in the simulation time span. The emergency duration is calculated with a Gaussian distribution. If the obtained emergency duration is less than a defined minimum duration threshold, it is recalculated. The total number of emergencies is calculated in an analog procedure, it is recalculated if less than 0 is returned. Distribution parameters for emergencies are set in the simulation configuration. Example probability density functions of the total number of emergencies are shown in Figure 6.

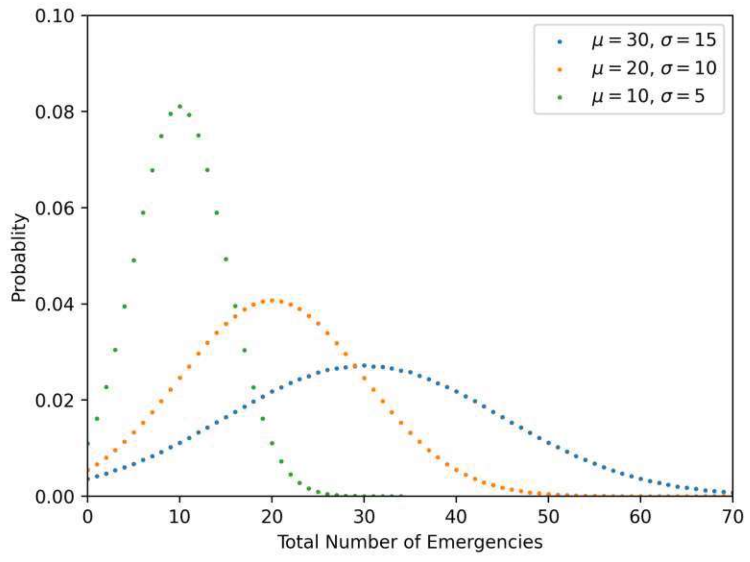


Figure 6: Example probability density functions for the total number of emergencies with different means μ and standard deviations σ.

Table 3 shows an example emergency data set with time values in seconds and the emergency location given as an OpenStreetMap node ID. The values displayed as "None" in the table are calculated and added when a police vehicle is assigned to the emergency.

start_time	2034
duration	1189
arrival_time	None
end_time	None
assigned_vehicle_id	None
location	84285756

Table 3: Example emergency data set.

5.2.3 Time Windows and Dummy Nodes

All patrol locations, the police station and the current vehicle locations need time windows. As the simulation time progresses, the time windows are adapted gradually. The police station is an exception because it always needs to be available for returning police vehicles. Therefore, the time window $[0, 86400]$, which is equivalent to a full day, is used and never updated. Each patrol location has its own time window, within which a police vehicle can patrol. For example, at the simulation time of 100 seconds, the time window $[50, 200]$ is changed to $[0, 100]$. Time windows are also important to enforce correct vehicle behaviour. In case of an emergency or a vehicle returning from an emergency, the routes for all vehicles are calculated again. In general, the position of patrolling vehicles could be anywhere on the network graph when recalculating the routes, as the vehicles are in motion and not necessarily on nodes. In this case, the upcoming network node on the vehicle route is assigned as the new start point for the route calculation. To avoid teleportation of the vehicles, the remaining travel time to the next node on the route is used for both values in the time windows, for example $[20, 20]$ if the vehicle arrives in 20 seconds at the new starting node. In case of patrolling vehicles, the time already spent for patrolling is subtracted from the total patrol time per location, for example $[260, 260]$ if the vehicle has already patrolled 40 seconds out of total planned 300 seconds patrolling time.

Time windows determine when a vehicle leaves a node. Therefore, with two or more vehicles at the same location, additional time windows are needed. Dummy nodes are introduced to tackle this problem. A dummy node is an exact copy of the original node, but has a different time window. The travel time from a copied node to the original node is 0 seconds.

5.2.4 Route Calculation

Patrol routes are calculated using OR-Tools [50]. The total travel time for all vehicles is used as optimisation target for the calculated routes. To be consistent with the defined quality criteria in Chapter 5.4.1, waiting times at locations other than the police station are not allowed. A penalty of 86 400 seconds is defined for missed patrol locations, which is equivalent to one day. The patrolling time per location is set to specify the time spent at patrol locations.
A first solution strategy and a local search metaheuristic need to be chosen in the simulation configuration file. Available methods were already discussed in Chapter 4.4. Also, solution and time limits need to be given as termination criteria for the local search algorithm.
Every time new routes are calculated, following data need to be prepared:

- Number of available police vehicles
- Location of the police station
- Start and end nodes for each police vehicle
- Travel time matrix for all used locations, including dummy locations
- Time windows for all used locations, including dummy locations

In the simulation, the calculation of new patrolling routes needs to be performed up to twice the number of emergencies: Patrolling routes are calculated each time an emergency occurs and a police vehicle drives to the emergency location, or an emergency ends and a police vehicle becomes available for patrolling again.

5.2.5 Simulation Algorithm

In the configuration file, the police station, the number of police vehicles, their starting locations and the patrol locations with their related time windows are defined. Emergency and route calculation configurations were already discussed in Chapter 5.2.2 and 5.2.4.

In Figure 7, the flow of a single simulation run is visualised. At the start of the simulation, if not specified otherwise, the current time in nanoseconds is used as seed and variables are initialised. The simulation time is set to 0. All emergencies are created and copied to the event queue. The simulation loops over all elements in the event queue.
At the start of each loop, the events in the event queue are sorted by occurrence. Using the current simulation time, the list of yet to be visited patrol locations and their corresponding time windows are updated. If vehicles are available, dummy locations and all time windows are adapted as described in Chapter 5.2.3, and routes are calculated as described in Chapter 5.2.4. If there are no events in the event queue and, therefore, the last event was the end of an emergency, the results are saved and the simulation is terminated. If there are more events left, the next one is taken and removed from the queue. The current simulation time is set to the event time and all current vehicle positions are calculated. Moreover, the list for already visited patrol locations and the list for locations which have not yet been visited are updated. If the event is the start of an emergency, the vehicle with the fastest response time is chosen and removed from the list of patrolling vehicles. The police vehicle's route is adapted and a return event is added to the event queue. If the event is the end of an emergency, the vehicle is marked as available again. The simulation continues with the next iteration of the loop.

As each simulation run is independent, multiprocessing is implemented to speed up the time it takes to run multiple simulations. The total number of simulation runs and the amount of parallel processes are set in the configuration file. Also, a list with simulation seeds can be used to run the simulation with the same emergencies but different configurations.

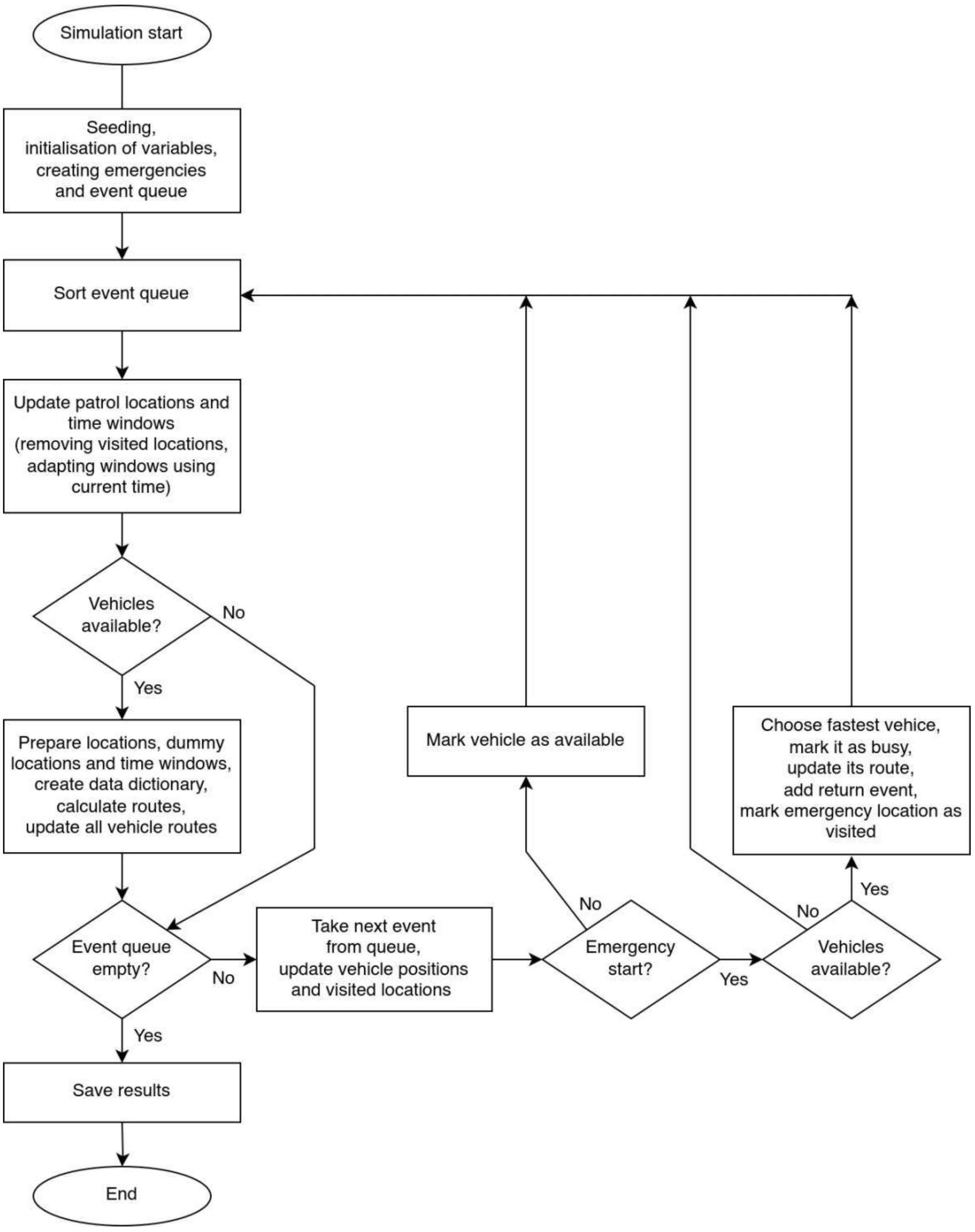

Simulation start
Seeding, initialisation of variables, creating emergencies and event queue
Sort event queue
Update patrol locations and time windows (removing visited locations, adapting windows using current time)
Vehicles available?
No
Yes
Prepare locations, dummy locations and time windows, create data dictionary, calculate routes, update all vehicle routes
Event queue empty?
No
Yes
Take next event from queue, update vehicle positions and visited locations
Emergency start?
No
Yes
Mark vehicle as available
Vehicles available?
No
Yes
Choose fastest vehice, mark it as busy, update its route, add return event, mark emergency location as visited
Save results
End

Figure 7: Flowchart of a single simulation run.

5.2.6 Collected Data

For each simulation run, a JSON file is saved when the simulation terminates. JSON files are human-readable and can be processed for further analyses.

The following generated data is collected:

- Calculation time
- Vehicle information: ID of each attended emergency, old routes and the final route
- All visited locations, including emergency locations
- Emergencies, including time stamps, location and assigned vehicle ID
- Missed emergencies, including start time and location

Moreover, configuration data are saved to some extent. This is helpful for running plausibility tests which are discussed in Chapter 5.3.2, and to carry out further analyses. Also, saved configuration data, especially the random seed, ensure reproducibility of simulation results. Additional saved data includes the following configuration information:

- Simulation seed
- Route calculation parameters: first solution strategy, local search meta-heuristic, solution and time limits
- Location of the police station
- Patrol locations
- Patrolling time per patrol location

5.3 Model Validation

To validate the model, two strategies are implemented. The first strategy is unit tests for isolated functions which are discussed in Chapter 5.3.1. The second strategy is about plausibility tests used to verify if the output is in line with the assumptions and expectations, which is further explained in Chapter 5.3.2. The full code for the model validation is available on GitHub [53].

5.3.1 Unit Tests

Unit tests for isolated functions are conducted to track down bugs. They allow to narrow down the search area for unwanted behaviour of the simulation. This is especially useful if bugs occur after changes in the code were made.

The practicability of simulation configuration data is validated and tests are implemented for the following simulation steps:

- Creation of emergency data
- Creation of dummy locations and updated time windows
- Data preparation for the route calculation
- Updating the patrol location: removing visited locations and updating time windows
- Updating the visited locations list
- Choosing the correct response vehicle
- Updating routes for each vehicle
- Updating the vehicle information, including positions, time to reach the next location and time spent at the current location

Unit tests are only as good as their test cases. Unfortunately, it is easy to miss edge cases. To tackle this issue, complementary to the unit tests, plausibility tests are conducted, which are discussed in the next chapter.

5.3.2 Plausibility Tests

The main goal of plausibility tests is to see if the output of the simulation is valid. The approach can be simpler to implement than unit tests, where the test case needs to be considered first. Furthermore, a huge number of saved simulation data can be validated with plausibility tests. However, the downside of this approach is that the origin of the bug cannot be narrowed down. A wrong time window could result in wrong travel times, prolonged or shortened stays at different locations and the absurdity of a vehicle leaving a location before it has not even arrived. A bug related to dummy locations could result in teleportation of vehicles.

To reduce the number of bugs, the following criteria are tested:

- The start time of an emergency is less than or equal the arrival time of the police vehicle
- The end time of an emergency is equal to the emergency duration added to the arrival time of the police vehicle
- Missed emergencies are tracked correctly
- Visited locations are tracked correctly
- The end node of each route is the police station
- The arrival time at a location is always less than or equal the departure time
- All travel times are consistent with the map data
- Stays are only allowed at the following locations: police station, emergency locations and patrol locations

- The time spent at an emergency location is consistent with the emergency duration
- Patrolling times are consistent with the defined patrolling time in the simulation configuration, except when the vehicle is called to an emergency while patrolling

5.4 Quality Criteria

As simulation targets or key performance indicators for police patrols, quality criteria are defined. Quality criteria are defined in Chapter 5.4.1 and the calculation is explained in Chapter 5.4.2.

5.4.1 Definition of Quality Criteria

Quality criteria aim to measure the quality of a patrol and the impact of changes in the simulation settings. Some simulation settings like patrol locations or the number of patrolling vehicles can be changed by someone planning a patrol, while settings like the map or emergencies are not controllable. However, settings that cannot be influenced may also be interesting to explore, for example to understand the impact of more frequent emergencies.

To express different interests, the quality criteria can be split in three categories: emergencies, patrols and resources.

Emergencies

The **number of emergencies** cannot be influenced and needs to be monitored. In the simulations, the number of emergencies is the only randomised parameter and influences all other quality criteria.
Ideally, no emergency should be unanswered and, therefore, emergencies always have priority. The **number of missed emergencies** or, alternatively, the number of additional vehicles needed can be used as quality criterion. If no patrolling vehicle is available, another team needs to fill in. This means that other tasks like office work need to be interrupted.

Alternatively, another team needs to be requested from another police station. This could potentially lead to increased response times.
The **response time** is also an important factor for police patrols, and should be as low as possible [1]. In a real world setting, the response time could make an important difference, for example, if a suspect is getting caught or fleeing a crime scene. Potentially, a low response time could even save lives.

Patrols

Patrolling reduces the number of crimes [3]. Thus, it is important to show presence and deter potential criminals [1]. Naturally, as many of the defined patrol locations as possible should be visited and the **number of visited patrol locations** should be high. Depending on the number of emergencies and their locations, it is possible that not every patrol location can be visited. To keep route planning flexible and increase the chances of every patrol location to be visited, time windows for patrol locations should be as large as possible.
Also, as a quality measure of the shown presence, the **map coverage** of the patrolling vehicles can be used.

Resources

Additional benefits of patrolling and showing presence decrease over time [5]. Also, the investment of a police vehicle with an assigned team of policemen yields cost. To measure cost and workload, the **total travelled time** and the **total travelled distance** can be used.
While a lower number of emergencies cannot be forced, lowering the time investment and the travelled times could free personnel resources for other tasks. From a resource point of view, patrolling should be finished as soon as possible.

Naturally, there is no solution that optimises all the categories at once, and, obviously, due to different interests, some of the quality criteria are contradictory.

5.4.2 Calculation of Quality Criteria

To calculate the defined quality criteria, the data collected in the simulation are used. Which data are collected was explained previously in Chapter 5.2.6.

Emergencies

The **number of emergencies** is calculated by counting the emergencies from the emergency list, given by the simulation.
The **number of missed emergencies** can easily be calculated by counting the elements of the missed emergency list, returned by the simulation.
For the **response time**, two cases need to be differentiated: attended and missed emergencies.
The response times for attended emergencies can easily be calculated using the saved time stamps in the output file with

$$\text{response\_time} = \text{arrival\_time} - \text{start\_time}. \tag{33}$$

The format of the saved emergency data was shown in Table 3 in Chapter 5.2.2.
However, due to potentially missed emergencies, the list of response times is not always complete. In this case, another vehicle with another team has to step in and deal with the emergency. The new vehicle might not be ready to leave or might be at another location, maybe even another district. To take those response times into account, a penalty value is used for missed emergencies.

The penalty value is defined as

$$\max_{s,e} \min_{x(s,e)} \sum_{i,j} t_{ij} \cdot x_{ij}, \tag{34}$$

with $x(s,e)$ being a route from a starting node s to an end node e. x is defined as a set of indicators x_{ij} which equal 1 if the path from the i^{th} to the j^{th} node is part of the route. The indicator function was defined previously in Equation 1 in Chapter 2.1. t_{ij} are the travel times from the i^{th} to the j^{th} node. Equation 34 yields the longest time a vehicle can travel between any

two nodes of the graph, while still taking the fastest route.
Using the map of the district Favoriten in Vienna, as explained in Chapter 5.1.1, the penalty value is the maximum entry in the travel time matrix T. It is determined as 1019 seconds, which equals $16.98\overline{3}$ minutes.
A complete list of response times that takes every emergency into account can be calculated with the penalty value. For easier visualisations, this list can be averaged to a single average response time value.

Patrols

The **number of visited patrol locations** is calculated by counting the patrol locations that are saved as visited locations in the output file.
The **map coverage** in percent can be calculated by adding the travelled distances d_{ij} for all traversed edges and dividing the sum by the total network size:

$$\text{map\_coverage} = \frac{\text{unique\_travel\_distance}}{\text{total\_network\_size}} \tag{35}$$

To calculate a map coverage which is between 0 % and 100 %, travelled distances are only added once, even if a segment is used multiple times in the travelled routes. Some roads are one-way streets and others are not. Thus, two-way roads are also only counted once for the unique travelled distance and the total network size. The removed edges, as explained in Chapter 5.1, are ignored.

Resources

Adding up the distances for all traversed edges including duplicates yields the **total travelled distance**.
The **total travelled time** is calculated analogously to the distance, but also the time spent at emergencies or patrolling is added. Notably, time spent at the police station is not considered.

5.5 Simulation Settings

In this chapter, the used settings for the simulations are summarised. As explained in Chapter 5.1.2, a fixed set of 30 patrol locations is drawn randomly from a list of relevant amenity nodes for all simulations. An image of the map with the marked patrol locations and the used police station is given in Figure 8. The available police stations and the chosen one were also discussed in Chapter 5.1.2.
If real hotspot data are available, the nodes can simply be interchanged.

As reference for the settings in the configuration file of the simulation, Table 4 can be used.

Heuristic Method	Parallel Cheapest Insertion
Metaheuristic Method	Guided Local Search
Solution Limit	10^6
Time Limit	4 s
Simulation Duration	10 800 s (= 3 h)
Number of Vehicles	4
Patrolling Time per Location	600 s (= 10 min)
Number of Events μ	6
Number of Events σ	4
Earliest Starting Time of Emergency	0 s
Latest Starting Time of Emergency	10 800 s (= 3 h)
Minimum Event Duration	120 s (= 2 min)
Event Duration μ	900 s (= 15 min)
Event Duration σ	300 s (= 5 min)

Table 4: Settings used for the simulations. If not stated otherwise, those settings are used in Chapter 6.

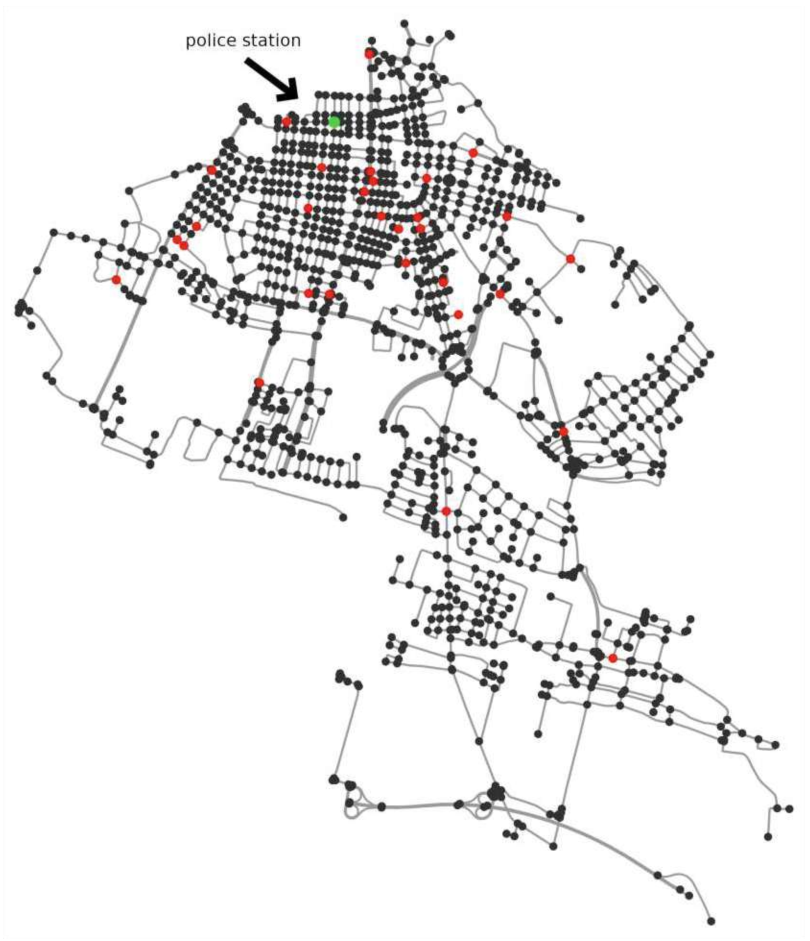


Figure 8: Network graph of the district Favoriten in Vienna, with patrol locations marked in red and the police station marked in green. (Map data © OpenStreetMap)

The first solution strategy, or construction heuristic, is an insertion method, as described in Chapter 3.2.1. For improving the solution, a metaheuristic algorithm is used. Metaheuristic algorithms were explained in Chapter 3.3. Mainly Guided Local Search is used in the following, which was explained in Chapter 3.3.3, but also Simulated Annealing is applied for some comparisons. Simulated Annealing was discussed in Chapter 3.3.4.
The simulation duration is chosen to be 3 hours and sets an upper time limit for the time windows of the patrolling locations. The patrol duration per

patrolling location is chosen to be 10 minutes, similar to the values in the literature [3], [5]. At the start of the simulation, all vehicles' starting locations are set to the police station.

The parameters for generating emergencies are also listed in Table 4. They were explained in Chapter 5.2.2. Those parameters can be replaced by real historic data or by data given by an expert.
A list of OR-Tools search parameters is given in Appendix B.

In the following chapters, some parameters are varied. Those parameters include the used metaheuristic method, the solution and time limit and the number of vehicles. Also, many simulations are conducted with and without patrolling locations. For the calculation of emergencies, seeds are used. Thus, for better comparability of the results, the same emergencies are generated. In Chapter 6, the settings and the number of simulations are noted. If not stated otherwise, the settings from Table 4 are used.

6 Results

In the following chapters, results and insights are discussed. Guided Local Search and Simulated Annealing are compared in Chapter 6.1. In Chapter 6.2, the quality criteria are analysed for different settings, showcasing the practical usability of the implemented model. The performance of the simulation is discussed in Chapter 6.3. The stopping rules and the quality criteria as target parameters are discussed in Chapter 6.4.

6.1 Guided Local Search and Simulated Annealing

In this chapter, Guided Local Search and Simulated Annealing are compared as solution methods for this simulation model. Notably, the final routes of the vehicles are compared, as they are considered a result of the simulation. The final routes include changes caused by emergencies.
The simulation is run for four vehicles and with different time limits for the route calculations. For each time limit, a sample of $n = 200$ simulations is calculated. For more comparability of the results, the same seeds and, therefore, the same emergencies are used for each set of calculations.
Results calculated with a solution limit of 10^9 and a time limit of 10 seconds are used as a reference for comparison. The amount of simulation outputs which are identical to the reference solutions are displayed in Table 5.

Solution Limit	1	10^6						10^9
Time Limit	1 s	1 s	2 s	3 s	4 s	5 s	6 s	8 s
Simulated Annealing	0	197	199	199	200			
Guided Local Search	0	50	132	157	176	186	189	198

Table 5: Number of identical simulation results compared to the calculations, with a time limit of 10 seconds and a solution limit of 10^9. The sample sizes are $n = 200$ respectively.

The values which are set for the time limit are the limiting factor. Only in the first case, the solution limit is set as low as possible to depict what happens at the beginning of the simulation.

In the time frame of the first second, most of the simulation results change if the Simulated Annealing algorithm is used. In the following 9 seconds, only three additional simulation results change. With the results already being the same as the reference solutions, the calculations for the empty cells are omitted. The simulation data collected with Guided Local Search change more gradually.

So far, it is not clear which algorithm performs better than the other in regards to the simulation. Due to having four vehicles available, barely any patrol locations or emergencies are missed, independently of the algorithm. To help deciding which metaheuristic algorithm will be used in the following, one of the quality criteria in Chapter 5.4.1 is used, namely the total travelled distance. The average total travelled distance over all 200 simulations should be as low as possible and is displayed in Figure 9.

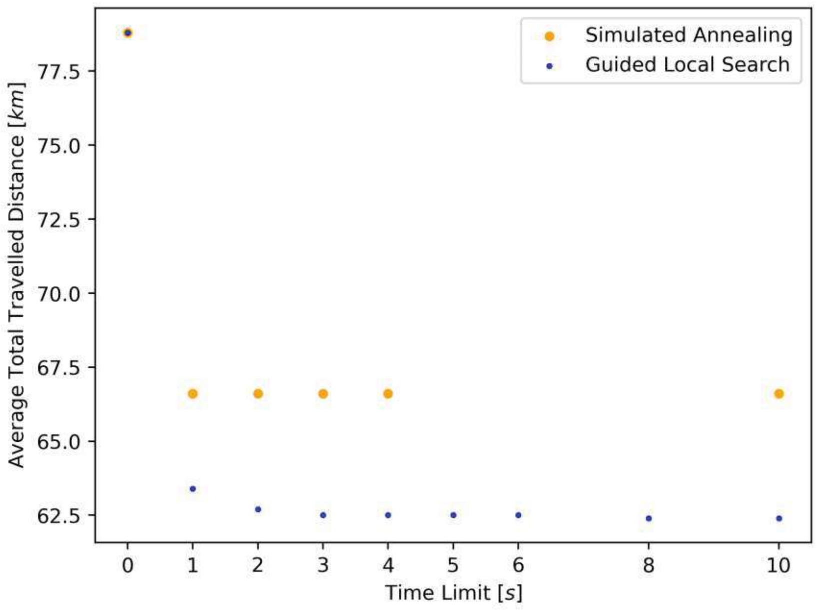


Figure 9: Average total travelled distances in kilometers for different time limits. The sample sizes are $n = 200$ respectively.

Obviously, by generating many of the same routes, the simulations using the Simulated Annealing metaheuristic yield almost the same average total travelled distances for different time limits.

As shown in Figure 9, the results calculated with the Guided Local Search algorithm are lower. For example, for a time limit of 4 seconds, the average total travelled distance is 66.61 km for Simulated Annealing and 62.47 km for Guided Local Search, which is 6.2 % less. The total travelled time is displayed in Appendix A in Figure 25. The results are similar, but percentage-wise, the differences are lower. The routes calculated for the special case without emergencies do not change after a calculation time of one second. The total travelled time and distance calculated with Guided Local Search are 20 399 seconds and 28.0 km, which are lower compared to 20 725 seconds and 32.6 km when using Simulated Annealing.
As the Guided Local Search metaheuristic with a time limit of 4 seconds already calculates 88 % of the simulation results received with 10 seconds, those parameters will be used in the following.

6.2 Analyses of Quality Criteria

To showcase the simulation model and assess the usability of the defined quality criteria, the simulation is conducted $n = 2000$ times, respectively for one, two, three and four police vehicles with and without patrolling at the previously defined patrolling locations. In the following, the special case of no emergencies is also referred to. With the randomness of the emergencies removed, only one simulation result is needed, which is already included in the other simulations.

6.2.1 Number of Emergencies

As described in Chapter 5.2.2, the Gaussian distribution is used for creating emergency data and if values below 0 appear, a new value is calculated. The results are shown in Figure 10. They are calculated with the in Chapter 5.5 defined mean $\mu = 6$ and standard deviations $\sigma = 4$. The average number of calculated emergencies for the used seeds is 6.48 and the calculated standard deviation is 3.60. As the same 2000 seeds are used for different settings – different amount of vehicles with and without patrolling locations –, the same sets of emergencies are created each time.

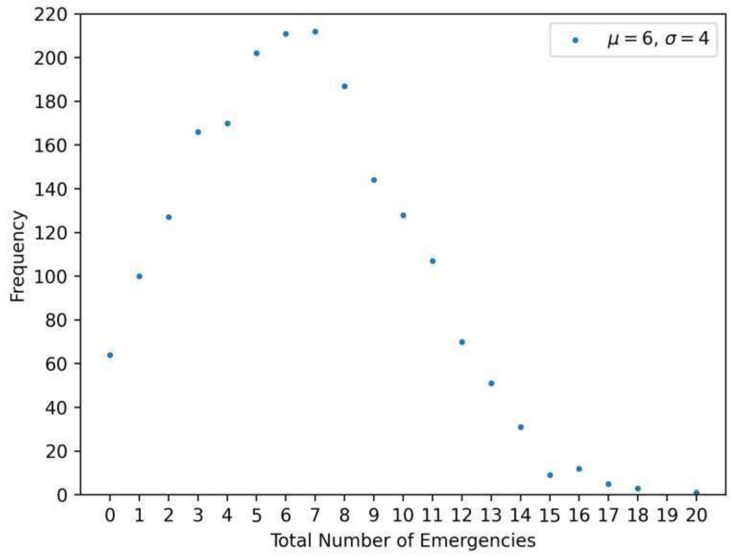


Figure 10: Frequency of the number of emergencies with mean $\mu = 6$ and standard deviations $\sigma = 4$, with a sample size of $n = 2000$.

6.2.2 Number of Missed Emergencies

With the used seeds, emergencies appear as frequently as 20 times per simulation. Naturally, with less vehicles available, the number of missed emergencies increases.

As shown in Table 6, removing a vehicle from the simulation increases the number of missed emergencies significantly. Removing one vehicle after another, starting with four, the average number of missed emergencies increases by a factor of 5.15, 4.17 and 3.13, despite only removing 25 %, 33.$\overline{3}$ % and 50 % of vehicles.

If the number of vehicles is decreased, the standard deviation increases. Four vehicles are always able to reach a minimum of 4 emergencies, even if they occur at the same time. One vehicle can theoretically miss all emergencies except one. Therefore, more variation in the results is plausible for a lower number of vehicles. Also, the maximum value of missed emergencies increases.

	Number of Missed Emergencies		
Number of Vehicles	Mean	SD	Maximum
1	2.690	2.346	14
2	0.859	1.257	9
3	0.206	0.563	5
4	0.040	0.224	3

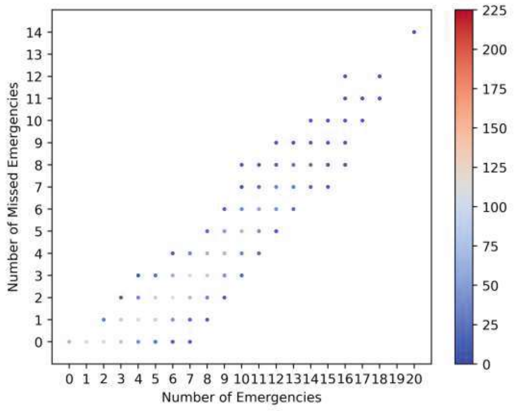
Table 6: Number of missed emergencies with calculated mean and standard deviation values for different numbers of patrolling vehicles. Also, the maximum occurred value is displayed.

The values are similar for the calculated simulation results without patrolling locations, as shown in Table 7. While the differences might not be significant, it is noticeable that all mean and standard deviation values are slightly higher than their respective values in Table 6. This observation might be explained by the following reasons: Without patrol locations, vehicles are no longer forced to move around the map and are more likely to be clumped together at or around the police station. Also, as shown previously in in Figure 8 in Chapter 5.5, the police station is located very peripheral in the north of the map.

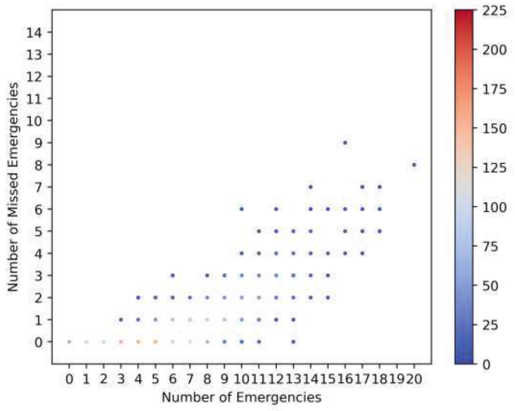
	Number of Missed Emergencies		
Number of Vehicles	Mean	SD	Maximum
1	2.716	2.361	14
2	0.900	1.291	9
3	0.229	0.603	6
4	0.045	0.246	3

Table 7: Number of missed emergencies with calculated mean and standard deviation values for different numbers of patrolling vehicles. Also, the maximum occurred value is displayed. The vehicles are not patrolling.

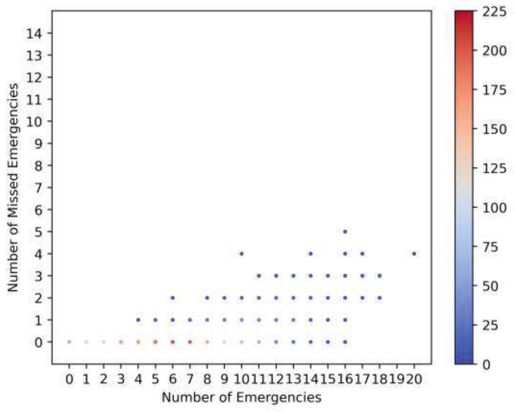
The impact of the number of emergencies on the number of missed emergencies is shown in Figure 11. As already discussed, a greater number of available vehicles decreases the number of missed emergencies. Figure 11 visualises that additional vehicles are especially beneficial for large numbers of emergencies, which are more likely to overlap in time.



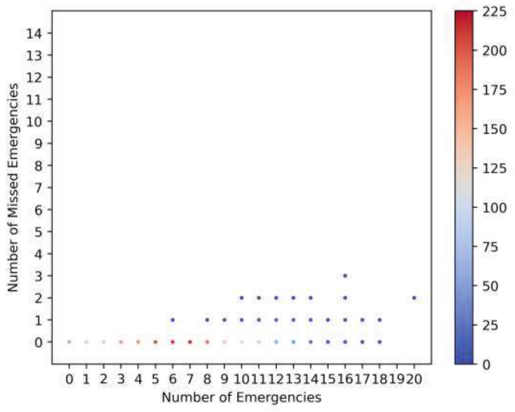
(a) One Police Vehicle



(b) Two Police Vehicles



(c) Three Police Vehicles



(d) Four Police Vehicles

Figure 11: Number of missed emergencies depending on the number of emergencies. Displayed for one, two, three and four patrolling police vehicles.

As shown by the colour grading used in Figure 11, markers with the highest frequency are distributed around the average number of emergencies, which is 6.48. Values marked in darker blue colour appear less often. While the mean number of missed emergencies might be low, each missed emergency represents an additionally needed vehicle. Therefore, from a practical point of view, the maximum amount of missed emergencies might also be important to consider.

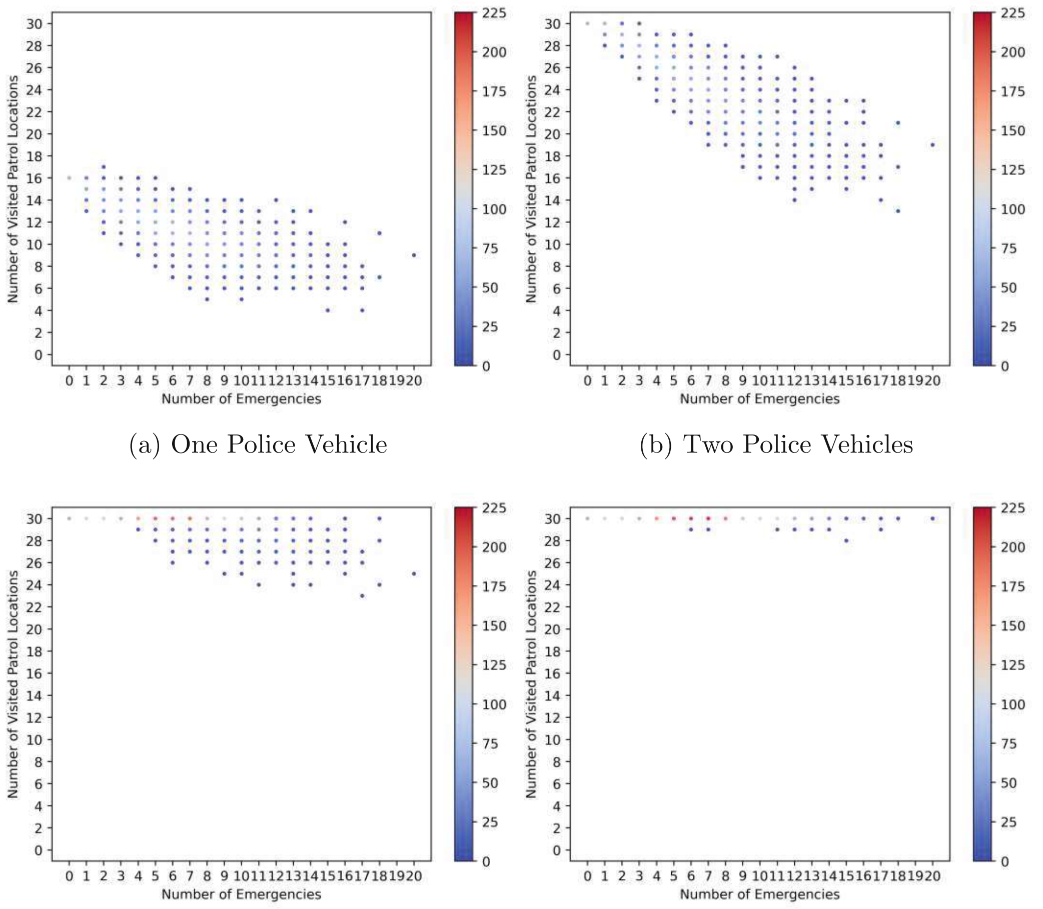
6.2.3 Number of Visited Patrol Locations

The number of visited patrol locations is shown in Table 8. Obviously, more vehicles are able to patrol more locations. On average, three vehicles already reach 29.678 out of the total 30 patrol locations. While the calculated mean is already almost at the maximum value, the fourth vehicle still increases the minimum number.

	Number of Visited Patrol Locations			
Number of Vehicles	Mean	SD	Minimum	Maximum
1	11.503	2.315	4	17
2	24.549	3.376	13	30
3	29.678	0.868	23	30
4	29.992	0.095	28	30

Table 8: Number of visited patrol locations with calculated mean and standard deviation for different numbers of patrolling vehicles. Minimum and maximum values are also displayed.

Figure 12 shows the number of visited patrol locations, depending on the number of emergencies. If no emergency appears, one vehicle is able to visit 16 patrol locations, while two, three and four vehicles are able to patrol all. Without emergencies, the maximum amount of two vehicles is sent patrolling by the model. The remaining vehicles stay at the police station.



(a) One Police Vehicle

(b) Two Police Vehicles

(c) Three Police Vehicles

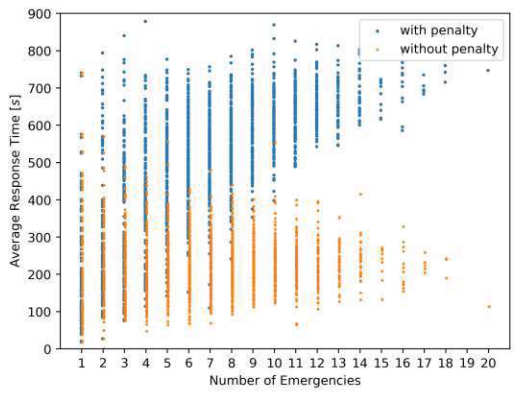
(d) Four Police Vehicles

Figure 12: Number of visited patrol locations depending on the number of emergencies. Displayed for different numbers of patrolling police vehicles.

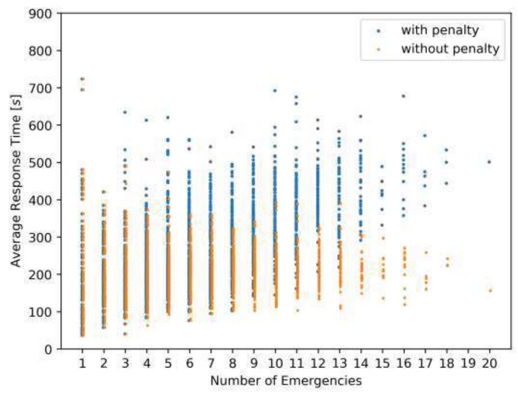
In Table 8, the maximum number of visited patrol locations for one vehicle is 17. Looking at Figure 12a, this maximum is reached in a simulation with two emergencies. With the used assumptions, emergencies can be advantageous for the number of visited patrol locations. A location is marked as patrolled, even if only a part of the total patrolling time is spent. Also, emergencies can occur at patrol locations. For three and more vehicles, emergencies can also be beneficial. As stated previously, at least two patrolling vehicles are needed to patrol all locations if no emergencies occur. Emergencies force vehicles to move to different positions on the map and occupy them for a certain amount of time. Depending on the state of the simulation, another vehicle might be forced to leave the police station. After the emergency is dealt with, it might not be optimal to drive back to the police station, but to visit patrol locations instead.

6.2.4 Response Times

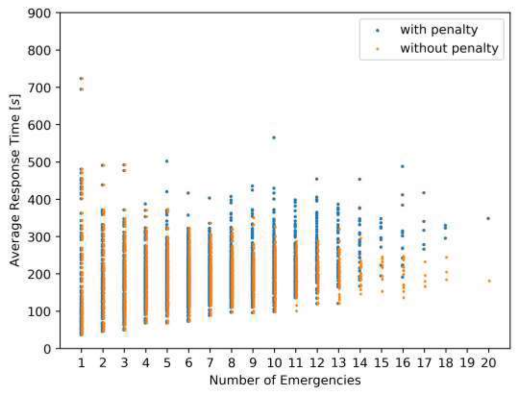
The average response times for each simulation are displayed with and without penalty for missed emergencies in Figure 13.



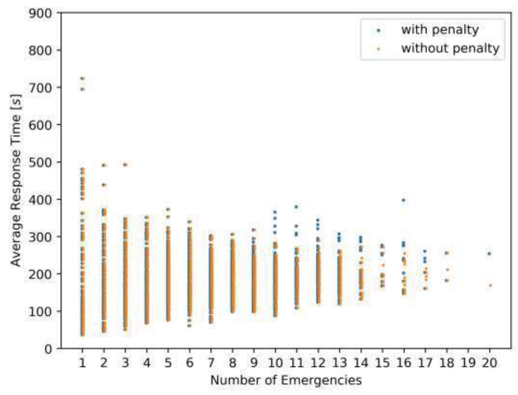
(a) One Police Vehicle



(b) Two Police Vehicles



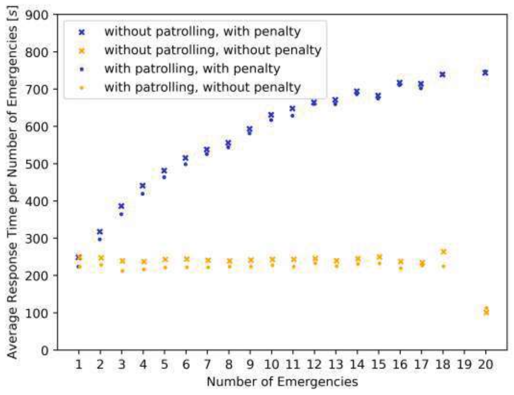
(c) Three Police Vehicles



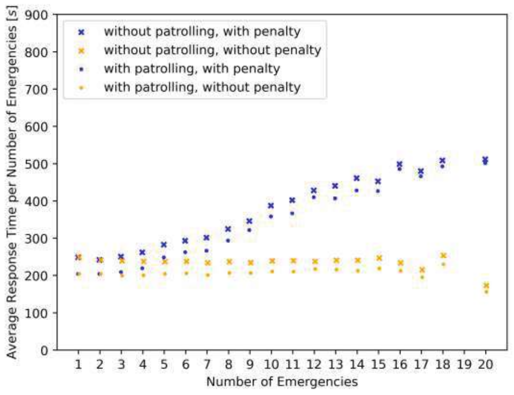
(d) Four Police Vehicles

Figure 13: Average response times for each simulation depending on the number of emergencies. Displayed for different numbers of police vehicles.

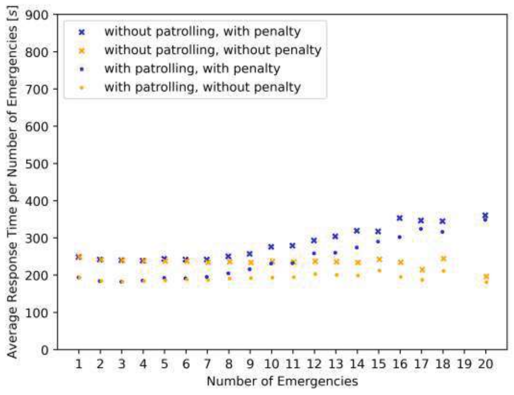
The charts show the importance of the added penalty value in the calculation of average response times. Without penalty for missed emergencies, less vehicles would barely yield worse response times, which is unrealistic. With the penalty value, however, the impact of additional vehicles is clearly visible. This also shows the importance of choosing a good penalty value. The average response times for each simulation without patrolling is shown in Figure 26 in Appendix A. The figure appears to be similar to Figure 13.



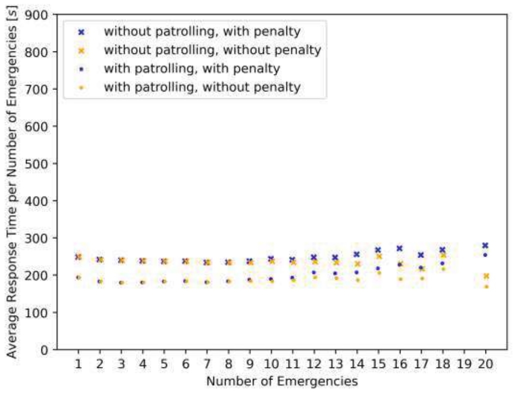
(a) One Police Vehicle



(b) Two Police Vehicles



(c) Three Police Vehicles



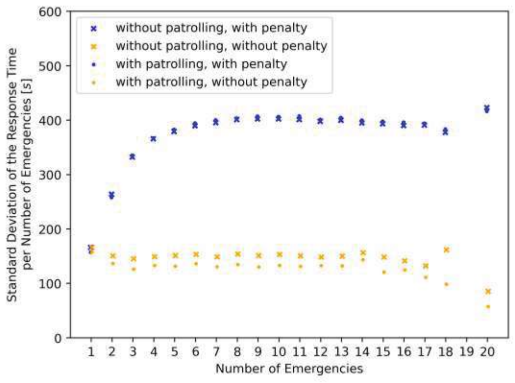
(d) Four Police Vehicles

Figure 14: Response times averaged for each number of emergencies. The average is calculated for simulations with and without patrolling locations and displayed depending on the number of emergencies for different numbers of police vehicles.

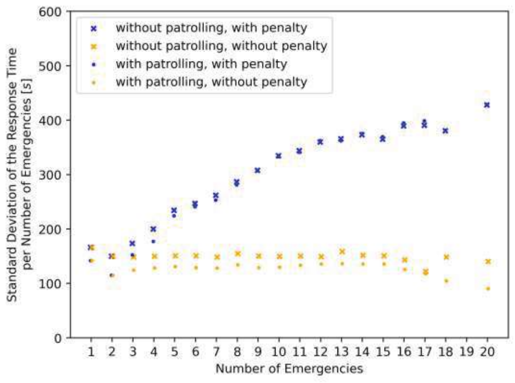
Notably, the values without patrolling points of interest are higher than their counterparts with patrolling. Also, the response times without penalty values appear to be almost constant. Thus, no significant impact on the response times is suggested for emergencies that are not missed.

The response times averaged for each number of emergencies are displayed in Figure 14. The values are calculated for the simulations with and without patrolling and with and without penalty. As shown previously in Chapter 6.2.1 in Figure 10, only a few simulations were designed to generate high numbers of emergencies. Therefore, higher numbers of emergencies have a small sample size and the displayed values are less representative.

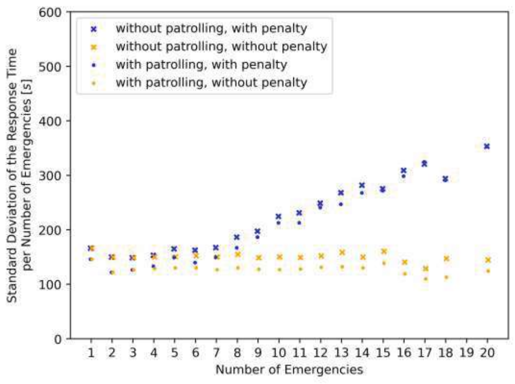
In Figure 15, the standard deviations are displayed for each number of emergencies. With the exceptions of one and two vehicles with penalty, the calculated values are again higher for simulations without patrolling, compared to those with patrolling. Notably, the standard deviation only increases significantly if a penalty value is used. This suggests that the response times for emergencies do not spread significantly more if more emergencies occur. Again, the smaller sample size for higher numbers of emergencies might result in less representative results.
To examine the influence of the number of emergencies on the response time in greater detail, it might be beneficial to change the probability distribution. For example, uniformly distributed numbers of emergencies could be used.



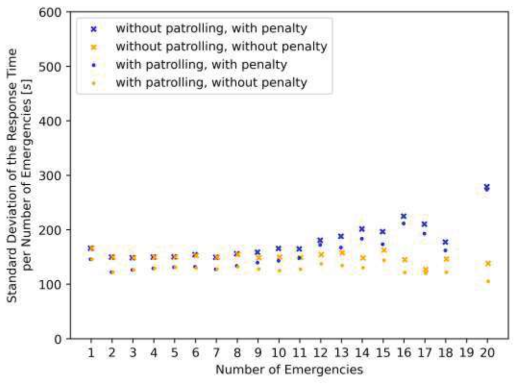
(a) One Police Vehicle



(b) Two Police Vehicles



(c) Three Police Vehicles



(d) Four Police Vehicles

Figure 15: Standard deviations of the response times for each number of emergencies. The standard deviation is calculated for simulations with and without patrolling locations and displayed depending on the number of emergencies for different numbers of police vehicles.

6.2.5 Total Travelled Distance

The average total travelled distances and their associated standard deviations are displayed in Figure 16.

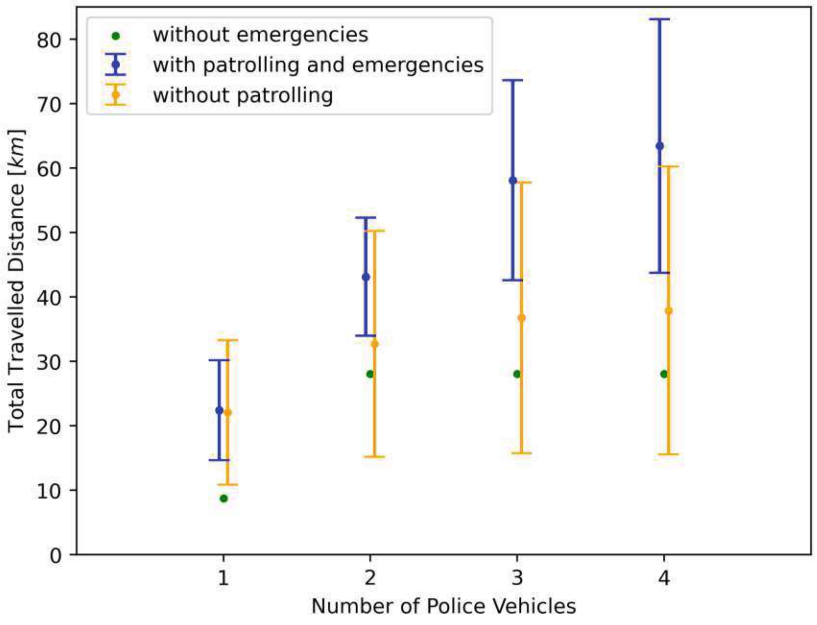


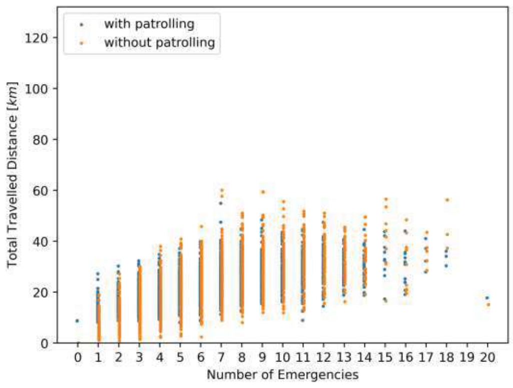
Figure 16: Average total travelled distances with standard deviations in kilometers for different numbers of police vehicles. Displayed for simulations with and without patrolling locations and also without emergencies.

The case without emergencies is the simplest: As all randomness is removed, all simulations calculate the same output, so only one simulation ($n = 1$) is used. Increasing the number of available vehicles from one to two increases the average total travelled distance from 8.716 km to 28.029 km. Additional vehicles yield no further improvement, as only a maximum of two vehicles is needed to visit all patrol locations.

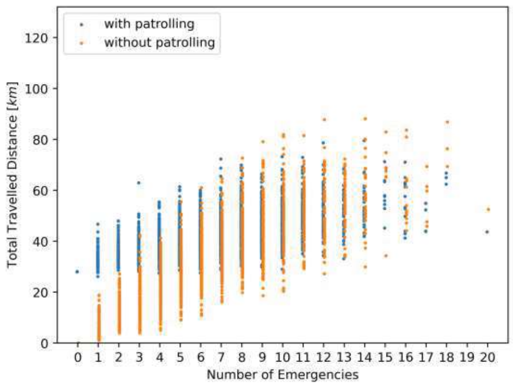
Emergencies force more vehicles to travel. For the simulations with emergencies, the average total travelled distances increase with the number of available vehicles. Notably, not only the mean increases if more vehicles are available. Also, the standard deviation increases with the number of vehicles. As one vehicle is already used at full capacity, similar mean values are re-

ceived with (22.416 km) and without patrolling (22.057 km). Adding a second vehicle almost doubles the average total travelled distance with patrolling (43.123 km). Without patrolling (32.689 km), the improvement is not as significant. Four vehicles, however, reach almost always all 30 patrolling locations and rarely miss emergencies. With the additional capacities, the mean total travelled distances increase to 63.443 km with and 37.899 km without patrolling. Also, due to the additional capacities, the variance of emergencies can be carried over to the total travelled distances.
The standard deviation is larger for simulations without patrolling locations, compared to those with patrolling vehicles. This is expected, as patrolling forces vehicles to travel, independent of emergencies. In a way, a lower bound for the total travelled distances is established, while the simulation duration provides an upper bound. Notably, the total travelled distance without emergencies can theoretically be undershot, as vehicles can be forced to spend time at emergencies instead of driving.

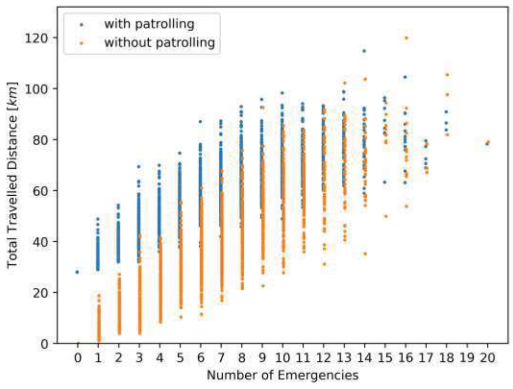
Figure 17 shows the total travelled distances for each simulation, broken down for each number of emergencies. The impact of patrolling becomes visible first for lower numbers of emergencies. Additional vehicles provide capacities to set the two clouds of data points further apart.



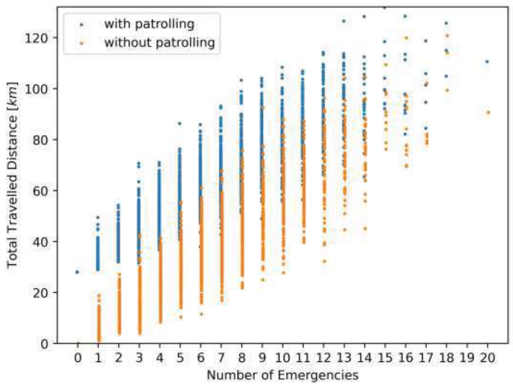
(a) One Police Vehicle



(b) Two Police Vehicles



(c) Three Police Vehicles



(d) Four Police Vehicles

Figure 17: Total travelled distances for each simulation depending on the number of emergencies. Displayed for different numbers of police vehicles.

6.2.6 Map Coverage

With the map coverage being closely related to the total travelled distance, the results are expected to behave in a similar way. However, as displayed in Figure 18, there are notable differences. When increasing the number of vehicles from three to four, there is a lack of improvement. As explained in Chapter 5.4.2, edges are only counted once for calculating the map coverage. While additional vehicles are able to travel more, the additional fourth police vehicle is not very likely to drive on many new road segments.

Thus, the map coverage only improves from 17.56 % to 17.63 % with and from 8.19 % to 8.31 % without patrolling.

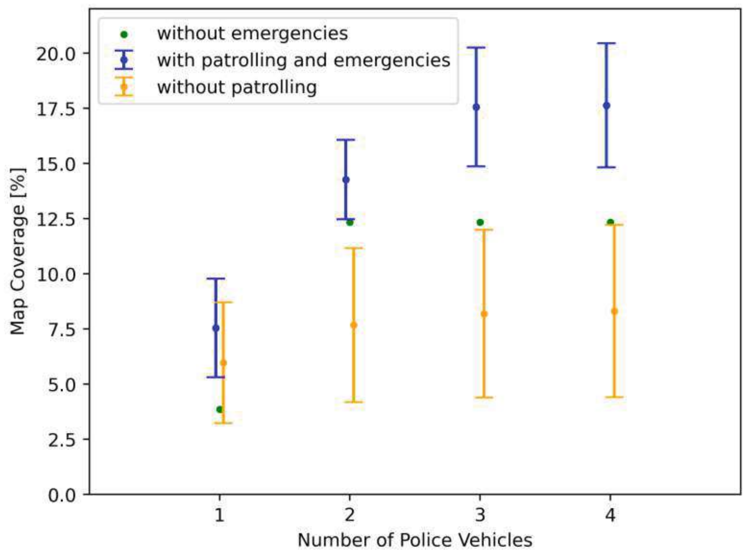
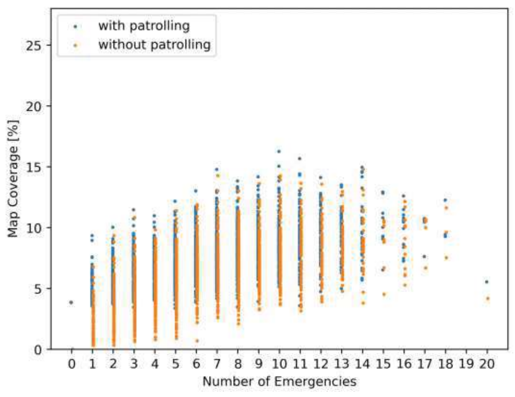


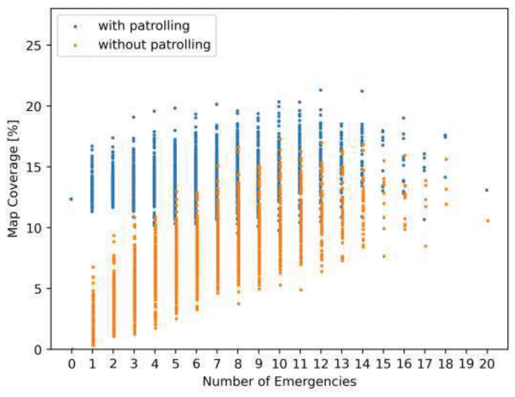
Figure 18: Average map coverage in percent for different numbers of police vehicles. Displayed for simulations with and without patrolling locations. Standard deviations in percentage points are displayed as error bars.

Patrolling vehicles without emergencies follow a route, calculated with the Guided Local Search algorithm. As this route is supposed to be an approximation of the fastest route possible, it is not expected to include many duplicated edges. On average, simulations with emergencies and without patrolling yield a greater total travelled distance for two and more vehicles, compared to the reference case without emergencies. Therefore, the lack of map coverage can be explained by the repeated use of the same roads of the road network, when only driving to emergencies. Again, the map coverage is constant for two and more vehicles if no emergencies appear.
Similar to the previous chapter, the standard deviation is smaller for patrolling vehicles with emergencies, compared to the data from the simulations in which patrolling is omitted.

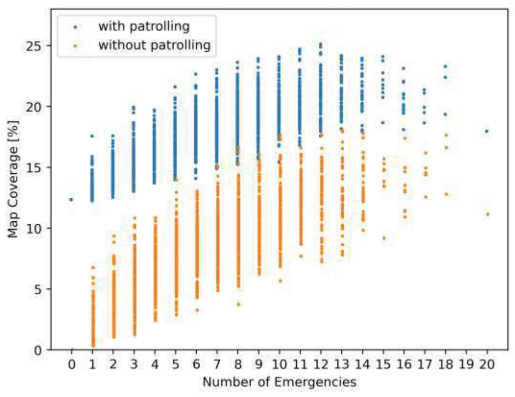
In Figure 19, the influence of the number of emergencies on the map coverage is displayed. Without patrolling and with few emergencies, the map coverage is low and increases with the number of emergencies. Again, patrolling enforces a lower bound and increases the map coverage. Hence, the lower variance observed in Figure 18 can be explained.



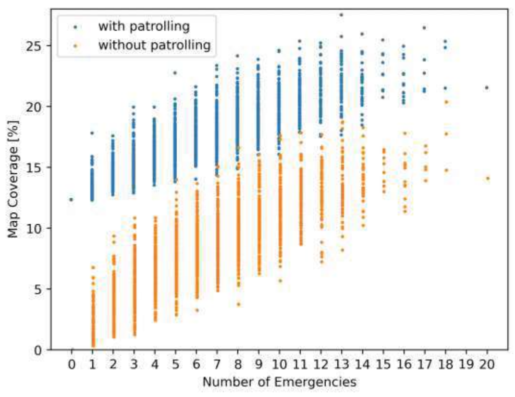
(a) One Police Vehicle



(b) Two Police Vehicles



(c) Three Police Vehicles



(d) Four Police Vehicles

Figure 19: Map coverages for each simulation depending on the number of emergencies. Displayed for different numbers of police vehicles.

6.2.7 Total Travelled Time

The mean values and the standard deviations for the total travelled times are displayed in Figure 20. Notably, waiting times at the police station is not counted towards the total travelled time. Therefore, the simulations without patrolling only collect time while responding to emergencies. This yields lower average total travelled times.

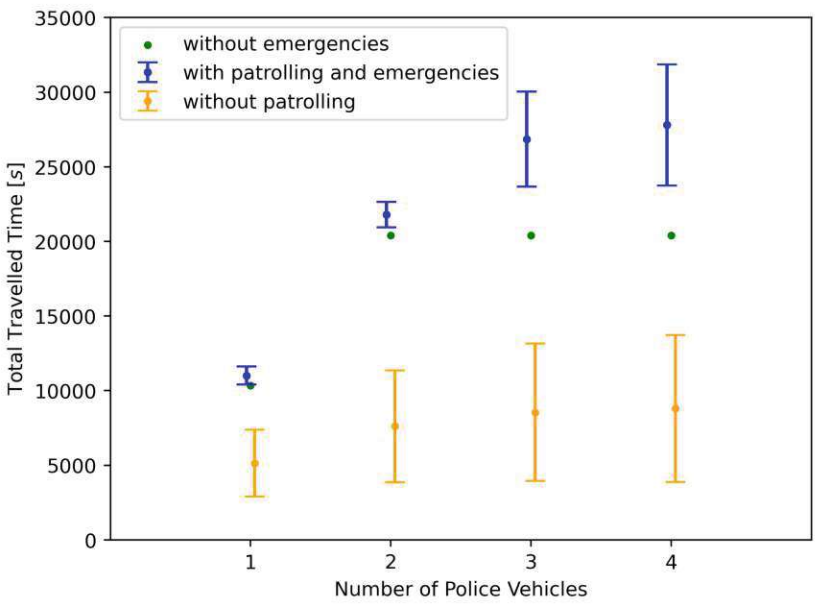
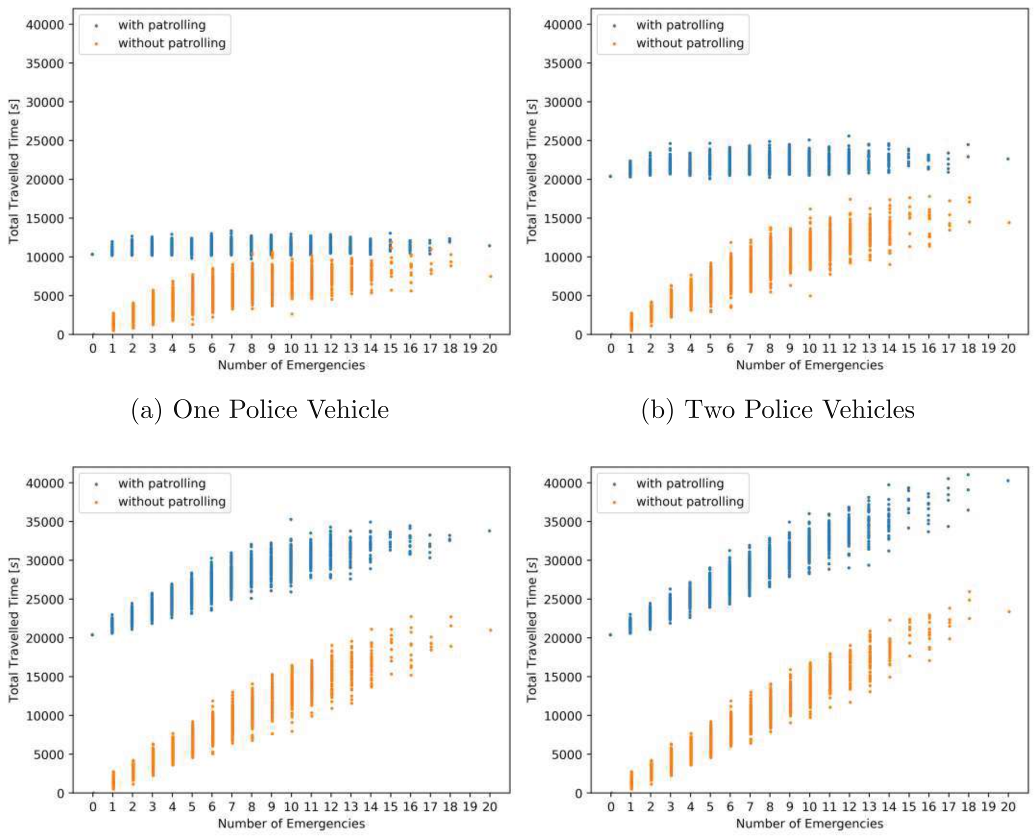


Figure 20: Average total travelled times in seconds for different numbers of police vehicles. Standard deviations are visualised with error bars. Displayed for simulations with and without patrolling locations.

The small standard deviations for one and two vehicles with patrolling locations can be explained by exhausted capacity. One vehicle travels on average 11 016 seconds which is equal to 3.06 hours. Two vehicles travel 21 804 seconds or 6.06 hours. The total patrolling time is set to 3 hours (10 800 seconds), but vehicles still have to return to the police station if time has run out. Also, emergencies can last longer than the planned maximum patrolling time.

As shown in Figure 21, one and two vehicles reach their maximum capacity. Therefore, the total travelled times appear to be independent of the number of emergencies. Three and especially four vehicles are still able to deal with higher numbers of emergencies and, therefore, are able to invest additional time.



(a) One Police Vehicle

(b) Two Police Vehicles

(c) Three Police Vehicles

(d) Four Police Vehicles

Figure 21: Total travelled times for each simulation depending on the number of emergencies. Displayed for different numbers of police vehicles.

Comparing Figure 21b with the time frame of 3 hours, it appears as if without patrolling, two vehicles still have additional time capacities. In fact, as discussed previously in Chapter 6.2.2, similar amounts of emergencies are missed. Therefore, the unused time capacities are spent waiting at the police station.

6.3 Performance of the Simulation Model

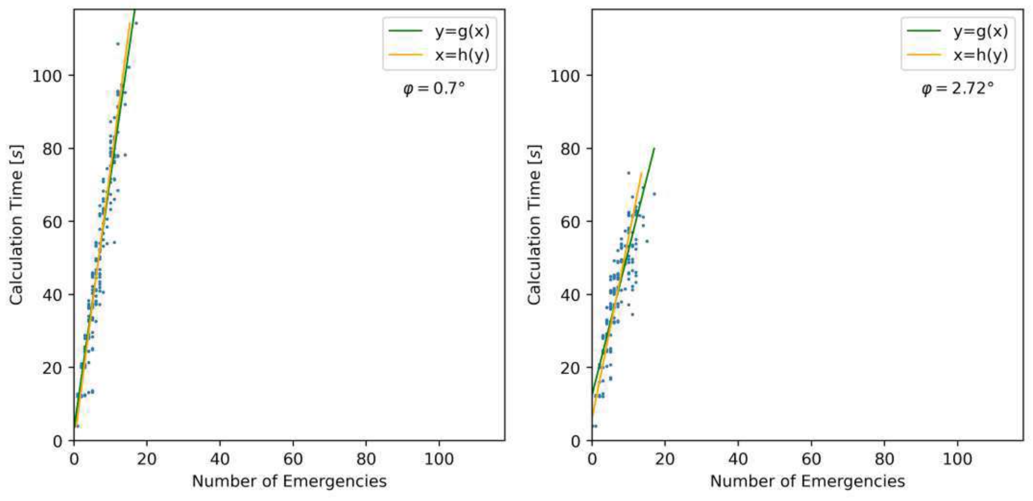
The simulations were performed using a computer with the following specifications:

- Operating System: Ubuntu 20.04.6 LTS
- Processor: Intel® Core™ i7-4790 CPU @ 3.60GHz $\times$ 8
- RAM: 32 GB
- Python 3.7.6

The full list of used Python modules and their version numbers can be found on GitHub [53].

The biggest amount of calculation time is spent for planning the routes. In this step, time is invested until the solution limit or the time limit is exceeded. Naturally, the question occurs, how often routes are calculated. Routes can be calculated up to two times the number of emergencies. The first time is when a police vehicle is sent to an emergency and the second time is when it continues patrolling. However, there are two cases in which recalculating the routes is not necessary: if no vehicles are available because they are all at emergencies and if no more locations are left for patrolling. The latter also includes exceeding the time windows of the patrol locations. In this case, all vehicles take the fastest route to the police station.

In Figure 22, the calculation time is displayed depending on the number of emergencies. A sample size of $n = 200$ simulations is used. In Figure 22a, the regression lines are almost identical and the angle between them is only 0.7°. If the number of police vehicles is reduced to two, as shown in Figure 22b, less vehicles are available and less data points with high calculation time can be found.



(a) Four Vehicles (b) Two Vehicles

Figure 22: Comparison of the calculation times for simulations with two and four vehicles. As less vehicles are available, less route planning is performed. Therefore, less calculation time is needed.

For a calculation time of 4 seconds, a slope of up to 8 is expected. In reality, the slopes of the regression lines are about 6.89 and 7.53 in Figure 22a. As expected, the slopes for two vehicles are even lower, with about 3.97 and 4.95 seconds per emergency, as shown in Figure 22b.
Simulations with no emergencies are calculated in about 4 seconds. In this case, the routes are calculated exactly once. If no patrol locations are defined, the calculations are even faster, as only the fastest route to an emergency location or back to the police station is calculated.

In Figure 23, different time limits are used, each with a sample size of $n = 200$ and four vehicles. The seeds for each set of simulations are the same and, therefore, each set uses the same emergencies. The calculation time divided by two times the average number of emergencies is expected to be similar to the time limit and, therefore, estimated by the identity function.

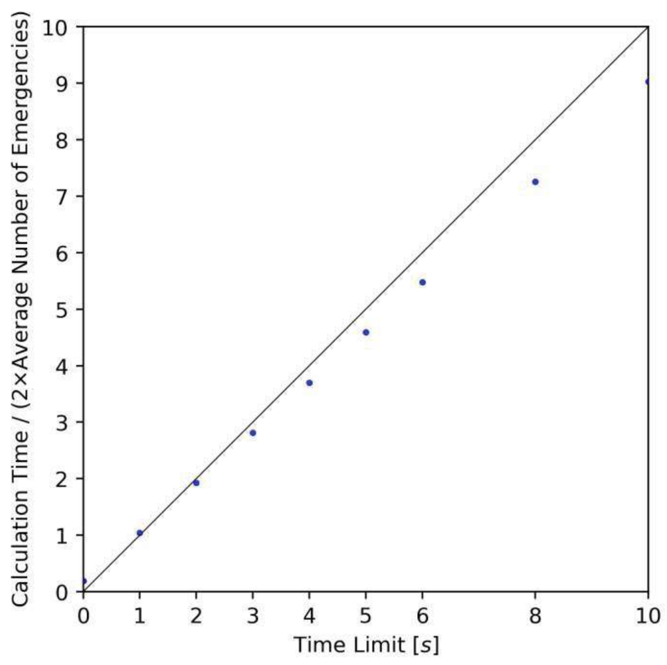


Figure 23: Comparison of the time limits and the calculation times divided by two times the average number of emergencies.

As shown in Figure 23, for time limits lower than 2 seconds, the calculation time divided by two times the average number of emergencies can be higher than the identity function. For higher calculation times, the identity function serves as an upper bound.

6.4 Stopping Rules for Monte Carlo Simulations

As discussed in Chapter 4.5, stopping rules can be used for Monte Carlo Simulations. In this chapter, an adaptation of Algorithm 7 with the Chebyshev stopping rule is used, which is defined in Formula 25. The simulations are run beforehand and only the results are used in the algorithm. This approach allows to use different quality criteria without the need of time-consuming recalculations of the simulations.

With a given confidence level p and an interval defined by δ_{abs}, an estimate for the number of necessary simulations can be calculated. Upon reaching this estimate, the real mean is determined to be inside the confidence interval, with a probability of at least the confidence level.
In the following, the previously defined quality criteria are used to show a few examples. The dataset with $n = 2000$, a time limit of four seconds, four patrolling vehicles and the Guided Local Search metaheuristic is used. The confidence level is assumed to be $p = 0.95$ in this chapter. δ_{abs} is dependent on the target parameter and the chosen settings.

Most of the quality criteria, as defined in Chapter 5.4.1 and as calculated in Chapter 5.4.2, can be used for calculating M_{stop}. Examples for the simulation with four patrolling vehicles are displayed in Table 9.

	δ_{abs}	M_{stop}	$\overline{X}_{M_{stop}}$
Total Travelled Time	450 s (= 7 min 30 s)	1591	27 783.79 s (≈ 7 h 43 min)
Total Travelled Distance	2 km	1951	63.345 km
Map Coverage	0.5 p.p.	599	17.467 %
Number of Emergencies	0.5	1008	6.39
Number of Missed Emergencies	0.05	492	0.043
Number of Visited Patrol Locations	$2.04 \cdot 10^{-7}$	111	$30 + 4 \cdot 10^{-15}$

Table 9: Examples for the usage of stopping rules with confidence level $p = 0.95$ and $M_0 = 100$. The sample calculated with four patrolling police vehicles is used. δ_{abs} is chosen and defines the confidence interval. Algorithm 7 terminates at the stopping index M_{stop} and the calculated mean value $\overline{X}_{M_{stop}}$ is returned.

Most of the defined δ_{abs} in Table 9 appear to be intuitive. However, the number of visited patrol location stands out with a very small δ_{abs}. The calculated M_{stop} barely surpasses $M_0 = 100$. In the chosen dataset with four vehicles, almost always all 30 patrol locations are visited. Therefore, the sample variance is very small. In Algorithm 7, the sample variance is used as replacement for the real variance for the Chebyshev stopping rule. Thus, for many chosen δ_{abs}, the algorithm terminates right after passing M_0.
In practice, if the number of visited patrol locations is chosen as target parameter for the stopping rules, δ_{abs} can be unintuitive to define. This quality criterion is very dependent on the used simulation settings.
For comparison, in simulations with only one vehicle, more emergencies are missed and less patrol locations are visited on average. A confidence interval can be defined by $\delta_{abs} = 0.3$ for the number of visited patrol locations and $M_{stop} = 1161$ and $\overline{X}_{M_{stop}} = 11.50$ can be calculated.
To sum up, the variance of target parameters heavily impacts the outcome. For practical use, it is important to be aware to not misinterpret results with different settings.

The response time is the only quality criterion which is not mentioned in Table 9. The response time, calculated as explained in Chapter 5.4.2, is a list of values. The length of the list varies, depending on the number of emergencies. Calculating the average response time for each simulation and then averaging again, as it would be done in Algorithm 7, is problematic. Single response times would not be weighted equally.
Theoretically, Algorithm 7 could be adapted and the response times could be used individually for the calculations in the algorithm. However, in this case, simulations with many emergencies would enter an over-proportional amount of values.
Due to this difficulty, the average response time cannot be easily used as target parameter in Algorithm 7 to calculate a corresponding M_{stop}.

7 Conclusion and Outlook

To target the Police Patrolling Routing Problem, a simulation model was built. The model uses real map data of the district Favoriten in Vienna. The data were retrieved from OpenStreetMap and prepared to be used in the simulation. With real amenity data, points of interest were defined. A sample of points of interest was chosen as patrol locations, which were supposed to be visited in certain time windows. Emergency data were simulated and used to create interruptions for patrolling vehicles.
The Dynamic Vehicle Routing Problem was reduced to the underlying Vehicle Routing Problem, which is solved repeatedly if interruptions occur. Metaheuristic algorithms were used to approximate solutions. Therefore, OR-Tools, Google's software suite for operations research, was integrated in the simulation. Simulated Annealing was compared to Guided Local Search. As Guided Local Search outperformed Simulated Annealing for the purpose of this simulation and the used settings, Guided Local Search was used for the majority of the simulations.
To retrieve results for statistical analyses, Monte Carlo Simulations were performed. Quality criteria were defined as key performance indicators for a patrol. The quality criteria include the number of missed emergencies, the vehicle response time, the number of visited patrol locations, the map coverage, the total travelled time, the total travelled distance and the number of emergencies.

With this simulation model, it was possible to calculate practical results in a very reasonable runtime and to show the impact of changed simulation settings, for example when removing the patrolling locations or varying the number of patrolling vehicles from one to four. The quality criteria proved to be useful to measure the influence of changed settings on the patrol. Also, workload and capacity of the patrolling vehicles could be visualised. The implemented simulation appears to be a promising tool for decision support in police patrol planning by offering approximations of required resources and expected benefits.

It was found that the response time was more difficult to analyse than the other quality criteria, especially when attempted to be used for stopping rules for Monte Carlo Simulations.

The performance of the model with the used settings was very dependent on the time limit of the OR-Tools solver. Therefore, an approach which considers the problem size, namely the number of patrol locations, could improve calculation times.

While the simulation model succeeded to deliver useful results, the Police Patrolling Routing Problem has many facets and is very comprehensive. Therefore, the model which tries to depict the reality in a simplified form can be further improved. For example, the simulated emergency data or the patrolling locations can be replaced by data from real crime statistics or by data given by an expert. Also, actual police practice can be incorporated into the model. For example, emergencies that require more than one police vehicle could be added to the model.

8 References

[1] Kelling, G.L.; Pate, T.; Dieckman, D.; Brown, C.E. *"The Kansas City Preventive Patrol Experiment"*. Police Foundation, Washington, DC, USA, 1974.

[2] Felson, M.; Clarke, R. "Opportunity Makes the Thief: Practical Theory for Crime Prevention". *Home Office*, vol. 98, Jan. 1998.

[3] Sherman, L.; Weisburd, D. "General deterrent Effects of Police Patrol in Crime "HOT SPOTS": A randomized, controlled Trial". *Justice Quarterly*, vol. 12: pages 625–648, Dec. 1995.

[4] Woodson, T. "Weapons of Math Destruction". *Journal of Responsible Innovation*, vol. 5(no. 3): pages 361–363, 2018.

[5] Koper, C. "Just enough Police Presence: Reducing Crime and disorderly Behavior by optimizing Patrol Time in Crime Hot Spots". *Justice Quarterly - JUSTICE Q*, vol. 12: pages 649–672, Dec. 1995.

[6] Dewinter, M.; Vandeviver, C.; Beken, T.; Witlox, F. "Analysing the Police Patrol Routing Problem: A Review". *ISPRS International Journal of Geo-Information*, vol. 9: p. 157, Mar. 2020.

[7] Pillac, V.; Gendreau, M.; Guéret, C.; Medaglia, A. "A Review of Dynamic Vehicle Routing Problems". *European Journal of Operational Research*, vol. 225: pages 1–11, Feb. 2013.

[8] Braekers, K.; Ramaekers, K.; Nieuwenhuyse, I. "The Vehicle Routing Problem: State of the Art Classification and Review". *Computers & Industrial Engineering*, vol. 99, Dec. 2015.

[9] Dantzig, G.; Fulkerson, R; Johnson, S. "Solution of a Large-Scale Traveling-Salesman Problem". *Journal of the Operations Research Society of America*, vol. 2: pages 393–410, Nov. 1954.

[10] Korte, B. H.; and Vygen, J. *"Combinatorial Optimization: Theory and Algorithms"*. Springer-Verlag, New York, NY, fifth edition, 2012.

[11] Tan, S.-Y.; Yeh, W.-C. "The Vehicle Routing Problem: State-of-the-Art Classification and Review". *Applied Sciences*, vol. 11: p. 10295, Nov. 2021.

[12] Ilavarasi, K.; Suresh Joseph, K. "Variants of Travelling Salesman Problem: A Survey". In *International Conference on Information Communication and Embedded Systems (ICICES2014)*, pages 1–7, 2014.

[13] Dantzig, G.B.; Ramser, J.H. "The Truck Dispatching Problem". *Management Science*, vol. 6: pages 80–91, Oct. 1959.

[14] Lenstra, J. K.; Rinnooy Kan, A. H. G. "Complexity of Vehicle Routing and Scheduling Problems". Econometric Institute Archives 272191, Erasmus University Rotterdam, Jun. 1979.

[15] Kallehauge, B.; Larsen, J.; Madsen, O.B.G.; Solomon, M.M. *"Column Generation"*, chapter "Vehicle Routing Problem with Time Windows", pages 67–98. Springer US, Boston, MA, 2005.

[16] Weisburd, D. L.; Telep, C. W.; Braga, A. A. *"The Importance of Place in Policing: Empirical Evidence and Policy Recommendations"*. Brottsförebyggande rådet, Stockholm, Sweden, 2010.

[17] Sak, T.; Wainer, J.; Goldenstein, S. "Probabilistic Multiagent Patrolling". In *Advances in Artificial Intelligence - SBIA 2008*, pages 124–133, Springer Berlin Heidelberg, Berlin, Heidelberg, Oct. 2008.

[18] Rossi, F.; van Beek, P.; Walsh, T. *"Handbook of Constraint Programming"*. Elsevier Science, 2006.

[19] Apt, K. R. *"Principles of Constraint Programming"*. Cambridge University Press, Cambridge, UK, 2003.

[20] Skiena, S. S. *"The Algorithm Design Manual"*. Springer, London, UK, 2008.

[21] Bellman, R. E. *"The Theory of Dynamic Programming"*. RAND Corporation, Santa Monica, CA, USA, 1954.

[22] Morrison, D. R.; Jacobson, S. H.; Sauppe, J. J.; Sewell, E. C. "Branch-and-bound Algorithms: A Survey of recent Advances in Searching, Branching, and Pruning". *Discrete Optimization*, vol. 19: pages 79–102, 2016.

[23] Marti, R.; Sevaux, M.; Sörensen, K. "50 Years of Metaheuristics". *European Journal of Operational Research*, Apr. 2024.

[24] Liu, F.; Lu, C.; Gui, L.; Zhang, Q.; Tong, X.; Yuan, M. "Heuristics for Vehicle Routing Problem: A Survey and Recent Advances", Mar. 2023.

[25] Laporte, G.; Ropke, S.; Vidal, T. *"Vehicle Routing: Problems, Methods, and Applications"*, chapter "Heuristics for the Vehicle Routing Problem", pages 87–116. Society for Industrial and Applied Mathematics, Nov. 2014.

[26] Clarke, G.; Wright, J. W. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". *Operations Research*, vol. 12(no. 4): pages 568–581, 1964.

[27] Christofides, N. *"Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem"*. Management Sciences Research Report No. 388. Management Science Research Group, Graduate School of Industrial Administration, Carnegie-Mellon University, Feb, 1976.

[28] Lin, S. "Computer Solution of the Traveling Salesman Problem". *Bell System Technical Journal*, vol. 44(no. 10): pages 2245–2269, Dec. 1965.

[29] Quenel Boussaid, I.; Lepagnot, J; Siarry, P. "A Survey on Optimization Metaheuristics". *Information Sciences*, vol. 237: pages 82–117, Nov. 2013.

[30] Solis, F.; Wets, R. "Minimization by Random Search Techniques". *Mathematics of Operations Research - MOR*, vol. 6: pages 19–30, Feb. 1981.

[31] Mladenović, N.; Hansen, P. "Variable Neighborhood Search". *Computers & Operations Research*, vol. 24(no. 11): pages 1097–1100, 1997.

[32] Glover, F. "Future Paths for Integer Programming and Links to Artificial Intelligence.". *Computers & Operations Research*, vol. 13: pages 533–549, Jan. 1986.

[33] Ochelska-Mierzejewska, J. *"Data-Centric Business and Applications: Towards Software Development (Volume 4)"*, chapter "Tabu Search Algorithm for Vehicle Routing Problem with Time Windows", pages 117–136. Springer International Publishing, Dez. 2019.

[34] Voudouris, C.; Tsang, E. "Partial Constraint Satisfaction Problems and Guided Local Search". In *Technical Report CSM-250*, Department of Computer Science, University of Essex, Colchester, C04 3SQ, UK, Apr. 1996.

[35] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. "Optimization by Simulated Annealing". *Science*, vol. 220: pages 671–680, Jun. 1983.

[36] Wen-Chyuan, C; Russell, R. A. "Simulated Annealing Metaheuristics for the Vehicle Routing Problem with Time Windows". *Annals of Operations Research*, vol. 63: pages 3–27, Feb. 1996.

[37] "Eiben, A.E.; Smith, J.E.". *Introduction to Evolutionary Computing.* Second Edition. Natural Computing Series. Springer, Berlin, 2015.

[38] Dorigo, M.; Birattari, M.; Stutzle, T. "Ant Colony Optimization". *IEEE Computational Intelligence Magazine*, vol. 1(no. 4): pages 28–39, 2006.

[39] Kennedy, J.; Eberhart, R. "Particle Swarm Optimization". In *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4: pages 1942–1948, 1995.

[40] Wang; K.P.; Huang, L.; Zhou, C.G.; Wei Pang, W. "Particle Swarm Optimization for Traveling Salesman Problem". In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*, vol. 3: pages 1583-1585, 2003.

[41] OpenStreetMap contributors. "Planet Dump retrieved from https://planet.osm.org". `https://www.openstreetmap.org`, 2017. [Online; accessed 01. Oct. 2024].

[42] OpenStreetMap Foundation. "About - OpenStreetMap Foundation". `https://osmfoundation.org/wiki/About`, 2022. [Online; accessed 01. Oct. 2024].

[43] OpenStreetMap Wiki. "Stats - OpenStreetMap Wiki". `https://wiki.openstreetmap.org/wiki/Stats`, 2023. [Online; accessed 01. Oct. 2024].

[44] OpenStreetMap. "Copyright and License - OpenStreetMap". `https://www.openstreetmap.org/copyright/en`, 2024. [Online; accessed 01. Oct. 2024].

[45] Boeing, G. "OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks". *Computers, Environment and Urban Systems*, vol. 65: pages 126–139, 2017.

[46] Jordahl, K.; Van den Bossche, J; Fleischmann, M.; Wasserman, J.; McBride, J.; Gerard, J.; Tratner, J.; Perry, M.; Badaracco, A. G.; Farmer, C.; Hjelle, G. A.; Snow, A. D.; Cochran, M.; Gillies, S.; Culbertson, L.; Bartos, M.; Eubank, N.; maxalbert; Bilogur, A.; Rey, S.; Ren, C.; Arribas-Bel, D.; Wasser, L.; Wolf, L. J.; Journois, M.; Wilson,

J.; Greenhall, A.; Holdgraf, C.; Leblanc, Fi.; Leblanc, Fr. "GeoPandas: Python Tools for geographic Data". `https://github.com/geopandas/geopandas`, 2014. [Online; accessed 01. Oct. 2024].

[47] Hunter, J. D. "Matplotlib: A 2D Graphics Environment". *Computing in Science & Engineering*, vol. 9 (no. 3): pages 90–95, 2007.

[48] Hagberg, A. A.; Schult, D. A.; Swart, P. J. "Exploring Network Structure, Dynamics, and Function using NetworkX". In *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Varoquaux, G.; Vaught, T.; Millman, J. (Eds.), Pasadena, CA, USA, 2008.

[49] Dijkstra, E. W. "A Note on two Problems in Connexion with Graphs". *Numerische Mathematik*, vol. 1 (no. 1): pages 269–271, 1959.

[50] Perron, L.; Furnon, V. "OR-Tools". `https://developers.google.com/optimization/`, 2024. [Online; accessed 01. Oct. 2024].

[51] Kroese, D.; Brereton, T.; Taimre, T.; Botev, Z. "Why the Monte Carlo Method is so important today". *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 6, Nov. 2014.

[52] Bicher, M.; Wastian, M.; Brunmeir, D.; Popper, N. "Review on Monte Carlo Simulation Stopping Rules: How many Samples are really ecinough?". *SNE Simulation Notes Europe*, vol. 32: pages 1–8, Jan. 2022.

[53] Langschwert, M. "master-thesis-pprp". `https://github.com/mlangs/master-thesis-pprp`, 2023. [Online; accessed 07. Oct. 2024].

[54] Stadt Wien. "Favoriten in Zahlen - Statistiken". `https://www.wien.gv.at/statistik/bezirke/favoriten.html`, 2022. [Online; accessed 07. Oct. 2024].

[55] Bundesministerium für Inneres. "Sicherheitsbericht 2021 Kriminalität: Anhang". `https://www.bmi.gv.at/508/`, 2022. [Online; accessed 07. Oct. 2024].

[56] Gilbert, N. "Agent-Based Models". *The Centre for Research in Social Simulation*, Jan. 2007.

[57] Barbati, M; Bruno, G.; Genovese, A. "Applications of Agent-Based Models for Optimization Problems: a Literature Review". *Expert Systems with Applications*, vol. 39(no. 5): pages 6020–6028, Apr. 2012.

A Additional Tables and Figures

Amenity Tag	Frequency	Amenity Tag	Frequency
bench	622	bank	27
waste_basket	430	pub	20
parking	376	fuel	19
bicycle_parking	229	fountain	18
recycling	194	dentist	16
restaurant	150	bts	13
shelter	146	car_wash	13
fast_food	108	waste_disposal	13
cafe	96	community_centre	12
vending_machine	94	social_facility	12
parking_entrance	84	taxi	11
kindergarten	61	bar	10
telephone	61	post_office	9
drinking_water	53	gambling	9
doctors	52	bicycle_rental	9
school	51	parcel_locker	9
charging_station	43	police	7
place_of_worship	41	parking_space	7
post_box	41	driving_school	6
pharmacy	30	brothel	6
toilets	30	childcare	6
atm	28	veterinary	5

Table 10: Part 1 of amenity tags of the map of Vienna's district Favoriten with their associated frequencies.

Amenity Tag	Frequency	Amenity Tag	Frequency
ice_cream	5	scale	1
clock	5	dancing_school	1
public_bookcase	5	swingerclub	1
car_rental	5	training	1
clinic	3	townhall	1
grit_bin	3	lost_and_found_box	1
library	3	studio	1
university	3	bicycle_repair_station	1
cinema	2	food_court	1
theatre	2	photo_booth	1
bbq	2	motorcycle_parking	1
hunting_stand	2	game_feeding	1
public_bath	2	prep_school	1
vacuum_cleaner	2	music_school	1
language_school	2	money_transfer	1
events_venue	2	internet_cafe	1
fire_station	2	hospital	1
marketplace	2	animal_training	1
biergarten	2	dojo	1
casino	1	lockers	1
courthouse	1	stage	1
prison	1	bus_station	1

Table 11: Part 2 of amenity tags of the map of Vienna's district Favoriten with their associated frequencies.

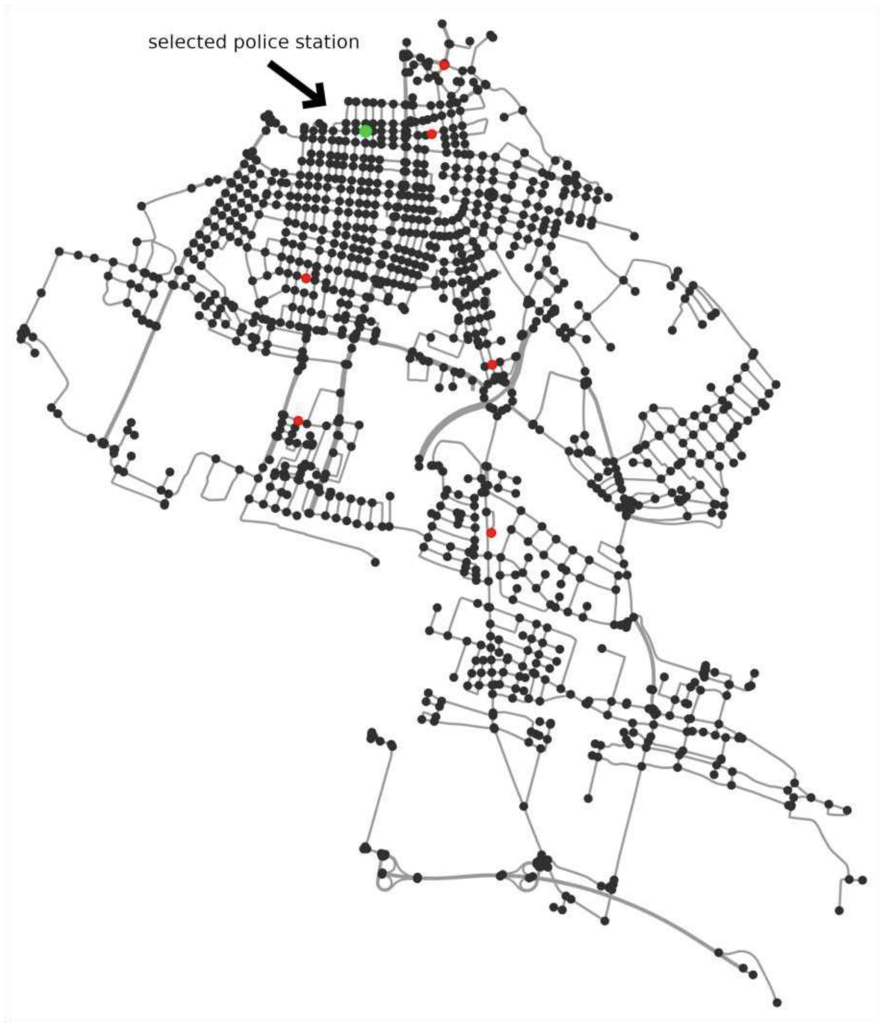


Figure 24: Network graph of the district Favoriten in Vienna with the selected police station marked in green. Other police stations are marked in red. (Map data © OpenStreetMap)

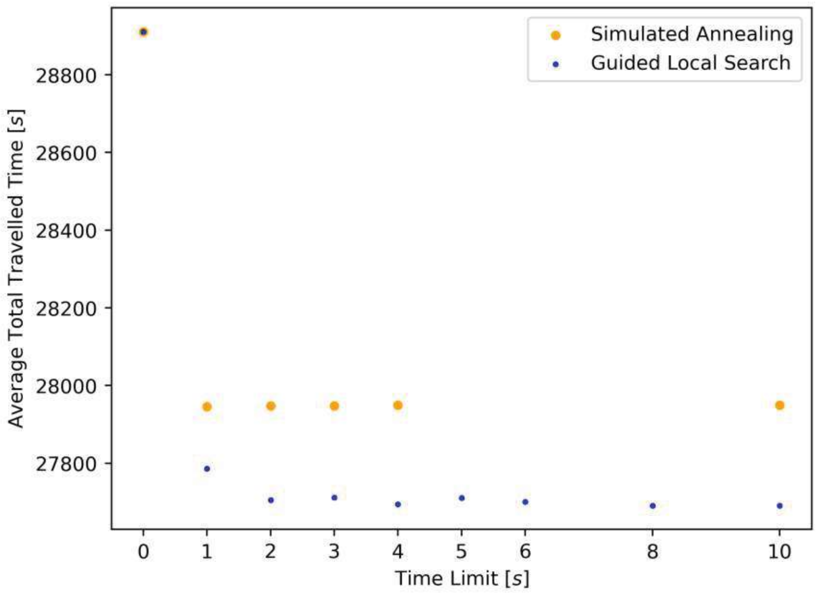
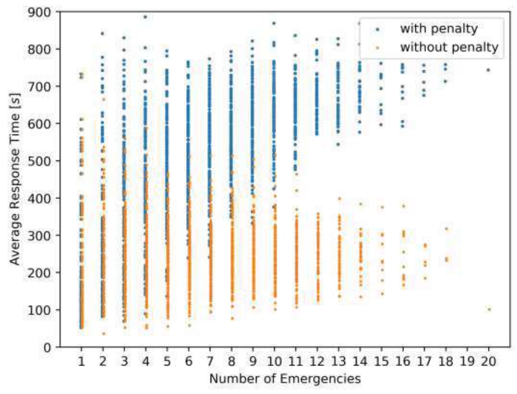
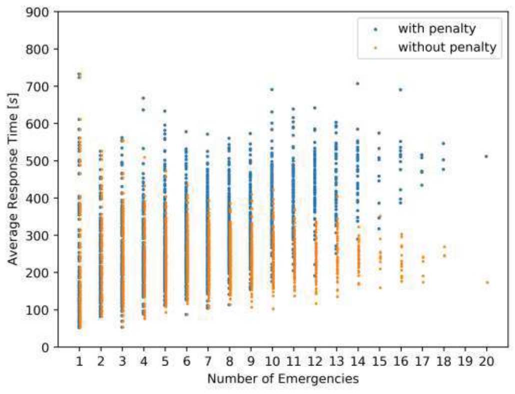


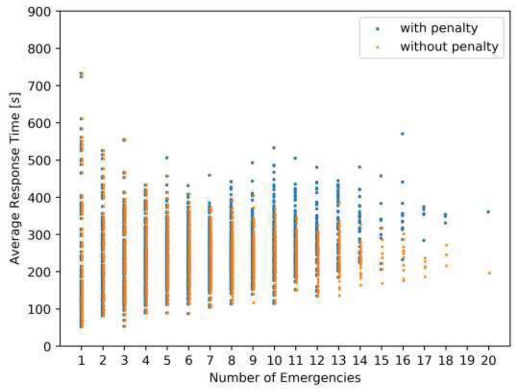
Figure 25: Average total travelled time in seconds for different time limits. The sample sizes are $n = 200$ respectively.



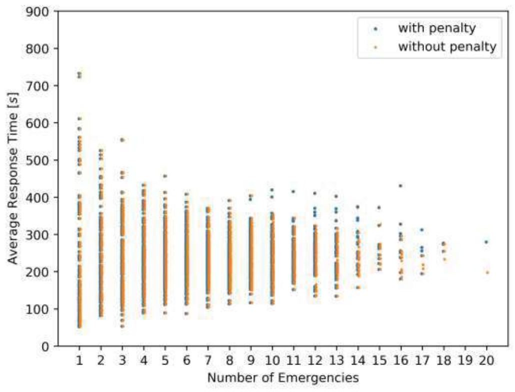
(a) One Police Vehicle



(b) Two Police Vehicles



(c) Three Police Vehicles



(d) Four Police Vehicles

Figure 26: Average response times for each simulation depending on the number of emergencies. Displayed for different numbers of police vehicles. The vehicles have no assigned patrol locations.

B OR-Tools Search Parameters

```
first_solution_strategy: PARALLEL_CHEAPEST_INSERTION
local_search_operators {
  use_relocate: BOOL_TRUE
  use_relocate_pair: BOOL_TRUE
  use_relocate_neighbors: BOOL_FALSE
  use_exchange: BOOL_TRUE
  use_cross: BOOL_TRUE
  use_cross_exchange: BOOL_FALSE
  use_two_opt: BOOL_TRUE
  use_or_opt: BOOL_TRUE
  use_lin_kernighan: BOOL_TRUE
  use_tsp_opt: BOOL_FALSE
  use_make_active: BOOL_TRUE
  use_make_inactive: BOOL_TRUE
  use_make_chain_inactive: BOOL_FALSE
  use_swap_active: BOOL_TRUE
  use_extended_swap_active: BOOL_FALSE
  use_path_lns: BOOL_FALSE
  use_full_path_lns: BOOL_FALSE
  use_tsp_lns: BOOL_FALSE
  use_inactive_lns: BOOL_FALSE
  use_node_pair_swap_active: BOOL_TRUE
  use_relocate_and_make_active: BOOL_FALSE
  use_exchange_pair: BOOL_TRUE
  use_relocate_expensive_chain: BOOL_TRUE
  use_light_relocate_pair: BOOL_TRUE
  use_relocate_subtrip: BOOL_TRUE
  use_exchange_subtrip: BOOL_TRUE
  use_global_cheapest_insertion_path_lns: BOOL_TRUE
  use_local_cheapest_insertion_path_lns: BOOL_TRUE
  use_global_cheapest_insertion_expensive_chain_lns: BOOL_FALSE
  use_local_cheapest_insertion_expensive_chain_lns: BOOL_FALSE
  use_global_cheapest_insertion_close_nodes_lns: BOOL_FALSE
  use_local_cheapest_insertion_close_nodes_lns: BOOL_FALSE
  use_relocate_path_global_cheapest_insertion_insert_unperformed:
  BOOL_TRUE
}
local_search_metaheuristic: GUIDED_LOCAL_SEARCH
guided_local_search_lambda_coefficient: 0.1
solution_limit: 1000000
time_limit {
  seconds: 4
}
lns_time_limit {
  nanos: 100000000
}
savings_neighbors_ratio: 1.0
number_of_solutions_to_collect: 1
savings_arc_coefficient: 1.0
relocate_expensive_chain_num_arcs_to_consider: 4
cheapest_insertion_first_solution_neighbors_ratio: 1.0
log_cost_scaling_factor: 1.0
savings_max_memory_usage_bytes: 6000000000.0
use_cp_sat: BOOL_FALSE
use_cp: BOOL_TRUE
cheapest_insertion_ls_operator_neighbors_ratio: 1.0
heuristic_expensive_chain_lns_num_arcs_to_consider: 4
continuous_scheduling_solver: GLOP
mixed_integer_scheduling_solver: CP_SAT
heuristic_close_nodes_lns_num_nodes: 5
multi_armed_bandit_compound_operator_memory_coefficient: 0.04
multi_armed_bandit_compound_operator_exploration_coefficient:
1000000000000.0
```