TECHNISCHE UNIVERSITÄT WIEN

D I P L O M A R B E I T

Development of an Agent-Based Simulation Model for Process Optimization in Task Planning

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Mathematik

eingereicht von

Alexander Scheichl BSc
Matrikelnummer: 01526669

ausgeführt am Institut für Stochastik und Wirtschaftsmathematik
der Fakultät für Mathematik und Geoinformation der Technischen Universität Wien
(in Zusammenarbeit mit Wiener Netze GmbH)

Betreuung
Betreuer: Senior Scientist DI Dr.techn. Nikolas Popper
Mitwirkung: Projektass.(FWF) Dipl.-Ing. Dr.techn. Martin Bicher
Dipl.-Ing. Dr.techn. Štefan Emrich

Wien, 14.05.2024 _____ _____

(Unterschrift Verfasser) (Unterschrift Betreuer)

Kurzfassung

Diese Diplomarbeit untersucht die Optimierung von Auftragsverplanungsprozessen bei den Wiener Netzen, Österreichs größtem Kombinetzbetreiber für Strom, Gas, Fernwärme und Telekommunikation. Durch die Entwicklung eines in MATLAB implementierten agentenbasierten Modells (ABM) untersucht unsere Studie verschiedene Szenarien, um deren Auswirkungen auf die Effizienz der Auftragsplanung und -ausführung zu evaluieren, wobei der Fokus auf der Struktur der Außendienstmitarbeiter und der räumlichen Aufteilung des Versorgungsgebiets liegt.

Unsere Forschung integriert die Analyse komplexer Systeme durch ABM, wobei das Vehicle Routing Problem (VRP) mit einer heuristischen Technik zur Optimierung der Routenplanung verwendet wird. Dieser Ansatz ermöglicht die Untersuchung der Auftragsverplanung unter verschiedenen Konfigurationen mit dem Ziel, Optimierungspotenziale innerhalb der Mitarbeiterstruktur und der geografischen Aufteilung des Versorgungsgebiets zu identifizieren. Die Ergebnisse unserer Studie sollen den Wiener Netzen praktische Empfehlungen zur Steigerung der betrieblichen Effizienz ihrer Auftragsplanungs- und Dispositionsprozesse geben.

Die wichtigsten Ergebnisse unserer Simulationsexperimente zeigen, dass eine Erhöhung des Prozentsatzes an Heimfahrern - das sind Servicetechniker, die ihren Arbeitstag von zu Hause aus beginnen - von 0 auf 90 Prozent zu einer Steigerung der Effizienz der Auftragsdurchführung um 5,8 Prozent führt. Außerdem könnte der bestehende Rückstand innerhalb von zwei Jahren abgebaut werden. Die Analyse verschiedener Qualifikationsverteilungen der Mitarbeiter zeigt, dass eine Anpassung der Qualifikationen keine signifikanten Auswirkungen hat, solange die Qualifikationsverteilung der Aufträge selbst berücksichtigt wird. Die Variation in der räumlichen Aufteilung des Versorgungsgebiets demonstriert, dass die Gestaltung von Polygongrenzen entlang natürlicher Orientierungspunkte, wie z.B. Straßen, im Vergleich zu streng geometrischen Aufteilungen zu praktikableren und effektiveren Ergebnissen führt.

Abstract

This diploma thesis investigates the optimization of task assignment and scheduling processes within Wiener Netze, Austria's largest combined network operator for electricity, gas, district heating, and telecommunications. By developing an agent-based model (ABM) in MATLAB, our study explores various scenarios to evaluate their impact on the efficiency of task scheduling and execution, particularly focusing on the field service workforce structure and the spatial division of the service area.

Our research integrates the analysis of complex systems through ABM, using the Vehicle Routing Problem (VRP) with a heuristic technique to optimize route planning. This approach allows the investigation of task allocations under different configurations with the aim of identifying optimization potentials within the workforce structure and the geographic division of the service area. The results of our study should provide practical recommendations for Wiener Netze to increase the operational efficiency of their task planning and scheduling processes.

Key findings from our simulation experiments show that increasing the percentage of home drivers - these are service technicians who start their workday from home - from 0% to 90% results in a 5.8% increase in task execution efficiency. Additionally, the existing backlog could be eliminated within two years. The analysis of varying employee qualification distributions shows that adjusting the qualifications has no significant impact as long as the qualification distribution of the tasks themselves is taken into account. The variation in the spatial division of the service area demonstrates that configuring polygon boundaries to follow natural landmarks, such as streets, yields more practical and effective results compared to strictly geometric divisions.

Acknowledgement

I would like to express my deepest gratitude to my supervisors, Nikolas Popper and Martin Bicher, for their invaluable guidance, support, and patience throughout the entire duration of this thesis. With their expertise, encouragement, and constructive feedback they have always helped me in the development of this work.
I would also like to thank Štefan Emrich as well, who introduced me to this topic and was always there for me in the initial phase of this work.
A special thanks goes to my friend Markus, who took the time to carefully proofread this work in the final phase.
I would like to express my sincere appreciation to Wiener Netze for their cooperation and invaluable support during this research. Their provision of data and active participation have greatly enriched the scope and depth of this thesis. I am grateful for the opportunity to work with such a dedicated and professional team whose contributions have been essential to the success of this project.
I am also deeply grateful to my family for their support, love and understanding. Their belief in me has been a constant source of motivation and strength. I am forever grateful to them for enabling me to pursue my academic goals. In addition, I would like to thank my friends for their encouragement and understanding during difficult times. Their companionship and support have brought joy and balance to my life and made the academic journey all the more meaningful.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 14.05.2024

Unterschrift

Contents

1. Introduction

In today's paced and constantly changing world, effective management of resources is crucial for any organization. This is particularly important for utility companies such as Wiener Netze, where the accurate allocation of tasks to employees is vital to ensure uninterrupted service and customer satisfaction. The relationship between the composition of the workforce structure and the geographic division of the service area contribute to the complexity of this already demanding process. Simulation models serve as tools in understanding and navigating the intricacies of such complex systems. They provide visual and analytical representations allowing planners to predict outcomes, identify bottlenecks and assess the effectiveness of various strategies without any real world consequences. This predictive ability is essential for refining task planning processes, ensuring that organizations can adapt quickly and efficiently to changes.

Wiener Netze is Austria's largest combined network operator for electricity, gas, district heating, and telecommunications, serving a large and diverse urban area. Efficient field service management is crucial to ensure consistent service quality and customer satisfaction. This includes the complex task of scheduling and dispatching service technicians to various locations to carry out installations, maintenance work, or emergency repairs. This challenge is compounded by the need to coordinate hundreds of tasks daily across a geographically dispersed service area, influenced by numerous variables such as task urgency, technician availability, and geographic constraints.
The operational efficiency of Wiener Netze is significantly influenced by how effectively tasks are assigned and planned. Inefficient task planning leads to delays, increased operating costs, and lower customer satisfaction. The complexity of this system currently includes the following:

- Resource allocation: Service technicians have different levels of qualifications and are spread across different geographical locations. The challenge is to assign the right task to the right technician while minimizing travel time and costs.

- Geographic dispersion: The service area is widely dispersed, with a mix of dense urban environments and more sparsely populated suburban areas. This dispersion requires

careful planning to optimize routes to minimize travel time and ensure timely task execution.

- Customer satisfaction: Delays in the execution of tasks and inefficient planning have a direct impact on customer satisfaction, especially when promised time windows are not met.
- Legal deadlines: There is a legal obligation to carry out certain tasks such as installations and meter readings within strict deadlines set by energy regulations.

Research Question and Aim of the Thesis

The core research question of this thesis is: How can the operational efficiency of task planning and execution at Wiener Netze be optimized through changes in the field service workforce structure and the geographical division of the service area? This question addresses the critical challenge of deploying field service agents more effectively by optimizing task assignments and route planning based on historical data and operating conditions.

The main objective of this thesis is to develop and validate an agent-based simulation model that can significantly improve the operational efficiency of field service activities at Wiener Netze. The model will simulate real-world operations to identify bottlenecks and inefficiencies in the current system and to test various strategies to optimize task scheduling and resource allocation. Specific aims include:

- Develop a detailed simulation framework: Create a flexible simulation environment that can model the complex interactions between tasks, service technicians, and geographic areas. This framework will use ABM to represent individual agents (technicians, tasks) with different characteristics and decision-making processes.
- Analyze workforce structure and geographic configuration: The aim is to investigate how different configurations of field service agents and the spatial division of the service area affect the efficiency of task allocation. This analysis will help to understand the impact of strategic decisions on operational performance.
- Validation of the model with empirical data: Ensuring the practical relevance and accuracy of the model by validating its results against historical data. This validation involves comparing the simulation results with records of actual task execution to assess the predictive accuracy and operational value of the model.

By achieving these goals, this thesis will not only contribute to the academic field of operations research and system simulation, but will also provide Wiener Netze with a powerful

tool to improve their service delivery and operational flexibility. It is expected that the results of this research will lead to significant improvements in resource utilization, operational efficiency, and customer service, ultimately supporting Wiener Netze's mission to provide reliable utility services throughout its service area.

In the first chapter, we provide an overview of the underlying concepts of our work and explain the research problem in detail. In the following two chapters we describe our simulation model in detail, and in Chapter 5 we discuss the experiments conducted and document the results of the simulation. In the last chapter, we give an outlook on possible future research opportunities and how the model can be extended to obtain more accurate results and predictions regarding the task allocation process.

2. Modeling Approach

This chapter deals with the modeling techniques and problems which are fundamental for our work. First, we will describe our research problem in detail, and afterwards we will elaborate on agent-based modeling (ABM), the vehicle routing problem (VRP) and the point in polygon problem.

2.1. Detailed Description of the Problem and Modeling Assumptions

In this section, we will describe all relevant processes taking place at Wiener Netze, and provide an explanation on the assumptions and simplifications employed in the modeling process which we will explain in detail in the next chapter.

2.1.1. General Work Processes in the Field Service

The service area of Wiener Netze is divided into polygons. Currently, there are 64 polygons in Vienna and its surroundings, with an additional polygon in the Ybbstal region.
The working days of service technicians span from Monday to Friday (except holidays), with working hours from 7 am to 3 pm. The workday commences at the headquarters in Erdberg, with five field service agents starting at the southern branch in Oeynhausen/Traiskirchen. There is an ongoing pilot project called “home drivers” (translated from German: “Heimfahrer”) since 2020. The service technicians carry a mobile inventory in their vehicles, and they start their workday on the first working day of each week at the Erdberg headquarters where they pick up the material they need for the rest of the week, when they start from their home. Participation in the “home drivers” program is restricted to those whose residence is within the service area.
Service technicians possess qualification profiles, currently classified as Q2 and Q3. Due to an ongoing internal training program there already is an additional qualification level, namely Q3 light. Depending on the qualification field service agents can execute certain tasks which require different skills. An overview of the required qualification for each service product is given in Tables 3.2 and 3.3. The service product new installations is a

special case regarding qualifications of service technicians, a part of these tasks are mass installations, that are at least 10 new meter installations at one location, for example a residential complex. These mass installations can be executed by Q3 light employees in presence with at least one Q3 technician. Single new installations have to be done by Q3 employees.

Field service representatives are assigned to multiple polygons, each with a specified priority. Three priorities are available, but currently only two are utilized. Vice versa there are multiple service technicians assigned to one polygon.

Limitations, Assumptions and Simplifications

The simulation is limited to Vienna and its surrounding areas. The reason for this is that the Ybbstal region has a large spatial distance to the rest of the service area and there are only few tasks to execute each year.

An additional simplification of the model concerns the pilot project "home drivers", we assume that the starting point for service technicians which are "home drivers" is at home every day.

Regarding mass installations, the requirement of the presence of at least one Q3 technician is not implemented in the model. That means it may happen that no Q3 employee is scheduled for these tasks, but also that more than one Q3 employee is present at the location.

Due to insufficient data, the accurate assignment of polygons to field service agents is not implemented, instead we allocate all polygons to each employee with the highest priority.

2.1.2. Rules for Scheduling of Tasks

Each task is assigned a specific appointment, dependent on accessibility of the meter, either time-limited (notification letter to the customer with a time slot) or day-limited. Weekly and daily jobs are executed by a software ([22]) to plan, schedule and optimize the tasks.

Key rules include a predefined execution period and prioritization of appointments with time slots over tasks without an appointment window.

In the weekly job run, the scheduling begins three weeks in the future, and the next four weeks are planned. This job run is mainly used for planning tasks which require a customer notification letter.

For the daily job run, planning and optimization cover the next 10 days. Packages (collections of tasks for a single field service agent on a specific day) are completed for the upcoming deadline day (currently set at 2 days before the planned execution date). Within

the deadline no automatic planning and scheduling is allowed. Exceptions to the deadline are made for legal deadlines of certain tasks.
If necessary, manual scheduling is conducted by office employees.
Additionally, it may happen for various reasons that an already planned task is cancelled, then this task is removed from the package of the affected field service agent.
Task scheduling for service technicians commencing their workday from home is done in a way to ensure they begin the workday at 7 am at home and return at 3 pm. Technicians starting at the headquarters or the branch are assigned their initial task to begin at 7:45 am and the last task is scheduled to guarantee their return to Erdberg at 3 pm.

Limitations, Assumptions and Simplifications

It is not ascertainable from the data whether a task requires a customer notification letter or not, i.e. the accessibility of the meter. Currently, 70% of all meters are accessible at any given time, with an increasing trend. Since the weekly job run is rarely used anyway, we restrict on the daily job run (which means the planning and scheduling of the next 10 days).
The deadline exceptions for legal deadlines, manual scheduling and cancelled tasks are not considered in the model.
Since the execution times given in the data are quite generous, i.e. tasks are often completed much quicker in reality, we introduce a reality reduction parameter which serves to scale down the provided execution times, yielding more realistic outcomes of the model. This parameter is determined by calibration, the process is explained in Section 4.3.

2.1.3. RSVK Meter Readings

Reading the current transformer meters for major customers involves fulfilling legal tasks at the beginning of each month, which must be completed by the 10th working day of the month. Field service agents are blocked for certain time slots to execute these tasks during the first 10 working days, allocating a maximum of 5 hours per day.
Service technicians receive all the tasks assigned to their polygon and autonomously organize the sequence in which they will execute the tasks. Frequently, individual scheduling arrangements, such as those with embassies, are necessary and are managed by the technicians themselves as well.

Limitations, Assumptions and Simplifications

It is not feasible to represent this aspect of the system in the model since there is hardly any automatic planning and scheduling. The simulation blocks periods in which no task

can be scheduled, aligning with the blocked time slots in the real system.

We will model the described system using an agent-based approach integrated with the vehicle routing problem (VRP) to effectively model the task assignment planning and scheduling processes.
Before specifying the model in detail, we provide background information on the modeling strategy and introduce general concepts for solving VRPs.

2.2. Agent-Based Modeling (ABM)

Agent-based modeling (ABM) serves as an important technique that allows us to analyze and understand complex systems by simulating autonomous agents within these systems. ABM is particularly useful to understand emergent behaviors that arise from the interactions between individual agents, making it applicable to a wide range of domains such as ecology, economics and social sciences. This approach allows researchers to explore scenarios and test interventions in a controlled virtual environment and gain insights into the dynamics and potential outcomes of real-world systems.
The key aspect of ABM lies in the flexibility and adaptability of agents who can take individual actions to achieve their objectives. Due to this property complex behaviors and interactions within a system can be represented. ABM becomes most valuable when traditional analytical models are insufficient to capture the nonlinearities and complexities of system dynamics, providing a more detailed and interactive analysis method.
ABMs consist of three core components: agents, environments, and rules of interaction. Agents are the individual entities that have their own unique behaviors, goals, and decision-making capabilities. The environment acts as the context in which agents operate and interact with each other, potentially influencing their behavior and vice versa. Rules of interaction define how agents interact with each other and with their environment, governing the dynamics of the system.
Agents possess several key characteristics, including autonomy, social ability, reactivity, and proactiveness. Autonomy allows agents to operate without direct human intervention, making independent decisions based on their programming and objectives.

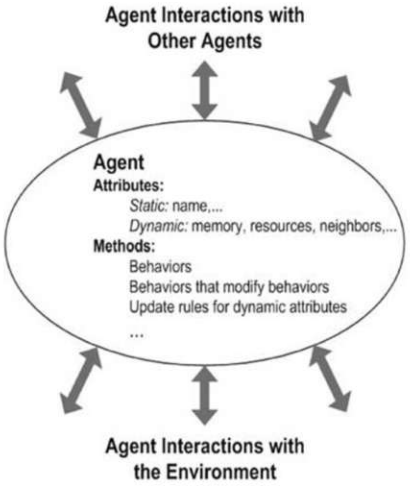


Figure 2.1.: A typical agent. Source: [16] (p. 154)

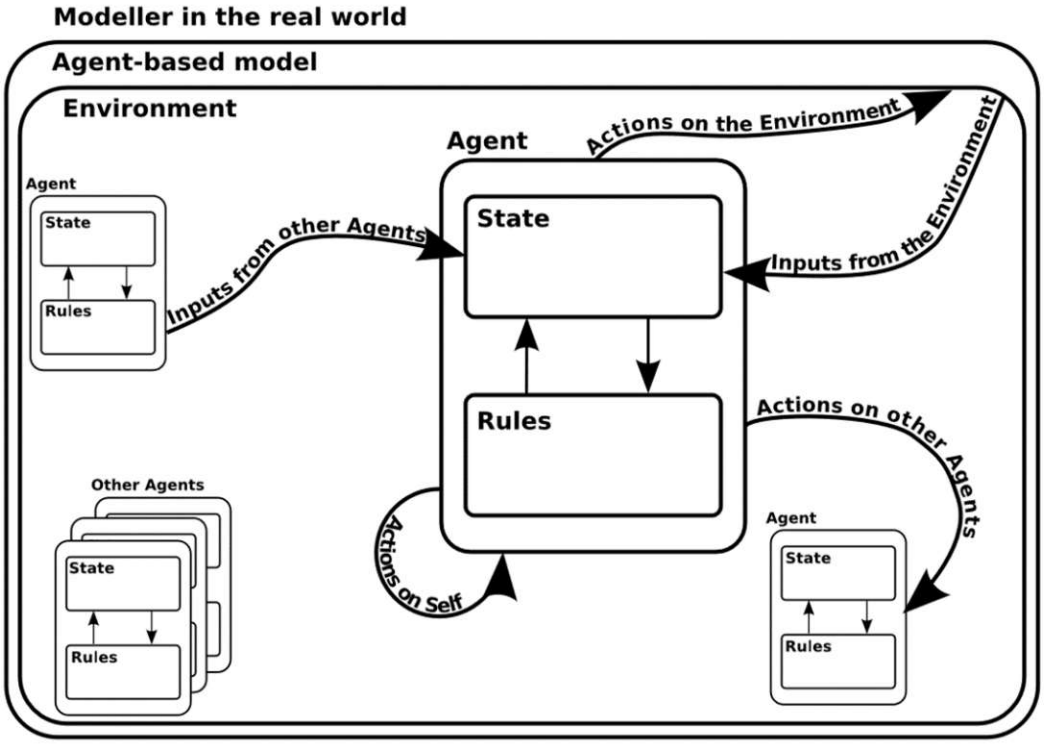


Figure 2.2.: Structure of an agent-based model. Source: [5] (p. 58)

Social ability enables agents to communicate and interact with other agents, forming the basis for complex social structures and behaviors within the model. Reactivity refers to an agent's ability to perceive and respond to changes in its environment. Proactiveness describes the capacity of agents to take initiative and perform goal-directed behaviors. [15], [16]

Agent-based modeling (ABM) provides us with a fairly different approach compared to other simulation methods such as discrete event (DE), system dynamics (SD), and dynamic systems (DS). Each modeling technique has its own strength depending on the field of application. ABM excels in examining the phenomena that result from the interactions between individual agents, making it particularly suitable for systems where individual behavior has a significant impact on the outcome. This micro-level focus contrasts from SDs macro-level approach, which uses aggregate variables and feedback loops to understand system behavior over time, often represented through flow diagrams and differential equations as discussed in [2]. DE, on the other hand, focuses on the sequencing and timing of discrete events, making it suitable for operations research and logistics. DS approaches, similar to SD, employ equations to model changes in system states over time, focusing on

continuous systems dynamics.

Each method has its domain of applicability: ABM for detailed behavioral simulations, DE for event-driven systems, SD for understanding feedback and accumulative effects in large systems, and DS for systems described by continuous dynamics. The choice between these methods depends on the specific research questions, the level of detail required, and the nature of the system being modeled.

In the context of this thesis, the choice of agent-based modeling (ABM) over methods such as discrete events (DE), system dynamics (SD) or dynamic systems (DS) is motivated by several crucial factors. First, ABM offers the key advantage of being able to simulate the behavior and interactions of autonomous agents representing service technicians and tasks. The ability to model individual decisions and their outcomes is crucial for analyzing the operational dynamics within Wiener Netze's task management system. In contrast to DE, which focuses primarily on the sequence and timing of events within a system, ABM enables a more nuanced simulation of workflows and interactions between agents.
Second, while SD and DS provide powerful tools for modeling aggregate data and system-level dynamics through feedback loops and differential equations, they do not adequately capture the stochastic nature and diversity of agent behavior and its impact on the system as a whole. ABM, on the other hand, enables detailed scenario testing and the study of how individual actions and interactions lead to new behaviors within the task allocation processes. This approach is particularly useful for understanding how changes in individual agent characteristics or strategies can impact the efficiency and effectiveness of the overall system, and provides an analytical framework well suited to the complex, adaptive system that Wiener Netze's operational challenges represent.

Now we will describe the vehicle routing problem in detail which is integrated in our ABM framework.

2.3. The Vehicle Routing Problem (VRP)

The vehicle routing problem (VRP) is a fundamental problem in operations research and logistics, aiming to optimize routes for a fleet of vehicles delivering goods or services to a set of customers. It first appeared in [6] as the "Truck Dispatching Problem" and extends the traveling salesman problem (TSP) by considering multiple vehicles originating from a central depot. The VRP is essential for minimizing transportation costs and improving logistical efficiency. Various VRP types address specific constraints and real-world systems, such as the capacitated VRP (CVRP), where vehicles have a maximum carrying capacity,

and the VRP with time windows (VRPTW), requiring deliveries within specific time frames. The TSP is intrinsically related to the VRP as its special case when the fleet size is one. The complexities introduced by multiple vehicles, capacity constraints, and customer service times make the VRP more challenging and applicable to a wider range of practical situations. Heuristics, metaheuristics, and exact algorithms have been developed to deal with the VRP's computational complexity, offering solutions that, while not always optimal, are efficient and practical for real-world applications.

A comprehensive study of various forms of the vehicle routing problem and both exact and heuristic methods is given in [25]. The book also discusses the practical implications and difficulties of VRP, providing a detailed analysis of the algorithms, models, and real-world case studies related to efficient routing and scheduling in logistics and distribution systems.

We will focus on the vehicle routing problem with time windows (VRPTW) since this is the VRP type we are dealing with. It includes constraints on the time within which deliveries or services must be performed. Solving the VRPTW requires sophisticated approaches that can effectively handle its complexities. We can categorize the solution methods into three main approaches: Exact algorithms, heuristic methods, and metaheuristic techniques. An overview of solution approaches and the corresponding publications can be found in [7], p.19.

2.3.1. Exact Algorithms

Exact algorithms aim to find the optimal solution to the VRPTW by exploring all possible combinations and configurations. These methods, such as branch-and-bound, branch-and-cut, and integer programming formulations, guarantee the discovery of the best possible solution but are often computationally infeasible for large instances due to the exponential growth of the solution space.

The branch-and-cut method starts with the linear relaxation of an integer program, where the integrality constraints on the variables are dropped, providing a lower bound to the optimal value of the integer program. The branch-and-cut method then solves this linear relaxation. If the solution is not integral, it branches by creating new linear programs that impose integer constraints on the fractional variables.

This process is iterative, splitting the problem into subproblems and solving their linear relaxations to find bounds. Each node in the search tree represents a subproblem with tighter bounds, and the tree is explored systematically to find the optimal integer solution. The branch-and-cut method is powerful because it combines the depth-first search of branch-and-bound with the power of cutting planes, which are used to eliminate fractional solutions by adding new constraints, thus tightening the linear relaxation. [25]

2.3.2. Heuristic Approaches

Heuristic methods provide feasible solutions more quickly by applying rules and intuition to guide the search process. These solutions may not be optimal but are often close to the best possible and are achieved in a fraction of the time required by exact algorithms. Common heuristic approaches include Clarke and Wright's savings algorithm, the sweep algorithm, and various insertion heuristics.

We will shortly describe two methods proposed by Solomon [23].

Savings Heuristics: This method extends the savings heuristic originally proposed for the routing problem by Clarke and Wright [4]. The approach begins with distinct routes in which each customer is served by a dedicated vehicle and then looks for cost savings that can be achieved by combining routes in a way that respects time windows, vehicle arrival times at the depot, and capacity constraints.

Time-Oriented Sweep Heuristic: This heuristic is part of a broader class of methods that decompose the VRPTW into a clustering phase and a scheduling phase. Initially, customers are assigned to vehicles based on a sweep heuristic, which is then followed by the creation of a one-vehicle schedule for the customers in this cluster using a tour-building heuristic.

Sequential and Parallel Insertion Heuristics

Sequential and parallel construction heuristics form the foundation for generating initial solutions in the VRPTW. Sequential construction builds routes one at a time, inserting nodes (customers) into a route until no further feasible insertions are possible without violating constraints. In contrast, parallel construction builds several routes simultaneously, where the number of vehicles is specified from the outset. Solomon's insertion heuristics [23] represent a cornerstone of sequential construction, methodically expanding routes by inserting the most suitable customer at each iteration based on cost and time window constraints. An overview of the VRPTW and common insertion heuristics is given in [14].

Sequential Insertion Heuristic

Solomon [23] describes an insertion heuristic approach which plays a crucial role in creating initial solutions for the VRPTW. This method starts with a seed customer and iteratively adds unrouted customers to the route, prioritizing insertions that minimize additional travel distance and time delay, while adhering to the time window constraints. The success of this approach lies in its simplicity and effectiveness in quickly generating feasible routes that respect the problem's constraints.

Parallel Insertion Heuristic

Potvin and Rousseau [18] expanded on the concept of insertion heuristics by introducing a parallel route building framework. This method improves the ability to handle vehicle routing problems with time windows by initializing routes with seed customers and expanding them simultaneously. Their approach optimizes the allocation of customers across routes taking into account a so called regret measure, which quantifies the cost increase of delaying a customers insertion into its most suitable route.

Campbell and Savelsbergh [3] focus on enhancing the efficiency of insertion heuristics for the VRPTW, taking into account complicating constraints that increase the problem's complexity. Their algorithm aims to maintain a balanced time complexity of $O(n^3)$ or, in more complex cases, extend it slightly to $O(n^3 \log n)$, despite the introduction of such constraints. The main innovation in their approach lies in the handling of these complicating factors, such as time windows, variable delivery quantities, and multiple routes per vehicle, without significantly compromising computational efficiency.

The algorithm works by iteratively inserting unrouted customers into partially constructed routes. It selects insertion points that minimize travel costs while adhering to constraints like time windows and vehicle capacity. The key to maintaining efficiency is the method's ability to assess the feasibility and profitability of insertions rapidly, relying on updated information about the current solution state to make decisions in constant or logarithmic time. Algorithm 1 presents their basic insertion heuristic, which is the foundation of our routing algorithm.

Potvin and Rousseau's parallel route building algorithm [18] also constructs multiple routes simultaneously. They primarily focus on the straightforward application of insertion heuristics to the VRPTW without the same depth of consideration for the wide range of complicating constraints addressed by Campbell and Savelsbergh. Potvin and Rousseau introduced a method that efficiently builds routes in parallel by evaluating the best customer insertion points across all routes under consideration, optimizing the process by using a regret-based measure to prioritize insertions.

Solution Improvement Methods

After constructing initial solutions, solution improvement heuristics are employed to refine them. The goal is to reduce the overall cost by making changes to the composition of routes without violating the constraints of the VRPTW. Common techniques include the 2-opt and 3-opt exchanges, which reorganize the sequence of customer visits within routes. Furthermore, Potvin and Rousseau [19] introduced an exchange heuristic specifically designed for the VRPTW, demonstrating the potential for significant enhancements in solution quality through strategic modifications of existing routes.

Algorithm 1 Basic Insertion Heuristic

```
1: N ← set of unassigned customers
2: R ← set of routes; always contains the empty route; initially contains only the empty
   route
3: while N ≠ ∅ do
4:     p* ← −∞
5:     for j ∈ N do
6:         for r ∈ R do
7:             for (i − 1, i) ∈ r do
8:                 if FEASIBLE(i, j) and PROFIT(i, j) > p* then
9:                     r* ← r
10:                    i* ← i
11:                    j* ← j
12:                    p* ← PROFIT(i, j)
13:                end if
14:            end for
15:        end for
16:    end for
17:    INSERT(i*, j*)
18:    N ← N \ {j*}
19:    UPDATE(r*)
20: end while
```

2.3.3. Metaheuristic Techniques

Metaheuristic techniques are strategies that enhance the effectiveness of search processes in exploring the solution space. They include genetic algorithms, simulated annealing, tabu search, and ant colony optimization. These methods have the ability to escape local optima and explore a broader range of potential solutions. [25] and [7] explore the application of metaheuristics to the VRPTW, showing their adaptability and efficiency in finding high-quality solutions for complex and large-scale problems.

Genetic Algorithms (GA): GAs simulate the process of natural evolution, using operations such as selection, crossover, and mutation to evolve a population of solutions towards better fitness.

Tabu Search (TS): TS is characterized by using memory structures that store previously visited solutions, typically called the tabu list, to prevent cycling back to them and encourage the exploration of new areas in the solution space.

Simulated Annealing (SA): This method is designed to escape local optima by allowing for occasional moves to worse solutions, inspired by the annealing process in metallurgy (physical process of heating and then slowly cooling a material to reach a state of minimum energy). It is a probabilistic technique that can find a global optimum by cooling the system

and reducing the likelihood of such moves over time.
Ant Colony Optimization (ACO): ACO is a probabilistic technique that takes inspiration from the behavior of ants searching for food. By simulating the pheromone trail laying and following behavior of ants, ACO can effectively find optimal paths through the solution space.

In our model, we implement a parallel insertion heuristic, more precisely a modification of Algorithm 1. Exact algorithms are not computationally feasible in many cases, and due to the huge solution space of our problem, metaheuristic techniques are not feasible either. Therefore we choose a heuristic approach, and since the number of vehicles, i.e. the number of service technicians, is fixed on each day, all available routes should be created simultaneously based on the locations of technicians and tasks. Hence, we prefer a parallel heuristic instead of a sequential heuristic.

2.3.4. Routing Methods

In the context of optimizing route planning for field service operations, defining an effective routing problem is crucial. Specifically, it is about finding the path that allows the fastest travel time between tasks, which is crucial for improving the efficiency of task planning and execution. Various methods can be used to estimate and optimize these routes, of which Contraction Hierarchies (CH), Landmarks (LM), A* search and Dijkstra's algorithm are particularly noteworthy (see [10]). We select the Contraction Hierarchies (CH) method in our work for performance reasons. CH significantly speeds up route finding by reducing the size of the road network during preprocessing, thus enabling faster query times during actual route planning.

Before describing our agent-based model with integrated VRP we will introduce the third main concept used, namely the point in polygon problem. It is crucial for the determination in which geographical areas the tasks and employees are located.

2.4. The Point in Polygon Problem

The service area of Wiener Netze is divided into polygons, and in order to assign the tasks to the service technicians in a meaningful way, it is necessary to determine in which polygon a task with certain geographical coordinates is located. Additionally, with the concept of grid cells which we will introduce later (Chapter 3.1) we need to identify the exact grid cell in which the tasks and field service agents are located. Therefore, a key problem in our research is to determine whether a given point lies inside or outside an arbitrary closed

polygon. This problem is not trivial, and an inefficient algorithm has a major impact on computing time. In this section we will discuss the two concepts known for solving this problem and derive an algorithm similar to the one implemented in MATLAB in the function `inpolygon`. We will outline the key facts presented in [13] and give a summary how to obtain the final Algorithm 2 (Algorithm 6 in [13], p.140).

The first concept is the even-odd or parity rule. From our given point R we draw a line to some other point S which is guaranteed to lie outside the polygon P represented by n points $P_0, P_1, ..., P_{n-1}, P_n = P_0$. If the line $\overline{RS}$ crosses the edges $e_i = \overline{P_i P_{i+1}}$ of the polygon an odd number of times, R is inside P, otherwise it is outside.

The second concept is based on the winding number of R with respect to P, which is the number of revolutions made around R while traveling once along P. Then R is inside the polygon if the winding number is nonzero. If we let the point R be inside P and if the winding number is odd, we obtain the same result as with the even-odd rule. Therefore both concepts are equivalent and we will focus on the winding number since this is the more general approach.

The winding number $\omega(R, C)$ of a point R with respect to a closed curve $C(t) = (x(t), y(t))^\top$, $t \in [a, b], C(a) = C(b)$, is the number of revolutions made around R while traveling once along C. Here we assume that R does not lie on the curve $C(t), t \in [a, b]$. If there exists $\tilde{t} \in [a, b]$ with $C(\tilde{t}) = R$, the winding number $\omega(R, C)$ is undefined. Otherwise we calculate it by integrating the differential of the angle $\varphi(t)$ between the edge $\overline{RC(t)}$ and the positive horizontal axis through R (see Figure 2.3 (a)). Since $C(t)$ is a closed curve, this always yields $\omega \cdot 2\pi$, where $\omega \in \mathbb{Z}$ is the winding number.

Without loss of generality we assume $R = (0, 0)$ so that $\varphi(t) = \arctan(y(t)/x(t))$ and

$$\omega(R, C) = \frac{1}{2\pi} \int_a^b d\varphi(t) = \frac{1}{2\pi} \int_a^b \frac{d\varphi}{dt}(t)dt = \frac{1}{2\pi} \int_a^b \frac{y'(t)x(t) - y(t)x'(t)}{x(t)^2 + y(t)^2} dt, \tag{2.1}$$

where we used the chain and quotient rule as well as $\arctan'(x) = 1/(1 + x^2)$ in the last equality.

Before we proceed we need further computations. Let $R = (0, 0)^\top, P = (P_x, P_y)^\top$ and $Q = (Q_x, Q_y)^\top$ be the vertices of a planar triangle and α, β, γ the angles at R, P and Q, respectively. Then we obtain by the definition of the dot product

$$P \cdot Q = \|P\|\|Q\| \cos \alpha,$$

hence

$$\cos \alpha = \frac{P \cdot Q}{\|P\|\|Q\|}. \tag{2.2}$$

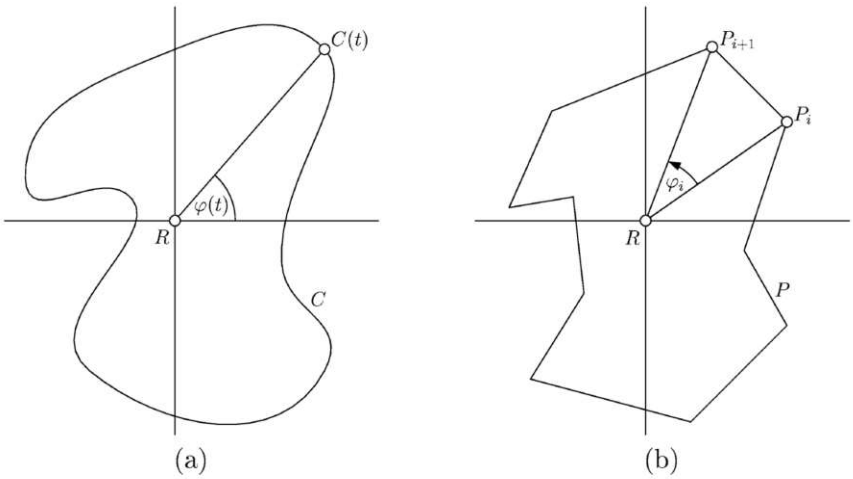


Figure 2.3.: (a) The continuous angle $\varphi(t)$ for curves. (b) The discrete signed angle φ_i for polygons. Source: [13] (p.133)

Using geometric properties (dot product, determinant, vector analysis) we calculate

$$(P-Q)\cdot P = \|P-Q\|\|P\|\cos\beta,$$
$$|D| = \begin{vmatrix}(P-Q)_x & P_x\\ (P-Q)_y & P_y\end{vmatrix} = \|P-Q\|\|P\|\sin\beta,$$

where $D = P_xQ_y - Q_xP_y$. Rearranging these equations and using the definition of the cotangent yields

$$\cot\beta = \frac{(P-Q)\cdot P}{|D|}, \quad \cot\gamma = \frac{(Q-P)\cdot Q}{|D|}. \tag{2.3}$$

For the linear curve $(x(t), y(t))^\top = tQ + (1-t)P, t \in [0,1]$, we then obtain the following equations:

$$\begin{aligned}\int_0^1 \frac{y'(t)x(t) - y(t)x'(t)}{x(t)^2 + y(t)^2}dt &= \int_0^1 \frac{D}{t^2(Q-P)\cdot(Q-P) + 2t(Q-P)\cdot P + P\cdot P}dt\\ &= \arctan\frac{(Q-P)\cdot Q}{D} + \arctan\frac{(P-Q)\cdot P}{D}\\ &\overset{(2.3)}{=} \operatorname{sign}(D)(\arctan\cot\gamma + \arctan\cot\beta)\\ &\overset{(*)}{=} \operatorname{sign}(D)(\pi-\gamma-\beta)\\ &= \operatorname{sign}(D)\alpha,\end{aligned} \tag{2.4}$$

where we used $\arctan(\cot x) = \pi/2 - x, x \in (0, \pi)$ in equation $(*)$.
Now, we view a closed polygon P represented as an array of n points $P_0, P_1, ..., P_{n-1}, P_n = P_0$ as a piecewise linear curve $t \mapsto (x_i(t-i), y_i(t-i))^\top, t \in [i, i+1]$, with $(x_i(t), y_i(t))^\top = tP_{i+1} + (1-t)P_i$. We then obtain

$$\begin{aligned}\omega(R,P) &\overset{(2.1)}{=} \frac{1}{2\pi}\sum_{i=0}^{n-1}\int_0^1 \frac{y_i'(t)x_i(t) - y_i(t)x_i'(t)}{x_i(t)^2 + y_i(t)^2}dt \\ &\overset{(*)}{=} \frac{1}{2\pi}\sum_{i=0}^{n-1}\arccos\frac{P_i \cdot P_{i+1}}{\|P_i\|\|P_{i+1}\|} \cdot \operatorname{sign}\begin{vmatrix} P_i^x & P_{i+1}^x \\ P_i^y & P_{i+1}^y \end{vmatrix} \\ &= \frac{1}{2\pi}\sum_{i=0}^{n-1}\varphi_i, \end{aligned} \tag{2.5}$$

where we used equations (2.2) and (2.4) in equality $(*)$ and φ_i denotes the signed angle between the edges $\overline{RP_i}$ and $\overline{RP_{i+1}}$ (see Figure 2.3 (b)).

For the next steps we give a short summary, a detailed explanation and the pseudo-codes of the algorithms can be found in [13].

We classify each vertex of the polygon by the number of the quadrant in which it is located with respect to the point R. Then we calculate the winding number using equation (2.5) for angles and the quadrant classifications for vertices. We simplify this calculation by considering only quarter-revolutions and adjust the angles based on our quadrant classifications. Next, we sum up the quarter angles to count the number of quarter ccw-revolutions around the point R. Introducing adjusted quarter angles and summing them to count the number of quarter ccw-revolutions leads to a modification of the algorithm. We further optimize the algorithm by considering differences between adjusted and unadjusted quarter angles, which results in improved performance. Then the algorithm is converted into a ray-crossing method, which counts the number of crossings of edges with a horizontal ray originating from the point. The resulting algorithm is able to handle degenerate cases, such as points coinciding with polygon vertices or lying on polygon edges.
For many applications the discussed algorithms are sufficient but may become a bottleneck in applications where they are frequently called (which is exactly the case is our work). Modification of these algorithms leads to two efficient versions: the efficient standard algorithm (Algorithm 2) and the efficient boundary algorithm (Algorithm 7 in [13], p.141), the latter handling the special case of the point lying on the polygon boundary.
The three main parts for potential improvement are the determinant function, henceforth referred to as *det*, quadrant classification, and the determination of the winding number. While the *det* function's calculation cannot be significantly accelerated, the quadrant clas-

Algorithm 2 Efficient Standard Algorithm ([13], p.140)

```
1: ω = 0
2: for i = 0 to n − 1 do
3:     if crossing then
4:         if P_i^x ≥ R^x then
5:             if P_{i+1}^x > R^x then
6:                 modify_ω
7:             else
8:                 if right_crossing then
9:                     modify_ω
10:                end if
11:            end if
12:        else
13:            if P_{i+1}^x > R^x then
14:                if right_crossing then
15:                    modify_ω
16:                end if
17:            end if
18:        end if
19:    end if
20: end for
21: return ω
```

sification is enhanced to reduce the average number of comparisons per vertex. A decision tree is introduced, which accelerates the basic algorithms by more than 30%.

The algorithm's restructuring involves combining two loops and eliminating the array which stores the quadrant numbers as we can process them on the fly. Memory usage and execution time are reduced, leading to a simplification of the algorithm. Further optimization is achieved by handling cases where the winding number changes directly, without explicitly using quadrant numbers. Edges that contribute to a change in the winding number are identified based on their endpoints' positions relative to a horizontal line through the point.

A subsequent improvement involves avoiding the costly computation of the determinant whenever possible. This results in the efficient standard algorithm (Algorithm 2), where the following macros are used:

$$\begin{aligned} \text{crossing:} &\quad (P_i^y < R^y) \neq (P_{i+1}^y < R^y), \\ \text{right\_crossing:} &\quad (\det(i) > 0) = (P_{i+1}^y > P_i^y), \\ \text{modify\_}\omega: &\quad \omega = \omega + 2 * (P_{i+1}^y > P_i^y) - 1. \end{aligned}$$

Finally, the efficient boundary algorithm (Algorithm 7 in [13], p.141) is developed to handle special cases. Edge coincidences are detected using a modified *det* function in the right_crossing condition. The special case of the point lying on a horizontal edge is addressed, along with the case of the point coinciding with one of the vertices.

3. Model Specification

The model description follows the ODD (Overview, Design concepts, Details) protocol for describing individual- and agent-based models. [11], [12]

3.1. Overview

This section provides an overall overview of the purpose and structure of the model, including the state variables and main processes.

3.1.1. Purpose

The purpose of the model is to find ways to optimize the task assignment planning and scheduling process at Wiener Netze by simulating various scenarios. A special focus is on various configurations of the field service workforce structure and different spatial divisions of the service area. The model is designed to explore a range of scenarios to determine optimal strategies for a grid operator's task allocation and execution process. Moreover, it abstracts from the detailed demographic and geographical conditions and decides for a simplified representation that allows a focused analysis of operational dynamics. In order to evaluate the results of the simulation, the model must provide a list of tasks executed by each service technician on each day, the total execution time of all tasks completed on each day as well as the backlog of task.

3.1.2. Entities, State Variables and Scales

The model consists of four entities, of which two are agents/individuals (*Tasks* and *Employees*) and two are spatial units (*GridCell* and *Polygon*). An overview of the agent's state variables and initial values is given in Table 3.1.

Agents of the entity *Tasks* are characterized by the state variables *ReferenceNumber*, *TaskNumber*, *ID*, *MassInstallation*, *Skill*, *ExecutionTime*, *GenerationDate*, *Appointment*, *StartExecutionPeriod*, *EndExecutionPeriod*, *StartTimeWindow*, *EndTimeWindow*, *Latitude*, and *Longitude*. All attributes except *Appointment*, *StartTimeWindow* and *EndTimeWindow* are fixed in the initalization and do not change during the simulation. The attribute

State Variable	Definition/Units	Initial Value
Tasks		
ReferenceNumber	reference number in simulation, unique values	1001, ..., (number of tasks −1000)
TaskNumber	Internal reference number of Wiener Netze, multiple occurrences	from data
ID	ID of service product	1, ...,10/1N, ...,11N
MassInstallation	grouping value for mass installations, each value occurs as often as there are meters in the mass installation task; 0 if it is no mass installation	0/1, ...,number of mass installations
Skill	lowest required qualification	1/2/3
ExecutionTime	time it takes to execute task; minutes	from data
GenerationDate	date	from data
Appointment	date	–
StartExecutionPeriod	date	Table 3.4
EndExecutionPeriod	date	Table 3.4
StartTimeWindow	earliest arrival time	0
EndTimeWindow	latest arrival time	450
Latitude	geographical coordinate of task	from data
Longitude	geographical coordinate of task	from data
Employees		
ReferenceNumber	reference number in simulation, unique values	1, ...,number of employees
Skill	qualification of the employee	1/2/3
HomeDriver	indicates whether employee starts at headquarters in Erdberg (0), at branch (2) or at home (1)	0/1/2
Latitude	geographical coordinate of start and end point of the route	48.18379 (headquarters), 47.98435 (branch)
Longitude	geographical coordinate of start and end point of the route	16.42232 (headquarters), 16.28355 (branch)
Prio1	list of polygons, numerical array	[1 ... number of polygons]
Prio2	list of polygons, numerical array	[]
Prio3	list of polygons, numerical array	[]
Presence	logical array	from data
StartEnd	numerical array	e.g. [0 450 0 450]
Routes	list of assigned tasks	[]

Table 3.1.: Overview of the state variables and their values of the entities *Tasks* and *Employees*

ReferenceNumber serves as a unique identifier for each task in the simulation, and the attribute *TaskNumber* may occur multiple times if a task has to be executed more than one time on different days, i.e. indicates how often a task has to be assigned until it is completed. *ID* is the number of the service product (Tables 3.2 and 3.3). The variable *Skill* specifies the lowest required qualification to execute the task. *Latitude* and *Longitude* specify the geographical coordinates in which the task is located. The variable *Appointment* is set and possibly changed during the simulation and eventually contains the date on which the task was executed. The variables *StartTimeWindow* and *EndTimeWindow* indicate the earliest and latest arrival time at the location, respectively.

Employees have the state variables *ReferenceNumber*, *Skill*, *HomeDriver*, *Latitude*, *Longitude*, *Prio1*, *Prio2*, *Prio3*, *Presence*, *StartEnd*, and *Routes*. All state variables except *StartEnd* and *Routes* are fixed in the initalization and do not change during the simulation. The attribute *ReferenceNumber* provides a unique identifier for each field service agent, and *Skill* specifies the qualification. *HomeDriver* indicates where the employee starts the working days, i.e. either at the headquarters in Erdberg, at the branch or at home. The geographical coordinates specify the exact starting positions based on the value of the variable *HomeDriver*. The attributes *Prio1*, *Prio2* and *Prio3* define which polygons the field service representative is assigned to and which priorities these assignments have in the task allocation process. *Presence* states on which days the employee is present. The variable *StartEnd* consists of four values, the first two specify the earliest and latest time, respectively, the field service agent can leave the starting position, and the last two values state the earliest and latest time to return. The first and last value of *StartEnd* correspond to the working hours, hence they are fixed and do not change during the simulation. Finally, *Routes* contains a list of the assigned tasks (*ReferenceNumber* of entity *Tasks*) for each working day.

To feasibly calculate travel times we introduce the spatial unit of grid cells, the attributes of this entity (i.e. *GridCell*) are the geographical coordinates of the vertices.

Another entity for spatial units is *Polygon*, its attributes are the geographical coordinates of the vertices and the name of the polygon.

The entities *Tasks* and *Employees* have the additional attributes *GridCell* and *Polygon*, but these are no state variables in the elementary sense since they can be calculated from the coordinates of the agent entities and the locations of the spatial units.

The temporal and spatial scales of the model are as follows: One time step in the simulation represents one day, and when we speak of a day we mean only working days because on weekends and holidays no work is being done in our case. Simulations are run for one

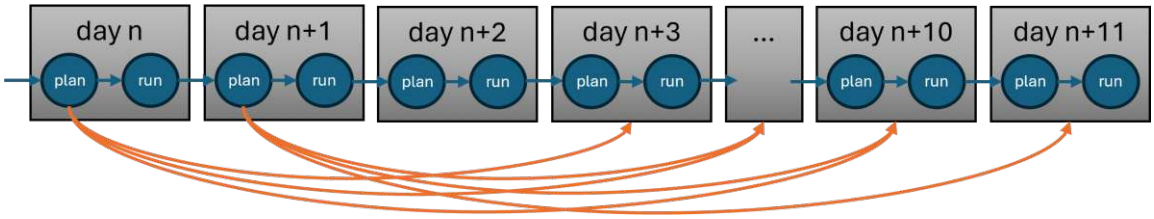


Figure 3.1.: Visualization of our scheduling algorithm. Source: Self-drawn

calendar year, which is approximately 250 working days. The polygons comprise an area of approx. 2,150 square kilometers. One grid cell represents 250 x 250 m, i.e. 62,500 square meters (= 6.25 ha).

3.1.3. Process Overview and Scheduling

The model consists of two main processes where time evolves in discrete steps. In each time step (i.e. one day) the two processes are executed one after another. Starting point of each process is always the current day. Figure 3.1 illustrates the planning and scheduling processes.

The first process models the task assignment planning and scheduling of the next 10 days, but excluding the following two days because according to the planning and scheduling rules (Section 2.1) two days before the planned execution date it is not allowed to change existing packages (which includes adding new tasks).

The second process models the field service activity of the employees, i.e. the execution of the planned tasks and travelling the route (Line 15 in Algorithm 3). We will elaborate on this process in Subsection 3.3.3 (Algorithm 4).

After these main processes, we also calculate the backlog of tasks on each day, i.e. all tasks that are open for planning and execution but have not yet been completed at the end of the day.

Planning Phase

For our planning and scheduling algorithm (Algorithm 3) we store all working days of our simulation period in the variable *workdays* and as mentioned above each time step represents one working day. On each day we first define the period which has to be planned, and since the deadline is fixed to two days we start planning 3 days ahead and end 10 days ahead. Therefore our planning period comprises 8 days. Then we loop over all polygons, whereby the loop is executed in a fixed sequence throughout the simulation. For each polygon the following steps are performed:

Algorithm 3 Scheduling

```
1: for t ∈ workdays do
2:     t_start ← t + 3 days
3:     t_end ← t + 10 days
4:     for m ∈ Polygon do
5:         R ← Routes of all employees from t_start to t_end, i.e. 8 days
6:         ▷ selection 1 ◁
7:         N ← tasks which are generated (i.e. GenerationDate ≤ t), not planned (i.e.
           Appointment does not exist), Polygon = m, StartExecutionPeriod < t_end, and
           have a deadline (i.e. EndExecutionPeriod exists)
8:         ▷ planning 1 ◁
9:         VRPTW(N, R, T, Employees, Tasks, m)
10:        ▷ selection 2 ◁
11:        N ← tasks which are generated (i.e. GenerationDate ≤ t), not planned (i.e.
           Appointment does not exist), Polygon = m, StartExecutionPeriod < t_end
12:        ▷ planning 2 ◁
13:        VRPTW(N, R, T, Employees, Tasks, m)
14:    end for
15:    ExecuteRoutes
16:    CalculateBacklog
17: end for
```

1. We select the *Routes* according to the planning period, i.e. the *Routes* of all field service agents on the specified days.

2. We define the tasks which have to be assigned and planned, we do that in two steps:

   (i) The first selection contains all tasks which are generated and ready for planning, located in the polygon we are currently in, the start of the execution period has to be before the end of the current planning period, and the task has to have a deadline, i.e. an end of the execution period. These tasks are then assigned and planned according to the VRPTW algorithm which we will elaborate in Subsection 3.3.3.

   (ii) The second selection of tasks is very similar to the first one, except we do not require the task to have a deadline, i.e. an end of the execution period for this selection of tasks may or may not exist. The planning is the same as for the first selection of tasks (VRPTW).

The input variable T of our VRPTW Algorithm contains the travel times between all grid cells, calculated with the Graphhopper Routing Engine as described in Section 4.1.1.

Execution Phase

After looping over all polygons we start our second process ExecuteRoutes (Line 15) as already mentioned above. During this process, all planned tasks are carried out in the specified order and the travel times are multiplied by a route delay factor that takes into account delays on the route. If certain tasks cannot be completed due to the delay, they are scheduled for the next planning phase.

3.2. Design Concepts

In this section, we describe the general concepts underlying the design of the model.

Basic principles

In each simulation step, two separate processes are executed in succession, namely the planning phase and the execution phase. The planning phase models the task assignment and scheduling for the respective planning period, while the execution phase represents the actual field service activity on the specific day. To simulate the task assignment planning and scheduling properly, our main concept is the vehicle routing problem (see Section 2.3). We use the expansion with time windows since there are fixed working hours on each day and the tasks have to be planned within these hours. To obtain the travel times between the tasks in an acceptable computational time we introduce the concept of grid cells and use the center of a cell as location for all tasks in the cell.

Emergence

Since the task allocation process is based on every single attribute of the agents, one change can influence all subsequent planning processes. So, this agent-based model has a typical emergent behaviour where little differences for an agent can affect the whole planning and scheduling process. It is the purpose of the model to observe this emergent behaviour and examine how the outcome changes.

Objectives

The objective of the adaptive trait Routes is the highest possible ratio of execution time vs total working time, i.e. to minimize travel times. When searching for an insertion position in a route and there are more than one possibilities the task chooses the one with the smallest extra travel time caused by the insertion.

Prediction

When planning the mass installations in the VRPTW algorithm, the actual planning and scheduling process only starts when certain conditions for the free working time of the field service agents are fulfilled. With these conditions it is very likely to find a feasible insertion position for the tasks. Otherwise every single position would have to be checked even though it is highly unlikely for them to be feasible and a lot of computational time is wasted.

Interaction

Agents interact with each other directly via the assignment of *Tasks* agents to *Employees* agents and the execution of the tasks. Additionally, there are indirect interactions via competition, that means each task is assigned to the employee where it fits best regarding already existing tasks in the route, qualifications and the assigned polygon priorities.

Stochasticity

In the scheduling algorithm we use stochastic elements during the execution of the routes. We apply stochasticity for the travel times between the tasks and the actual execution times. These features model possible traffic jams and delays on the route as well as account for varied operational circumstances during task execution. The whole planning process including the vehicle routing problem algorithm is purely deterministic.
The initialization on the other hand uses quite a few random processes. When running the simulation with actual data, we block the slots for the RSVK meter readings randomly. When running the simulation based on anticipated future tasks or scenarios, there are more stochastic processes involved. The assignment of qualifications to employees uses a random permutation of the generated list of qualifications. When assigning the starting positions of the field service agents we use random permutations as well, first after creating HomeDriver and then when assigning random coordinates from our data to those field service agents who start their workday at home. The creation of the *Tasks* agents for anticipated future tasks uses random samples for the geographic coordinates, the generation dates and the execution times and a random permutation for the service products.

Collectives

Since the service area is divided into polygons, and service technicians are assigned to polygons, tasks in a specific polygon can only be executed by allocated technicians. Additionally, field service agents are grouped by their qualification. Regarding travel times, all

tasks inside one grid cell are treated as one location, so the travel time to some other grid cell is the same for all these tasks.

Observation

Throughout the simulation, for each day we store the total execution time of all tasks completed on that day as well as the current backlog of tasks. The output of the model is the list of tasks each service technician has executed on each day (ReferenceNumber of *Tasks* in Routes of *Employees*) and refers to the *Tasks* agent where the time window when the task has been done is stored. Based on this output, we can calculate every aspect of the model we need, like the number of executed tasks per day or per month. Of great importance is the percentage of the execution time from the working time on each day, the higher this percentage the better the result regarding distribution of resources.

3.3. Details

In this section, we introduce the initialization process and explain the submodels briefly mentioned in the overview in detail.

3.3.1. Initialization

For initializing the grid we first import a complete grid over Austria (for data reference see Subsection 3.3.2). To save memory and keep the travel time calculation feasible we only store the grid cells in which there are actual coordinates of Wiener Netze.
The travel times are calculated as described in Section 4.1.1. In our calculation traffic is not considered, therefore we add a fixed amount of time for traffic and searching for a parking space to all calculated travel times.

Initialization of *Tasks*:
To initialize the *Tasks* agents we use the data provided by Wiener Netze, in particular we first import the following properties into a table: task number, task type, service product, primary working reason, execution time, generation date, the actual appointment and the geographic coordinates (*Latitude* and *Longitude*).
Then we initialize the state variables of the *Tasks* agents: The values for *TaskNumber*, *ID*, *ExecutionTime*, *GenerationDate*, *Latitude*, and *Longitude* are simply copied from the imported properties (where we use Table 3.2 for *ID*).
To initialize the state variable *MassInstallation* we extract all tasks with *ID* 1 and search for all unique tasks with regard to *ExecutionTime*, *GenerationDate*, the actual appointment as well as *GridCell* and *Polygon*. Then we go through this list of tasks and if a subset of

ID	Service product	Skill	Quantity 2018	Quantity 2019	Quantity 2020	Quantity 2021
1	New installation	2/3	15,453	26,506	25,147	27,465
2	Removal	3	1,696	1,995	1,756	1,791
3	Replacement	3	12,334	14,423	14,730	13,490
4	Inspection	3	30,916	32,544	28,114	23,009
5	Periodic change	2	14,950	10,678	10,302	3,394
6	Sampling	3	0	0	8,241	0
7	Special meter reading	1	44,910	68,579	45,558	37,543
8	Shutdown control	1	0	0	832	14,404
9	Commercial shutdown	1	35,774	79,900	60,595	50,442
10	Commercial reactivation	1	9,812	15,871	11,851	14,892
			165,845	250,496	207,126	186,430

Table 3.2.: Service products, Skill: 1 $\widehat{=}$ Q2, 2 $\widehat{=}$ Q3 light, 3 $\widehat{=}$ Q3, the quantity refers to the generation date (not the execution date!)

ten or more tasks in the complete task list shares these identical attributes we group these tasks by assigning a grouping value to the variable *MassInstallation*.

The attribute *Skill* is initialized by using Table 3.2 or 3.3 and the variables *StartExecutionPeriod* as well as *EndExecutionPeriod* are defined as given in Table 3.4.

The inital value for *StartTimeWindow* is 0 (representing the beginning of the workday at 7 am) and the intial value for *EndTimeWindow* is set to 450 (representing the end of the workday at 3 pm minus a 30 minute break, resulting in a maximum total working time of 450 minutes).

If we want to run the simulation with anticipated future tasks (see Table 3.3 for the service products and their expected quantity) we have to create and initialize the state variables *TaskNumber*, *ExecutionTime*, *GenerationDate*, *Latitude*, and *Longitude* of the *Tasks* agents in a different way: The variable *TaskNumber* is assigned sequential values ranging from 10,000,001 to (10,000,000 plus the total number of tasks), ensuring unique identification for each task. To initialize the attribute *ExecutionTime* we sample execution times from the data using the MATLAB function `randsample`. These sampled values are then allocated to the respective tasks according to Table 3.5.

To acquire the attribute *GenerationDate* we sample from the generation dates of the data of the year 2019. As for the initialization of geographical coordinates, we sample from the *Latitude* and *Longitude* values within the data.

ID	Service product	Skill	Quantity 2027	Quantity 2028
1N	Special meter reading [Electricity]	1	221	0
2N	Special meter reading [Electricity] Smart Meter	1	29,070	29,070
3N	Annual periodic reading (Ferraris meter)	1	115	0
4N	Annual periodic reading (Smart Meter)	1	85,000	85,000
5N	Installation	2/3	17,698	19,468
6N	Device replacement	3	5,846	4,969
7N	Periodic change (recalibration)	2	21,932	29,828
8N	Configuration/QC/Circuits	3	54,102	54,102
9N	Dismantling	3	4,272	4,571
10N	Sealing	3	17,734	19,684
11N	Preparations for firmware updates	1	30,061	30,061
			266,051	276,753

Table 3.3.: Anticipated future tasks, Skill: 1 $\widehat{=}$ Q2, 2 $\widehat{=}$ Q3 light, 3 $\widehat{=}$ Q3

IDs	*StartExecutionPeriod*	*EndExecutionPeriod*	Remarks
1	*EndExecutionPeriod* - 3 days	Appointment	(*)
2 - 6	t_{gen}	–	
7	t_{gen} + 1 day	t_{gen} + 5 days	
8	t_{gen}	–	
9	t_{gen} + 6 days	t_{gen} + 14 days	
10	t_{gen} + 1 day	t_{gen} + 2 days	(*)
1N - 4N	t_{gen} + 1 day	t_{gen} + 5 days	
5N	t_{gen} + 5-20 days	*StartExecutionPeriod* + 3 days	
6N, 7N	t_{gen}	–	
8N	t_{gen} + 1 day	t_{gen} + 2 days	30% of 8N tasks
	t_{gen}	–	all other 8N tasks
9N - 11N	t_{gen}	–	

Table 3.4.: Execution periods of all service products, t_{gen} denotes the generation date of the task; exact parameters of Wiener Netze are indicated with *, all others are assumptions/estimations based on actual values due to lack of data

ID	*ExecutionTime*
1N	ID 7
2N	ID 7
3N	ID 7
4N	ID 7
5N	ID 1
6N	ID 3
7N	ID 5
8N	ID 9 and 10
9N	ID 2
10N	ID 4
11N	14.5 min

Table 3.5.: Execution times of anticipated future tasks

The state variable *MassInstallation* requires a small modification in initialization as well, namely when searching for unique tasks we only consider *ExecutionTime*, *GenerationDate*, *GridCell* and *Polygon*. The remainder of the initialization process stays unchanged. Due to the random sampling of the attributes of the future tasks we are not able to generate mass installations with this process. This is because the random samples produce a more random distribution of tasks than it is the case in reality.
In order to generate tasks which require multiple visits, we increase the total number of initialized tasks by 10%. These additional tasks are duplicates of already generated tasks, we randomly copy them between 1 to 5 times, obtaining a maximum number of 6 visits as given in the data.

Initialization of *Employees*:
The initalization process of the *Employees* agents is as follows: For the state variable *Skill* it depends on whether we run the simulation based on actual data or based on predictions or possibilities how the field service workforce structure could be in the future. In the first case we directly import the list of qualifications from the data provided by Wiener Netze and assign these values to *Skill*. In the latter case we first define percentages of how many field service agents have qualification Q2, Q3 light and Q3, respectively. Then we assign the qualifications to *Skill*, with each qualification occurring as often as the percentage specifies and permute the qualifications randomly using the MATLAB function `randperm`.
To initialize the state variables *HomeDriver*, *Latitude* and *Longitude* of field service agents we have to define a percentage of how many employees are starting their workday at home. First we assign the values 0, 1 and 2 to *HomeDriver*, where the value 0 corresponds to employees who start their workday in the headquarters at Erdberg, field service represen-

tatives with value 1 start at home and with value 2 at the branch south of Vienna. The value 2 occurs exactly 5 times (information provided by Wiener Netze, see Section 2.1), the value 1 occurs as often as the percentage defined earlier specifies and the value 0 fills up the vector. The order of these values is randomized with the MATLAB function `randperm`. In order to obtain the geographic coordinates of *HomeDriver* values 0 and 2 we define the locations of the headquarters and the branch and assign these coordinates. For *Latitude* and *Longitude* of employees with *HomeDriver* value 1 we randomly select coordinates from Austrian population data in the service area (see Subsection 3.3.2 for data reference).

Although three priorities are available in our model and two are actually used in reality, we allocate all polygons to *Prio1* of each employee due to insufficient data availability. *Prio2* and *Prio3* remains empty.

The values of *Presence* are imported from Wiener Netze data.

The variable *StartEnd* is initialized differently depending on whether the field service agent begins the workday at home or at the headquarters/branch. If the agent starts at home, the values are set to [0 450 0 450], while if they start at the headquarters or branch, the values are set to [20 450 20 450]. Here, the value 20 corresponds to 7:20 am, the anticipated departure time for service technicians to reach their first task by 7:45 am, as described in Section 2.1. Additionally, blocked slots for RSVK meter readings at the beginning of each month have to be taken into account. If the simulation run is based on actual data, we implement a random blocking scheme, where 3, 4, or 5 hours are reserved on the first working day of each month, followed by 2, 3, or 4 hours on working days $2-5$, and finally 0, 1, or 2 hours on working days $6-10$. If we run the simulation for the anticipation that only 1% of all RSVK meter readings will remain in the future, we uniformly block 2 hours for all on the first working day of each month, and randomly reserve 0 or 1 hour on working days $2-5$. These blocked hours mean adding the specified time to the first and third value of *StartEnd*.

Routes are simply initialized as an empty list.

3.3.2. Input Data

The model mainly uses data provided by Wiener Netze, which cannot be provided here completely for data protection reasons. Nevertheless, we give a short description of the structure of the input data. For each task, the following properties are given: service product, required qualification, execution time, generation date, actual appointment, and the geographical coordinates, whereby the coordinates have been slightly falsified for data protection reasons. In Table 3.2 an overview of the task volumes for the different service products is given. Data regarding the field service agents include the presence and qual-

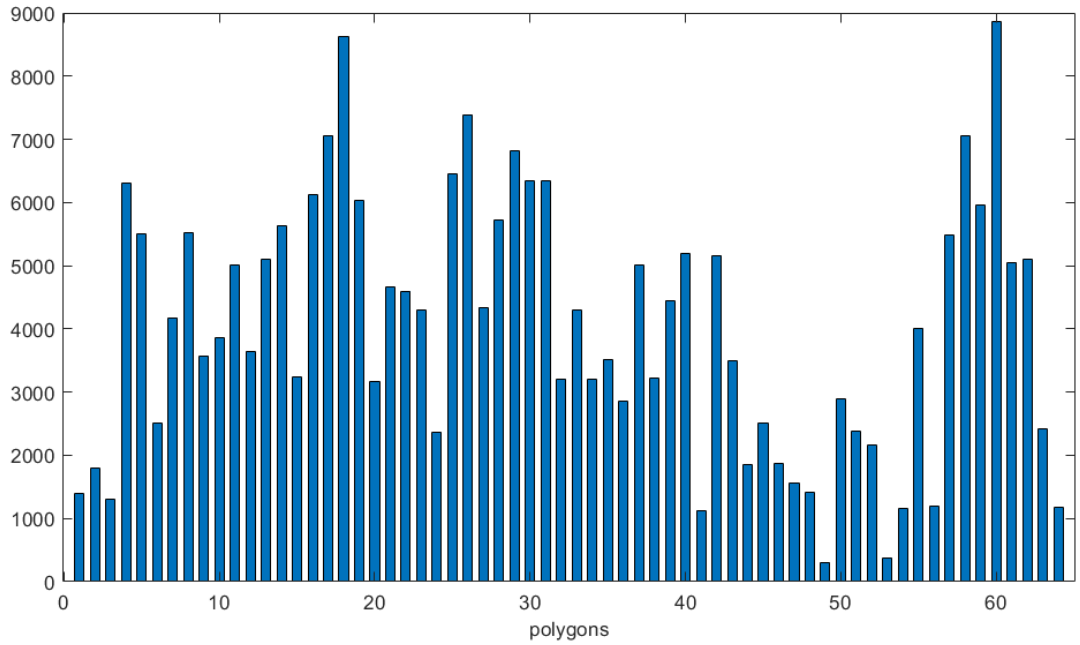


Figure 3.2.: Number of executed tasks per polygon

ification for each employee, places of residence are not specified due to data protection reasons. The polygons are given as a shapefile, they are visualized in Figure 5.12a. In Figure 3.2 the number of actually completed tasks in 2019 in each polygon is depicted.
The coordinates for initializing the starting positions of the employees are extracted from Austrian population data. [1]
The grid used is based on data provided by Statistik Austria [24]. There are different grids available, in our simulation we use the 250m grid.

3.3.3. Submodels

The submodels VRPTW, PROFIT, FEASIBLE and UPDATE were taken from [3] and modified for our needs. The algorithms of these submodels are given in the appendix for completeness. The submodel EXECUTEROUTES was developed from scratch, the algorithm is given at the end of this section. We also briefly describe our submodel CALCULATEBACKLOG.

VRPTW:

Our algorithm consists of two steps, the first step is assigning and planning the mass installations and in the second step we assign and schedule all other tasks. Before starting with step 1, we remove all tasks which require multiple visits (specified by *TaskNumber*)

except one, so that each task in the list is unique. Throughout the whole planning process we use adjusted execution times, in detail we multiply the given execution time by $(1-x)$, with x denoting the reality reduction parameter introduced in Section 2.1.

Step 1: We take all tasks which are mass installations and loop over this list. In each run, we select one group of mass installations and the employees who are assigned to the considered polygon (any priority) with qualification Q3 light or Q3. Then we loop over the selected days. On each day we first calculate the free working time of all employees, and if this value is greater than the sum of the execution time of the tasks in the considered group plus a rough estimation of the driving time of 20 minutes, we start planning. Additionally, we only assign mass installation tasks to a field service agent if its free working time is greater than five times the execution time of one task plus a rough estimate for driving of 10 minutes.

Step 2: We take all tasks which are not mass installations and loop over this list. First we try to find a suitable insertion position for employees with priority 1 in the considered polygon, there we first try to assign the task to employees with the lowest required qualification. We loop over these properties and if necessary we increase the qualification and lower the priority. Inside this loop we try to find the best possible position on all selected days, i.e. the earliest arrival time has to be smaller than the latest arrival time and the value *possible* has to be true (see explanation of the submodel FEASIBLE). Additionally, we compute the profit, i.e. the negative of the extra travel time (see submodel PROFIT). Best position then means the smallest extra travel time. We then insert the task in the best position found and delete it from the task list. Then we update certain attributes of the entities (see submodel UPDATE). If no position in any route could be found we delete the task from the list and it will be planned in a later planning period.

PROFIT:

In the function PROFIT we calculate the negative of the extra travel time when inserting a task in a route. Assuming we want to insert task i between tasks j and $j+1$, we calculate the profit

$$p = -(T_{j,i} + T_{i,j+1} - T_{j,j+1} + ExecutionTime(i)).$$

FEASIBLE:

In the function FEASIBLE we calculate the earliest and latest arrival time for the task to be inserted and whether or not the insertion is possible. We now use the same notation as in [3] and denote with e_i the earliest time the employee can arrive at the task i, i.e. the variable $StartTimeWindow$, and with l_i we denote the latest time the arrival can take place

at task i, i.e. *EndTimeWindow*.
To calculate the earliest arrival time (which is returned as *earliest*) we distinguish between two cases: If the task to be inserted is the first one in the route, we compute

$$e_i = \max(e_i, StartEnd(1) + T_{l,i}),$$

where i is the task to be inserted and $StartEnd(1)$ is the beginning of the working time for employee l corresponding to the route. If we insert task i between two existing tasks j and $j+1$ in a route, we compute

$$e_i = \max(e_i, e_j + T_{j,i} + ExecutionTime(j)).$$

The computation for the latest arrival time (returned as *latest*) is very similar. In case the task to be inserted is the last one in the route, we calculate

$$l_i = \min(l_i, StartEnd(4) - T_{i,l} - ExecutionTime(i)),$$

where $StartEnd(4)$ is the end of the working time for employee l corresponding to the route. If task i is inserted between two existing tasks j and $j+1$ in the route, the formula reads

$$l_i = \min(l_i, l_{j+1} - T_{i,j+1} - ExecutionTime(i)).$$

Finally, we return the value *possible* which gets the value *true* if the considered day is after the start of the execution period and the end of the execution period is after the considered day or the end of the execution period does not exist. Additionally, if the end of the execution period is before the start of the planning period, *possible* is *true* as well. In all other cases we assign $false$.

UPDATE:

In the submodel UPDATE we update the state variables *StartTimeWindow*, *EndTimeWindow* and Appointment of the task which has been inserted and *StartEnd* of the field service agent executing the route. Additionally, *StartTimeWindow* of all tasks after the insertion position and *EndTimeWindow* of all tasks prior to the insertion position have to be updated.
To update the inserted task we assign the value *earliest* from FEASIBLE to *StartTimeWindow* and *latest* to *EndTimeWindow* and the considered day to Appointment.
We again use the notation of e_i as the earliest time the employee can arrive at the task i and l_i the latest time the arrival can take place at task i as explained in the submodel FEASIBLE.

Assuming we insert a task between the already planned tasks j and $j+1$ in the route, for $k = j$ to 1 we compute

$$l_k = \min(l_k, l_{k+1} - T_{k,k+1} - \mathit{ExecutionTime}(k)),$$

since the insertion impacts the latest arrival times of all prior tasks. Analogously, the earliest arrival times of all tasks after the insertion is altered and for $k = j+1$ to n we calculate

$$e_k = \max(e_k, e_{k-1} + T_{k-1,k} + \mathit{ExecutionTime}(k-1)),$$

where n is the number of tasks in the route.
When we identify l_k with $\mathit{StartEnd}(2)$ and e_k with $\mathit{StartEnd}(3)$ the update of these values is analogous to the above equations, except we do not consider an execution time for the calculation of $\mathit{StartEnd}(2)$.

ExecuteRoutes:
The submodel ExecuteRoutes (Algorithm 4) models the field service activity of the employees. We take into account delays on the route (e.g. traffic jam), which is done by multiplying the travel time with a route delay factor. Additionally, we consider not successfully completing a task, i.e. a field service agent has to go there and execute the task again. Finally, we compute the total execution time of all tasks (*ExecutionTime*) being done on one day.
In detail, we proceed as follows: We take all routes (i.e. the state variable *Routes* of *Employees*) for the day we are considering and loop over each route. When processing one route, we first calculate the time when the employee arrives at the first task (*time*). Then we loop over all tasks which are scheduled in the route. We check whether the task is successfully completed or not. In the latter case, we take half of the execution time of the task (specification of Wiener Netze). Then we define a slightly modified execution time based on a triangular distribution with lower limit $\mathit{ExecutionTime} \cdot (1-2x)$, peak location $\mathit{ExecutionTime} \cdot (1-x)$ and upper limit *ExecutionTime*, where x is our reality reduction parameter introduced in Section 2.1. Next, we assign *time* to *StartTimeWindow* and if necessary to *EndTimeWindow*, i.e. if *time* is lower than *EndTimeWindow*. We add the travel time to the next task on the route to *time* and check if this next task can be done in the remaining working time of the employee, i.e. if the current time plus the execution time plus the travel time back to the headquarters or home exceeds the end of the working time or not. In the first case, we delete all remaining tasks from the route since they cannot be accomplished.

CalculateBacklog:

Algorithm 4 ExecuteRoutes

$r \leftarrow$ *Routes* on day t
ExecutionTime $\leftarrow 0$
▷ When not specified otherwise we refer to values on day t ◁
for $k = 1, ..., |r|$ **do** ▷ k = ReferenceNumber of Employee, $|r|$ = |Employees|
  if $|r(k)| > 0$ **then**
    time $\leftarrow$ *StartEnd*(1) for employee executing route $r(k)$ + U(1,1.5) * travel time from starting point to first task
  end if
  $l \leftarrow 1$
  while $l < |r(k)| + 1$ **do**
    if task l in route $r(k)$ couldn't be completed, i.e. has to be executed again **then**
      exectime $\leftarrow$ *ExecutionTime*(l)/2
      ExecutionTime(l) $\leftarrow$ *exectime*
    else
      exectime $\leftarrow$ *ExecutionTime*(l)
    end if
    exectime $\leftarrow$ TriangularDistribution($exectime \cdot (1 - 2x), exectime \cdot (1 - x)$, *exectime*)
    ExecutionTime $\leftarrow$ *ExecutionTime* + *exectime*
    StartTimeWindow(l) $\leftarrow$ *time*
    if *EndTimeWindow*(l) > *StartTimeWindow*(l) **then**
      EndTimeWindow(l) $\leftarrow$ *time*
    end if
    if $l < |r(k)|$ **then**
      time $\leftarrow$ *time* + U(1,1.5) * travel time to next task
      if next task on route $r(k)$ cannot be done in remaining working time **then**
        ▷ triangular distribution for execution times and uniform distribution for travel times included in if statement
        delete all remaining tasks from route, reset *Appointment*, *StartTimeWindow*, *EndTimeWindow* of these tasks
      end if
    end if
    $l \leftarrow l + 1$
  end while
end for
▷ *ExecutionTime* now equals total execution time of all tasks on day t ◁

Our last submodel serves to calculate the number of tasks which have not been completed by the end of each day but are open for planning and execution. More precisely, the backlog contains all tasks that have been generated, the start of the execution period has already begun, and they have not yet been completed or they are scheduled for execution in the future.

4. Model Implementation, Parametrization, Calibration and Validation

In this chapter, we first explain the concepts of our MATLAB implementation including a short introduction to the GraphHopper routing engine as a tool for route calculation and optimization. Subsequently we give an overview of the model's parameters and specification of their values as well as the determination of these values. In Section 4.3 we give a detailed explanation of the calibration process of the reality reduction parameter, initially introduced in Section 2.1, and then we describe the process of validating our model.

4.1. Implementation

Our model is implemented in MATLAB R2023a, where we use the following Toolboxes:

- Automated Driving Toolbox (additionally requires Computer Vision Toolbox and Image Processing Toolbox) to compute and visualize the routes of the service technicians
- Financial Toolbox (additionally requires Optimization Toolbox and Statistics and Machine Learning Toolbox) for date calculations
- Mapping Toolbox to process and visualize geographical data like the polygons
- Parallel Computing Toolbox to accelerate the initialization and travel time computations
- Statistics and Machine Learning Toolbox for stochastic processes

The initialization phase is organized through two distinct scripts. The first script initializes all variables that remain constant throughout all scenarios, therefore it needs to run only once. Our second script is based on the previously initialized variables, we adjust the input parameters (Table 4.2) and create a complete initialization file (*.mat) for each scenario. Next, our main simulation script reads this initialization file, carries out the simulation and generates an output file (*.mat) containing all relevant variables. This script contains the submodel VRPTW as a function, which in turn contains the submodels PROFIT, FEASIBLE and UPDATE as functions (see Section 3.3.3). Finally, these output files are the basis of

our evaluation and visualization of the various scenarios considered. Since a separate file is stored for each scenario, we can carry out any analysis we want.
We run the simulation on an ASUS Vivobook Flip 14 TP12FA-EC417T with the operating system Microsoft Windows 10 Home 64-bit Version 22H2. The computer has an Intel® Core™ i3-10110U CPU 2.1 GHz (4M Cache, up to 4.1 GHz, 2 cores) and 8GB DDR4 memory.
On this machine, a simulation run, i.e. one simulation year, has a computing time of about $6-9$ hours, except for the scenarios with anticipated future tasks, these simulations only take about $1-2$ hours. The reason for this difference is the fact that in the future scenario we start the simulation without any backlog. A larger backlog drastically increases the computing time because starting the planning algorithm with a very large task list requires more resources than with a smaller list.

4.1.1. GraphHopper Routing Engine

In our model, we use the GraphHopper Routing API to obain travel time estimates and routing information. The GraphHopper Routing API is an open-source routing engine that can be hosted on a local server, offering precise routing and travel time information. To integrate this tool into our workflow, we establish a direct connection from MATLAB to the GraphHopper API running on a local server. This section outlines the process of initiating a HTTP POST request from MATLAB to the GraphHopper server for routing information.

First, GraphHopper is installed and configured on our local server. This involves setting up the required dependencies, downloading the required OpenStreetMap (OSM) data, and configuring the server to listen for incoming API requests. Detailed instructions on installing and configuring GraphHopper can be found in the GraphHopper GitHub repository [8] and in the GraphHopper documentation [9].
The GraphHopper server is deployed on localhost, allowing us to access the routing engine within our closed network. This approach provides us with the flexibility to customize the server settings in every detail for our specific needs and maintain control over the data and routing profiles. Additionally, the localhost configuration allows us to make an unlimited number of requests without paying for an API key.
The core of our system involves sending a POST request to the GraphHopper server to obtain travel time estimates. In MATLAB, we construct a POST request with the required parameters, which include the origin and destination coordinates. Additionally, we specified other relevant routing options such as vehicle type, routing profile, and travel mode. The MATLAB script utilizes the `webwrite` function, which writes data to a RESTful web service. We use it to send a HTTP POST request to the GraphHopper API endpoint. The request

Parameter	Value	Reference
minimum number of meters for mass installation	10	Wiener Netze
additional time for traffic and parking	6 minutes	Estimation based on comparison with Google Maps
reality reduction	0.22625	Calibration (section 4.3)
VRPTW Step 1		
minimum driving time for all employees	20 minutes	Own Estimation
minimum number of mass installations assigned to one employee	5	Own Assumption
minimum driving time for one employee	10 minutes	Own Estimation
ExecuteRoutes		
route delay factor	uniformly distributed number in the interval $(1, 1.5)$	Estimation based on comparison with Google Maps and the validation process

Table 4.1.: Overview of parameters, values and references

payload contains the necessary information to calculate the travel time for the specified route. The server processes the request and returns a detailed response which includes the estimated time, distance, and detailed route instructions. After receiving the response from the GraphHopper server, the JSON data is parsed to extract the travel time information we need for our VRPTW Algorithm described in Subsection 3.3.3. In addition, we use the path information contained in the server response for calculating and visualizing specific routes of the field service agents.

4.2. Parameters and Parameter Values

Tables 4.1 and 4.2 present the two categories of parameters within our model: fixed parameters, which remain constant across all simulation runs, and input parameters, which are varied for each scenario. To estimate the additional time for traffic and parking we compared random samples of travel time results provided by Graphhopper with those of Google Maps for various times of the day and used the average value of the deviations.
The values for the allocated time slots reserved for the RSVK meter readings are detailed in Table 4.3. When considering the full set of RSVK tasks we randomly select a value based on the uniform distribution, i.e. the reserved amount of time varies for each employee.

Parameter	Value
home drivers	0%/45%/90%
qualification distribution	
Q2	15% – 40%
Q3 light	0% – 25%
Q3	45% – 70%
polygons	shapefile with geographical information
planning period	January 2, 2019 – December 30, 2019 January 2, 2019 – June 28, 2019 (Calibration)
task selection	
AppointmentReal	January 1, 2019 – December 31, 2019 January 1, 2019 – June 30, 2019 (Calibration)
GenerationDate	December 27, 2018 – December 31, 2019

Table 4.2.: Overview of input parameters and values

working day of month	blocked time (hours)	
	all RSVK tasks considered	1% RSVK tasks remaining
1	3/4/5	1
2 – 5	2/3/4	0
6 – 10	0/1/2	0

Table 4.3.: Parameter values for blocked time slots for RSVK meter readings

These parameter values for simulations taking into account all RSVK tasks are determined by specifications provided by Wiener Netze and during the validation process. To obtain the values for the scenario of 1% remaining RSVK meter readings, we first calculate the execution times needed to completing the RSVK tasks for each month, then we compute 1% of the mean of these execution times. In the final step we adjust the reserved time slots so that they match this calculated value as closely as possible.

4.3. Calibration

In this section we will describe the calibration of the reality reduction parameter serving to scale down the execution times as explained in Section 2.1. The use of the reality reduction parameter $x \in (0, 1)$ is explained in the appropriate places in Section 3.3.
Our aim with the calibration is to find the parameter so that the ratio of the number of tasks executed in the simulation (denoted with n_{SIM}) to those executed in reality (denoted with n_{REAL}) is nearly one, i.e. the goal is to minimize $|1 - n_{SIM}/n_{REAL}|$. We consider the

period January to June 2019 and select all tasks that were actually executed during this period. When evaluating the outcome, February is omitted because in the simulation always fewer tasks are executed than it was the case in reality, unless we choose an unrealistically high parameter value. This divergence is due to the fact that although the tasks in the simulation are all completed within the first half of 2019, only 56.2% are executed in the same month as in reality. Although this variance occurs in all six months, it is particularly pronounced in February. The values n_{SIM} and n_{REAL} are calculated as the means of the number of executed tasks during the months January, March, April, May, and June 2019. In order to accelerate the calibration process we performed preliminary manual testing and determined that the exact parameter value has to lie between 0.22 and 0.24. The procedure for determining the precise value is a bisection method, which is defined as follows:

1. Define the search interval as $[a, b]$, where $a < b$, with a resulting in an insufficient number of tasks executed in the simulation ($n_{SIM}/n_{REAL} < 1$) and with b in an excessive number ($n_{SIM}/n_{REAL} > 1$).
2. Compute the midpoint $c = (a+b)/2$ and assess the simulation outcome for this value. If $n_{SIM}/n_{REAL} < 1$, update $a = c$ (i.e., $[c, b]$ becomes the new search interval); otherwise, set $b = c$ (i.e., $[a, c]$ becomes the new search interval).
3. Repeat steps 1 and 2 until the search interval is sufficiently small.

For the initial interval we set $a = 0.22$ and $b = 0.24$, iteratively refining until $b - a < 0.002$. In step 3 we could alternatively use the condition to repeat the previous steps until $|1 - n_{SIM}/n_{REAL}|$ is sufficiently small. We neglect this condition because with our implementation (interval sufficiently small) we can control the number of steps and the computation time. The parameter obtained is 0.22625, which serves as the basis for all subsequent simulations.

4.4. Validation

Our validation process consists on the one hand of comparing the simulation results with the real data and on the other hand of face validation in the form of numerous meetings with the project managers at Wiener Netze.
To further assess the feasibility of our simulations and the initial locations of the field service agents, we compute the routes for selected field service agents on specific days across various scenarios and visualize these results for evaluation. Two such visualizations where we select 11 field service representatives on July 23, 2019 are presented in Figures 4.1 and 4.2. In each figure one employee is based at the branch in Oeynhausen/Traiskirchen,

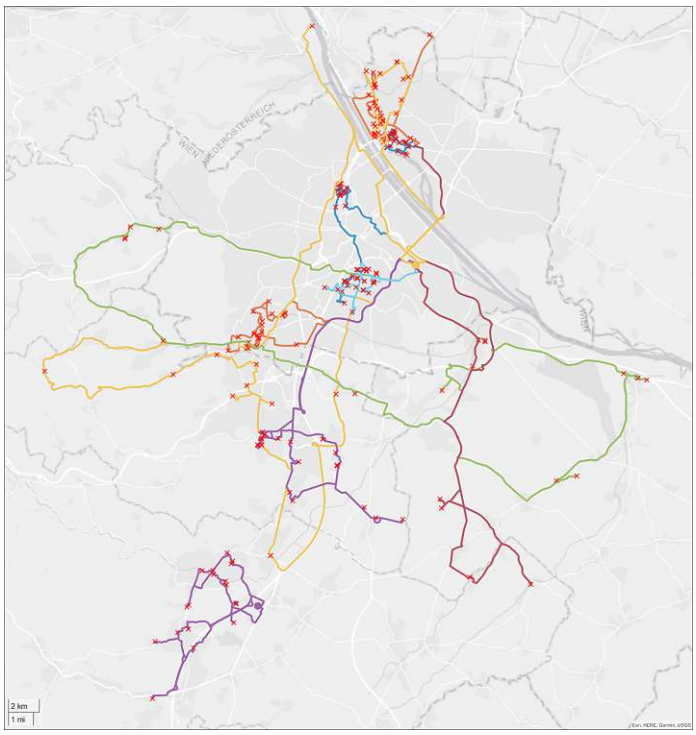


Figure 4.1.: Selection of routes for 0% home drivers on July 23, 2019

while the remaining 10 service technicians commence their routes at Erdberg and at their homes, respectively.

For comparison of the simulation results with the real data we consider the calendar year 2019 and the parameter values according to the real state, namely 0% home drivers and a qualification distribution of 30% Q2 and 70% Q3. We then visualize the number of executed tasks per month and refine certain parameter values and the model itself, until the result is sufficiently accurate to ensure reliability and applicability of our simulation outcomes. In Figure 4.3 we show the comparison between the actual data and the results of our simulation for the period from January to June 2019, the simulation input consists of all tasks that were actually executed during this period. Our simulation shows a high degree of fit to the reality, with 98.9% of all tasks executed in reality also being completed in the simulation.

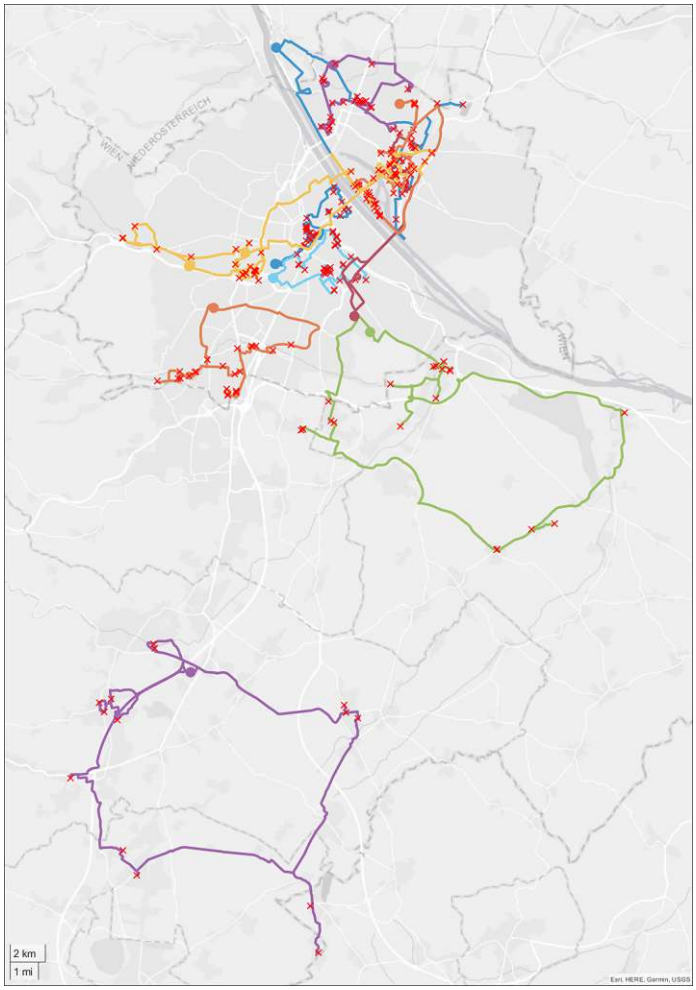


Figure 4.2.: Selection of routes for 90% home drivers on July 23, 2019

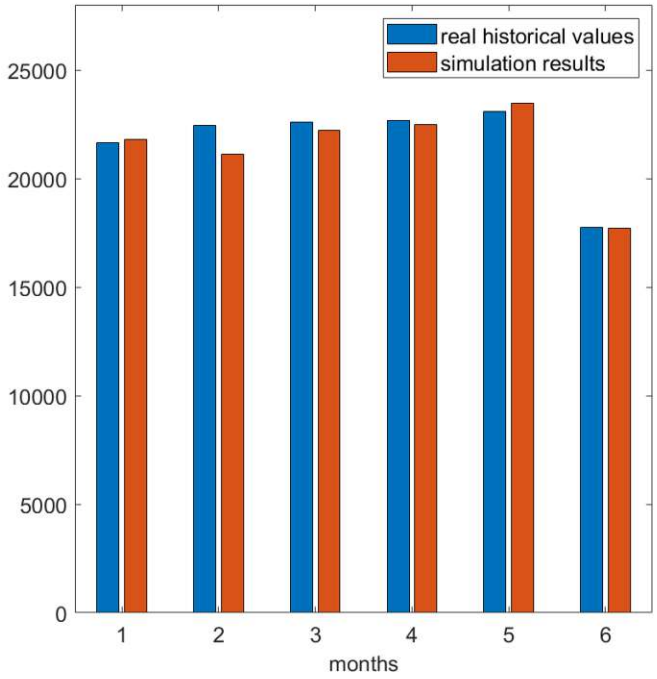


Figure 4.3.: Comparison of the number of executed tasks per month in reality and the simulation for January – June 2019

5. Simulation Experiments, Results and Discussion

In this chapter, we will present and discuss the outcomes of our simulation experiments. All simulations are executed over a duration of one calendar year, more precisely the year 2019. We use the actual attendance records of the employees throughout 2019 as given in the data, the simulation's task list is selected from the comprehensive task list in the data by the following criteria: the tasks have been executed within the year 2019, i.e. the actual appointment of the tasks is between January 1, 2019 and December 31, 2019, and the task generation date lies between December 27, 2018 and December 31, 2019. This results in a task list comprising a total of 277,624 tasks.

Our basis simulation is run with the parameters given in the data, i.e. in 2019 there were no home drivers and the distribution of employee qualifications was around 30% Q2 and 70% Q3 (see Reference scenario in Table 5.1). This simulation represents the reality in 2019 and serves as a reference for the subsequent scenarios, where we compare the number of completed tasks in each month, the ratio of the execution time relative to the total working hours of all employees in each month, and the backlog of tasks. Here, the backlog refers to tasks that are either not scheduled or not executed by the end of the day. More specifically, the backlog contains all tasks that are generated, the start of the execution period has already begun and they have not been completed by the end of the day. The backlog from our Reference scenario is shown in Figure 5.1, while the number of executed tasks and the execution time percentages are presented in the following sections for comparison with the different scenarios. An overview of the scenarios is given in Table 5.1, and we will refer to the names of the scenarios defined in this table throughout this chapter.

5.1. Variation of Employee Starting Positions

By the end of 2023 around 90% of all field service agents are home drivers. Given that five field service agents commence at the branch in Oeynhausen/Traiskirchen, as detailed in Section 2.1, the maximum feasible percentage of home drivers is less than 95%, with

Scenario Name	Home Drivers (%)	Qualifications (%)			Polygons
		Q2	Q3 light	Q3	
Reference	0	30	0	70	Original
HomeDriver45	45	30	0	70	Original
HomeDriver90	90	30	0	70	Original
Quali1_0	0	15	15	70	Original
Quali2_0	0	15	25	60	Original
Quali3_0	0	25	20	55	Original
Quali1_90	90	15	15	70	Original
Quali3_90	90	25	20	55	Original
Quali4_90	90	40	15	45	Original
PolygonVoronoi0	0	30	0	70	Voronoi
PolygonVoronoi90	90	30	0	70	Voronoi
PolygonNone0	0	30	0	70	None
PolygonNone90	90	30	0	70	None
PolygonV1_0	0	30	0	70	Variation 1
PolygonV1_90	90	30	0	70	Variation 1
PolygonV2_0	0	30	0	70	Variation 2
PolygonV2_90	90	30	0	70	Variation 2
PolygonV3_0	0	30	0	70	Variation 3
PolygonV3_90	90	30	0	70	Variation 3
ReferenceFuture	90	15	15	70	Original
Future1	90	15	15	70	Original
Future2	90	25	15	60	Original
Future3	90	40	15	45	Original

Table 5.1.: Overview of scenario parameters and definition of unique scenario names

the precise maximum percentage depending on the overall number of field service representatives. We conduct simulations with two distinct percentages of home drivers, namely 45% and 90%. For our first two simulations we use the original polygons and the qualification distribution according to the data, which is around 30% Q2 and 70% Q3. Figure 5.2 presents the monthly distribution of executed tasks for the Reference scenario alongside the HomeDriver45 and HomeDriver90 scenarios. Our findings show a direct correlation between the percentage of home drivers and the number of executed tasks. More precisely, the introduction of 45% home drivers results in a 4% increase in the number of tasks executed compared to the Reference scenario, and when escalating the percentage of home drivers to 90% this increases the task execution efficiency by 5.8%.

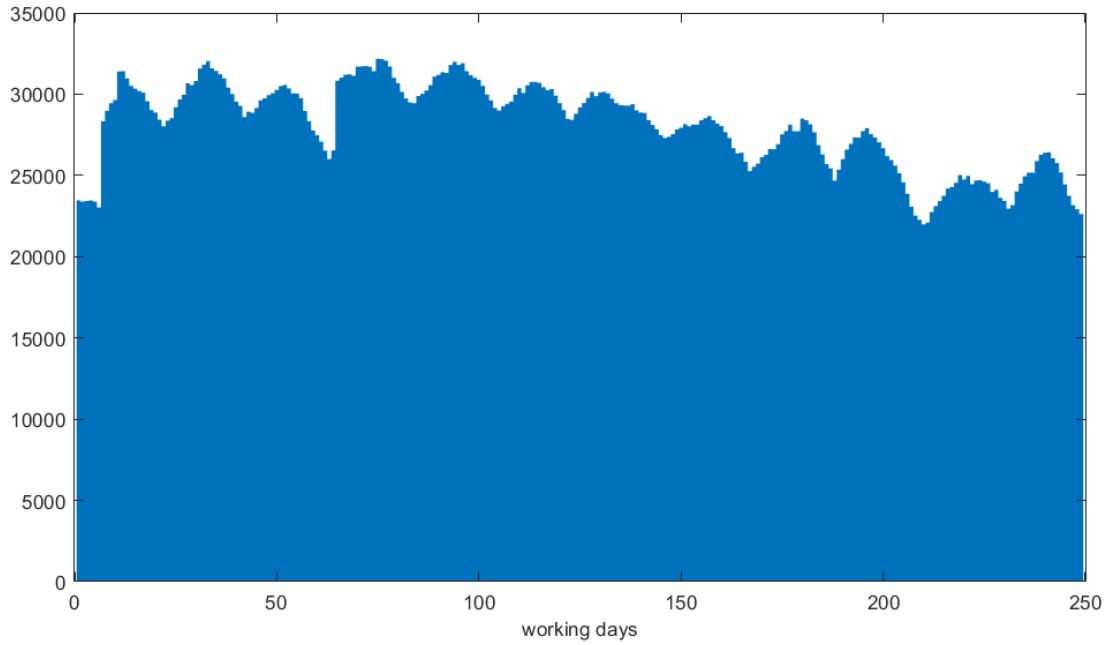


Figure 5.1.: Backlog for Reference scenario

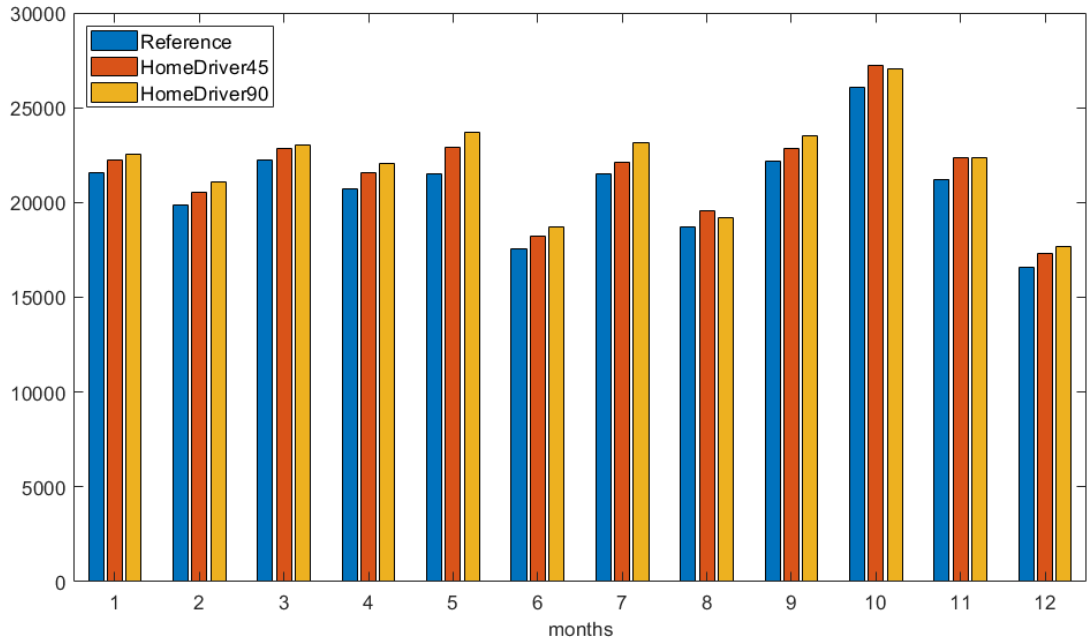


Figure 5.2.: Monthly distribution of executed tasks for Reference, HomeDriver45 and HomeDriver90 scenario

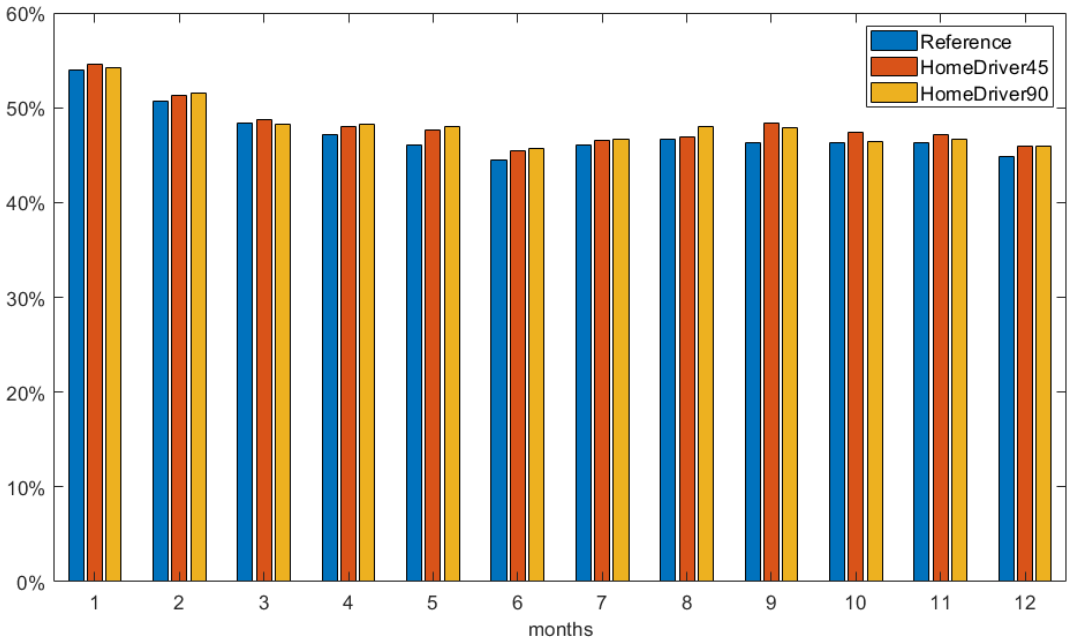


Figure 5.3.: Percentage of the execution time of the total working time for Reference, HomeDriver45 and HomeDriver90 scenario

The percentage of the execution time of the total working hours is visualized in Figure 5.3, where we observe that this ratio increases concurrently with the number of home drivers. However, the percentage increase is small, we have to consider that not only the execution time but also the working hours of a home driver increases (by exactly 20 minutes per day), which then results in a slightly higher percentage. This increase represents a more effective routing of the field service agents since not all of them start at the same location, i.e. the travel times are reduced. Note, that the outcome of the home driver scenario is significantly influenced by the precise locations of residence of the employees, and if there is a clustering of employee residences or a more even distribution within the service area. The starting positions for all field service agents in our simulations with 90% home drivers are shown in Figure 5.4.

The backlog of tasks for the HomeDriver45 and HomeDriver90 scenarios in comparison to the Reference scenario is presented in Figures 5.5 and 5.6, respectively. In contrast to the Reference scenario - depicted in Figure 5.1, where the backlog remains relatively constant from 23,500 to 22,600 tasks across the year - we observe a considerable reduction of the backlog in the HomeDriver90 scenario. More specifically, around 8,200 tasks remain at the end of the year, which is a 63.7% reduction of the backlog compared to the Reference

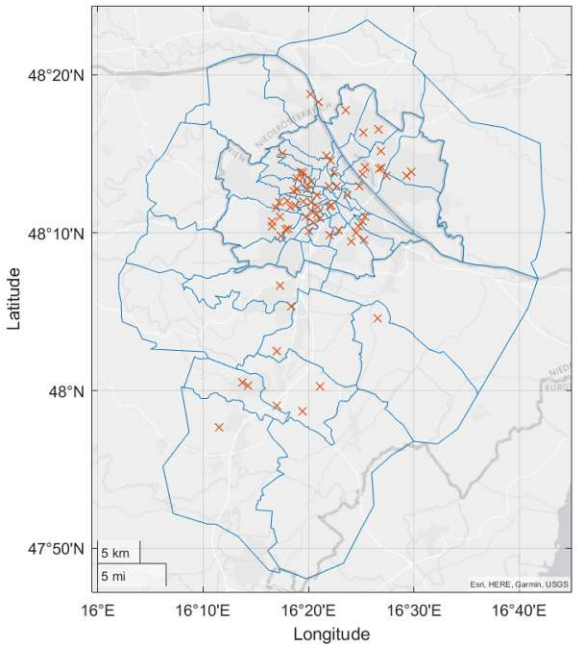


Figure 5.4.: Employee starting positions for 90% home drivers

scenario without home drivers. Furthermore, we observe that the basic shape of the curve is retained, which shows that temporal backlogs cannot be avoided even with home drivers. From our analysis we conclude that if the task volume does not significantly increase, the existing backlog could be eliminated within two years, provided that 90% of all field service representatives are home drivers.

5.2. Variation in Employee Qualification Distribution

By the end of 2023 the qualification distribution of the service technicians is around 15% Q2, 15% Q3 light, and 70% Q3. Our first simulations are conducted with the parameters of the Quali1_0, Quali2_0, and Quali3_0 scenarios as listed in Table 5.1.
Figure 5.7 shows the monthly volume of tasks accomplished. Compared to the Reference scenario, the introduction of the Q3 light qualification does not influence the annual number of completed tasks, more specifically the Quali1_0 scenario. However, we observe a minor decrease in task completion for the Quali2_0 scenario. In this case, 0.3% fewer tasks are executed. Further modification of the qualifications as in the Quali3_0 scenario results in a 0.5% reduction in the number of tasks executed in comparison with the Reference scenario. The homogeneity of the results across all scenarios can primarily be attributed to the

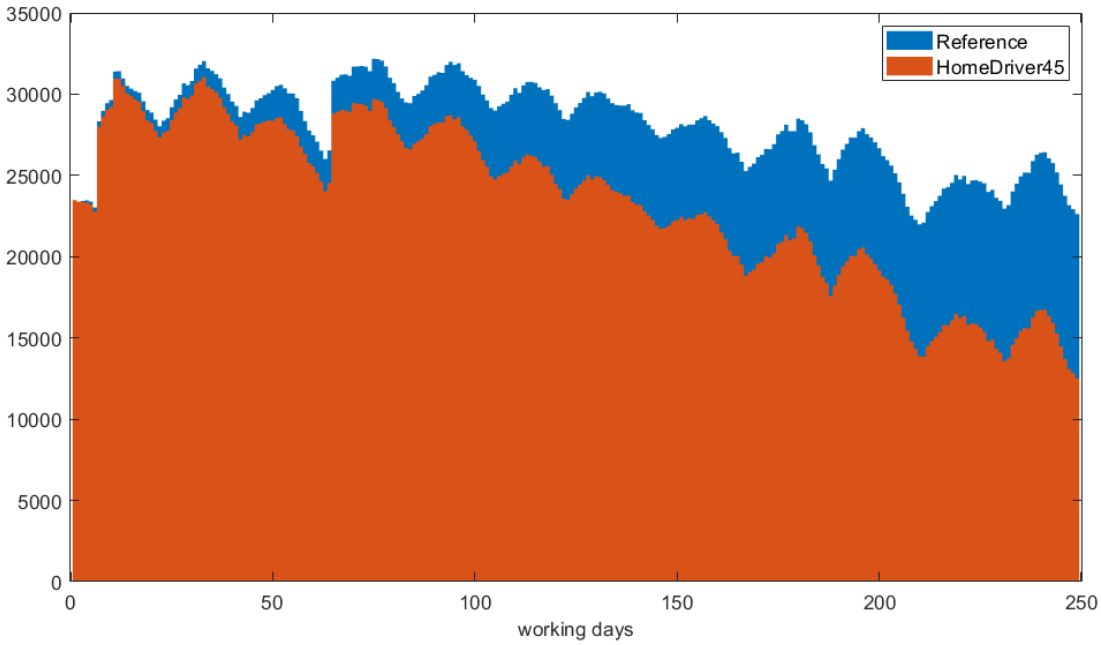


Figure 5.5.: Backlog for HomeDriver45 scenario compared to Reference scenario

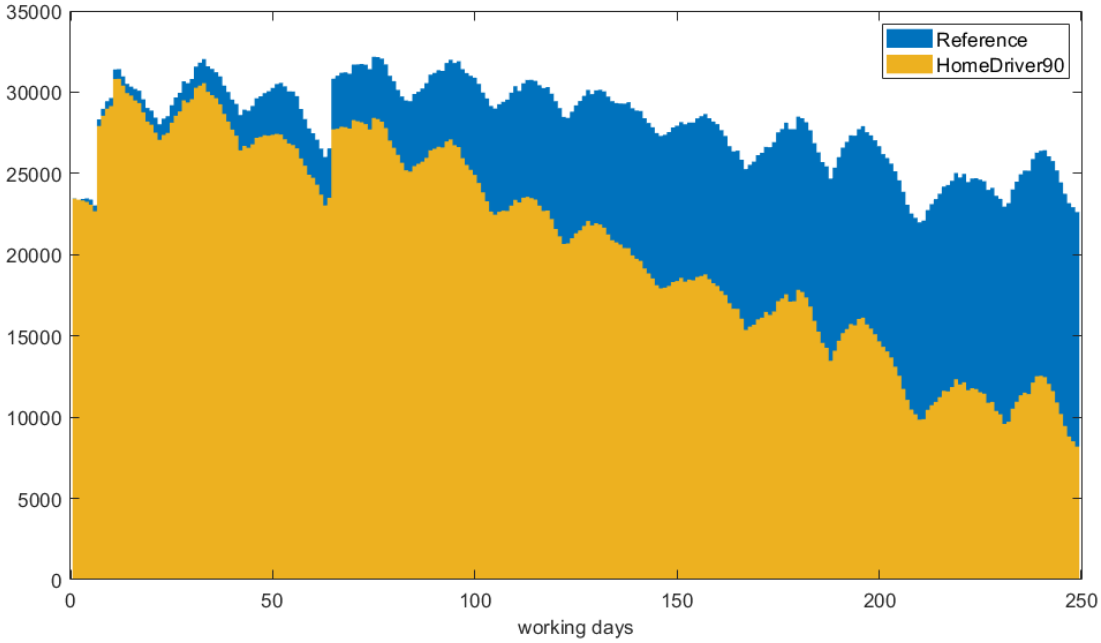


Figure 5.6.: Backlog for HomeDriver90 scenario compared to Reference scenario

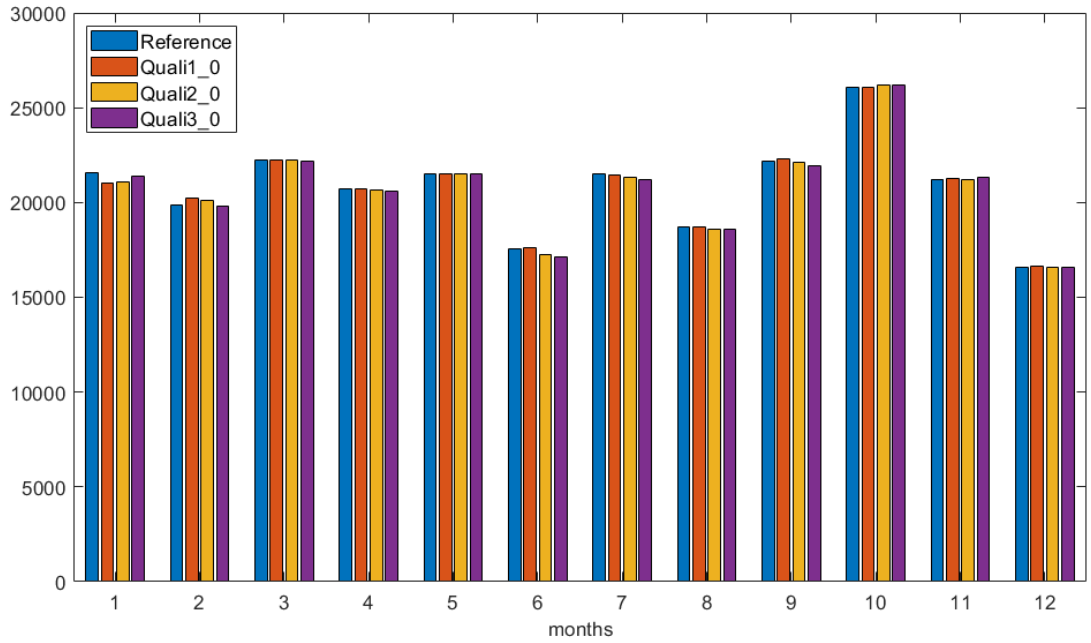


Figure 5.7.: Monthly distribution of executed tasks for Reference, Quali1_0, Quali2_0, and Quali3_0 scenario

distribution of qualifications for the tasks themselves. Our task list comprising a total of 277,624 tasks consists of 60% tasks which require at least qualification Q2, 13% require Q3 light, and the remaining 27% demand Q3. When examining the execution times rather than the number of tasks, the qualification requirements change to 45% Q2, 16% Q3 light, and 39% Q3. This latter distribution is the one we should focus on, since the cumulative execution time of tasks is the crucial parameter for determining the minimum required workforce qualifications in order to feasibly complete the tasks.

In our further analysis, we consider both home drivers and qualifications, as depicted in Figure 5.8. These simulations use the parameters of the Quali1_90, Quali3_90, and Quali4_90 scenarios specified in Table 5.1. In the reference simulation we apply the parameters of the HomeDriver90 scenario. The introduction of the Q3 light qualification results in a slight decrease of 0.2% in task completion for the Quali1_90 scenario. In the other two scenarios, i.e. Quali3_90 and Quali4_90, this reduction increases to 0.8% and 1%, respectively.

Given that 39% of all tasks require a Q3 qualification and that we only observe a 1% decrease in task completion with 45% Q3 service technicians - as opposed to 70% Q3 - this shows that additional Q3 employees do not significantly increase productivity. We conclude that there is a certain degree of flexibility in managing qualification allocations

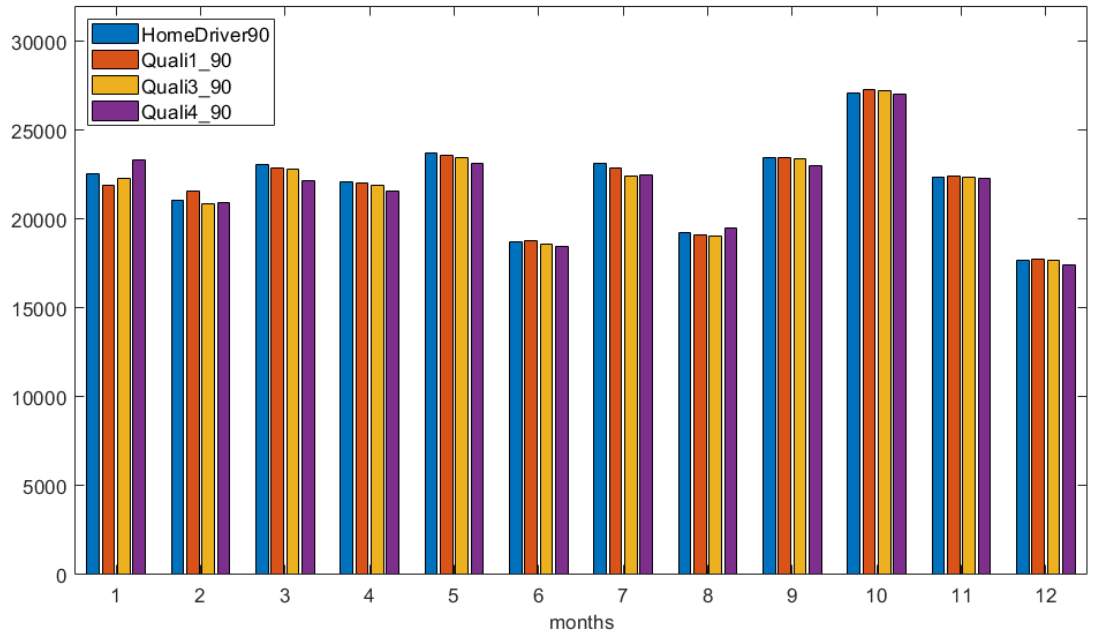


Figure 5.8.: Monthly distribution of executed tasks for HomeDriver90, Quali1_90, Quali3_90, and Quali4_90 scenario

without substantially affecting operational outcomes.

It is important to note that the real-world qualification distribution of the field service agents varies throughout the year due to holidays, sickness absence, and staff changes. When taking into account home drivers, the result also depends on the spatial distribution of the qualifications, i.e. the specific locations where field service agents with particular qualifications are located. For instance, it may be possible that there is a cluster of Q3 employees in one area and the Q2 employees are fairly evenly distributed in the service area. In such a case, the Q3 technicians have significantly larger travel times than the Q2 technicians, and in contrast to an evenly distribution we would need more Q3 employees in order to accomplish the same amount of tasks in the same time.

Figures 5.9 and 5.10 show the backlog of tasks for the Quali1_90 and Quali4_90 scenarios compared to the HomeDriver90 scenario, respectively. We observe that the Quali1_90 scenario yields a 5.5% increase in remaining tasks at the end of the year, i.e. around 450 additional uncompleted tasks. Furthermore, a distribution of qualifications as specified in the Quali4_90 scenario increases the backlog by 34% (or 2,800 tasks) at the end of the year compared to the HomeDriver90 scenario.

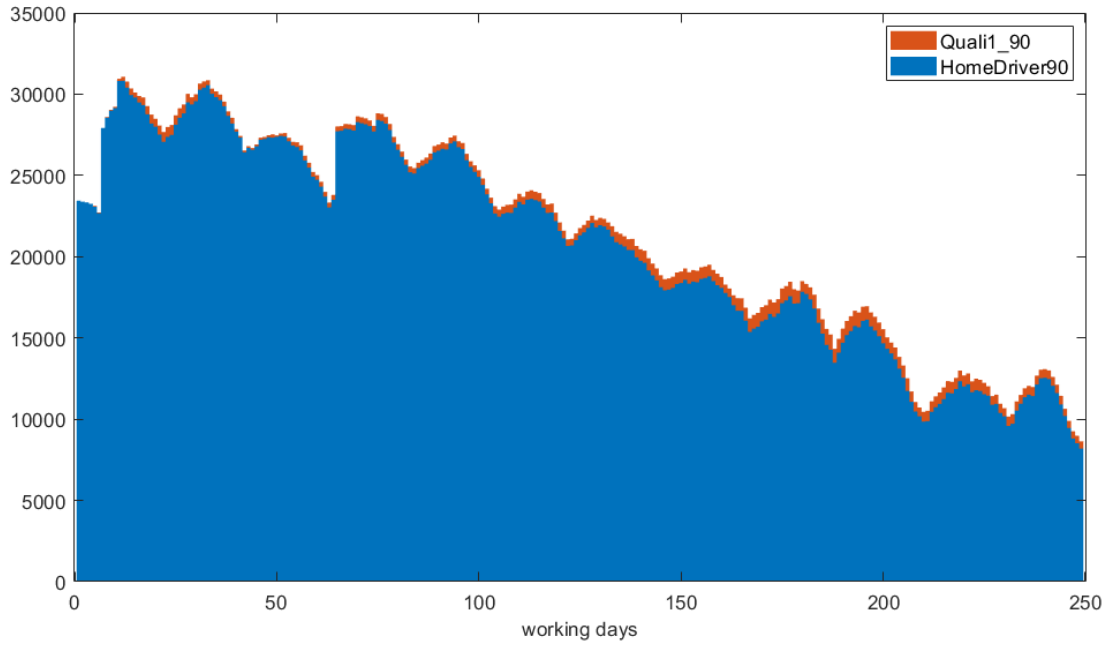


Figure 5.9.: Backlog for Quali1_90 scenario compared to HomeDriver90 scenario

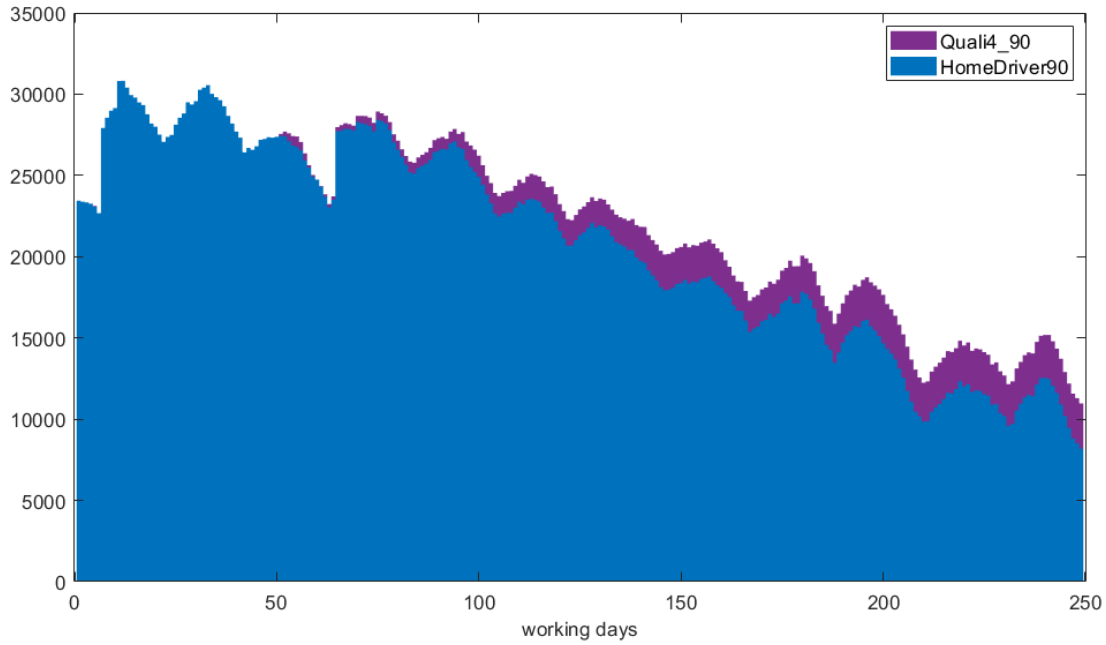


Figure 5.10.: Backlog for Quali4_90 scenario compared to HomeDriver90 scenario

5.3. Variation of Spatial Division of the Service Area

In this scenario, we employ the software QGIS ([21]) for the modification of the polygons. Since there are little indications about how to optimize the existing partition, we choose a method that partitions the whole geographic area regardless of the current division. A relatively simple method that takes into account the distribution of task locations is the construction of a Voronoi diagram. Therefore our first approach is the creation of Voronoi polygons, which are defined as follows:

"Given a set of two or more but a finite number of distinct points in the Euclidean plane, we associate all locations in that space with the closest member(s) of the point set with respect to the Euclidean distance. The result is a tessellation of the plane into a set of the regions associated with members of the point set. We call this tessellation the planar ordinary Voronoi diagram generated by the point set, and the regions constituting the Voronoi diagram ordinary Voronoi polygons." ([17], p.44)

The mathematical formulation of this definition is the following:

"Let $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$, where $2 < n < \infty$ and $x_i \neq x_j$ for $i \neq j, i, j \in I_n$. We call the region given by

$$V(P_i) = \{x \,|\, \|x - x_i\| \leq \|x - x_j\| \;\text{ for }\; j \neq i, j \in I_n\}$$

the planar ordinary Voronoi polygon associated with p_i (or the Voronoi polygon of p_i), and the set given by

$$\mathcal{V} = \{V(p_1), \ldots, V(p_n)\}$$

the planar ordinary Voronoi diagram generated by P (or the Voronoi diagram of P). We call p_i of $V(p_i)$ the generator point or generator of the ith Voronoi polygon, and the set $P = \{p_1, ..., p_n\}$ the generator set of the Voronoi diagram $\mathcal{V}$." ([17], p.45)

In this definition, $\|.\|$ denotes the Euclidean distance. However, Voronoi diagrams can also be defined using the Manhattan metric, as explained in [17], Section 3.7.

In order to obtain the desired set of points, our initial step is to apply k-means clustering to all available data points, i.e. the coordinates where meters are located. A comprehensive discussion on the methodology and application of k-means clustering can be found in [26].

Afterwards, we aggregate these clustered points with the QGIS function *Aggregate*, which "takes a vector or table layer and aggregates features based on a `group by` expression. Features for which `group by` expression return the same value are grouped together." [20]

Subsequently, we calculate the centroids of these grouped clusters. The centroid of a finite

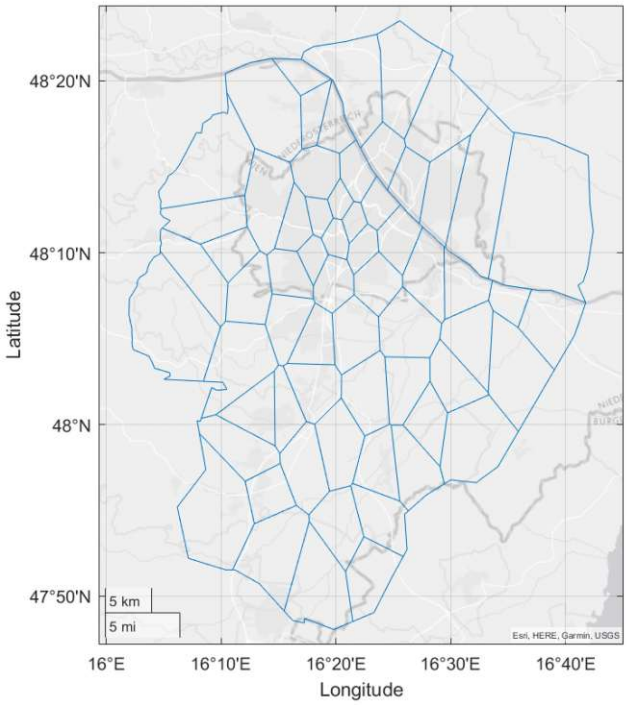


Figure 5.11.: Voronoi polygons (64)

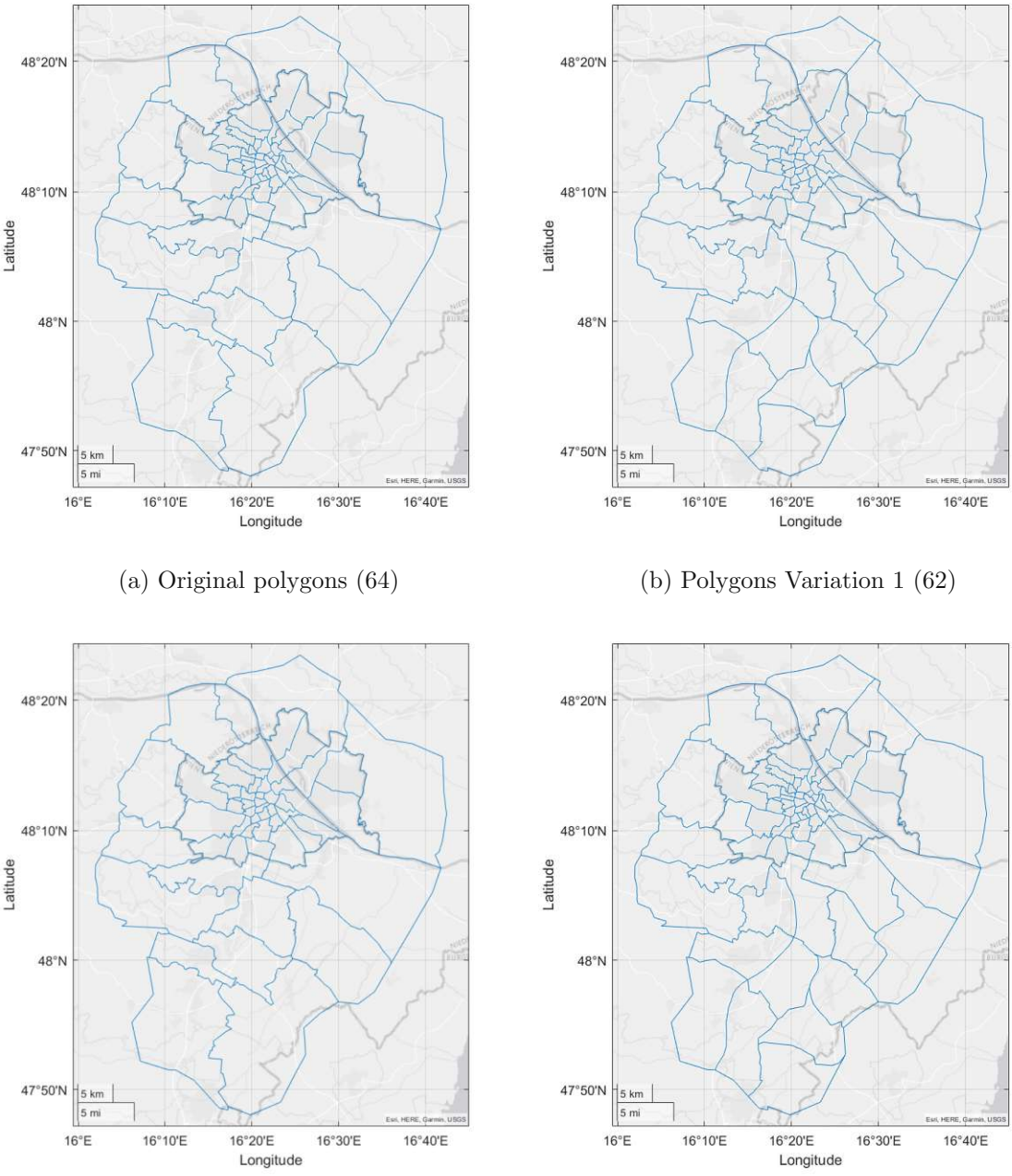
set of k points $x_1, x_2, \ldots, x_k \in \mathbb{R}^n$ is

$$C = \frac{x_1 + x_2 + \cdots + x_k}{k}.$$

This point minimizes the sum of squared Euclidean distances between itself and each point in the set.

These centroids then define our set of points from which we construct the Voronoi polygons. The resulting spatial division is presented is Figure 5.11.

In order to obtain another variation of the spatial division, we commence with the original polygons (Figure 5.12a) and modify the boundaries towards the Voronoi polygons, while still respecting natural boundaries such as streets, which then yields the 62 polygons depicted in Figure 5.12b. Basically, we have combined some existing polygons within Vienna and divided up the polygons in the surrounding area. The polygons depicted in Figure 5.12c (54 polygons) are obtained by combining some of the original polygons within Vienna the same way as in variation 1 and leave the polygons in the surrounding area as they are. In Figure 5.12d (72 polygons) we have left the polygons within Vienna as they are and divided up the polygons in the surrounding area as in variation 1.



(a) Original polygons (64)

(b) Polygons Variation 1 (62)

(c) Polygons Variation 2 (54)

(d) Polygons Variation 3 (72)

Figure 5.12.: Different spatial divisions of the service area in our simulations

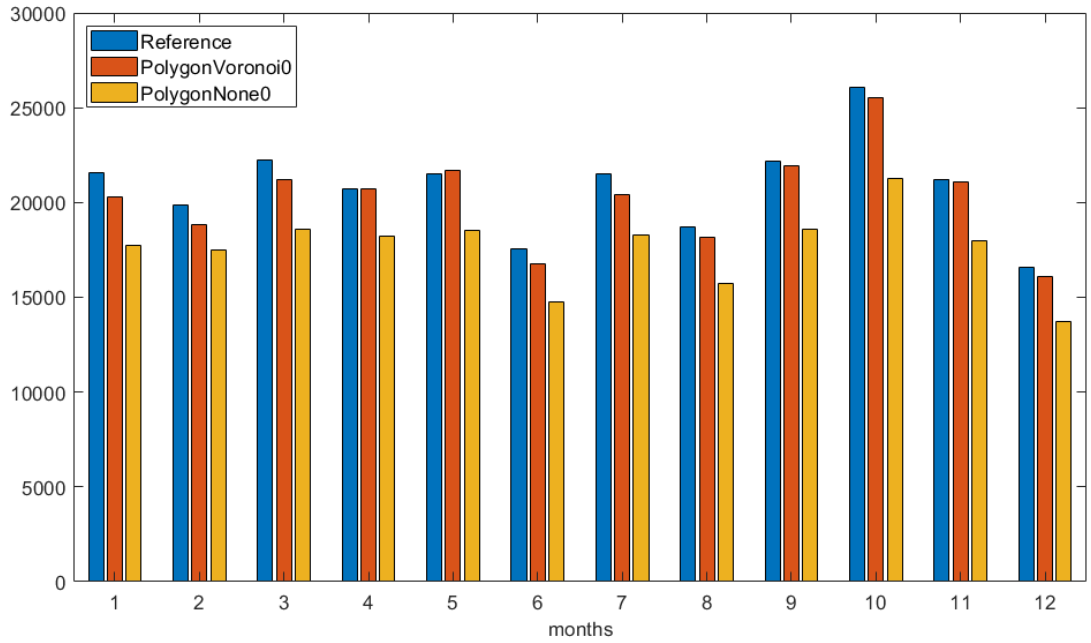


Figure 5.13.: Monthly distribution of executed tasks for Reference, PolygonVoronoi0, and PolygonNone0 scenario

For our initial two simulations, we apply the Voronoi polygons (PolygonVoronoi0 scenario) as well as a configuration without a spatial division, i.e. the whole service area is viewed as one polygon (PolygonNone0 scenario). Figure 5.13 shows the number of executed tasks per month for these scenarios, where we observe a clear difference in efficiency based on the spatial division of the service area. The Voronoi polygons result in a 2.7% decrease in the number of executed tasks compared to the original polygons (Reference scenario), and without a spatial division we even observe a 15.5% decrease. This result shows the tremendous importance of a sensible division of the service area.

When applying our three polygon modifications, i.e. PolygonV1_0, PolygonV2_0, and PolygonV3_0 scenarios, we observe an improvement in efficiency compared to the PolygonVoronoi0 scenario, but the PolygonV1_0 and PolygonV2_0 scenarios still lead to a decrease of 0.4% and 0.6%, respectively, in task completion in contrast to the Reference scenario. The PolygonV3_0 scenario, however, yields a slight increase in the number of completed tasks by 0.2%.

In scenarios considering 90% home drivers (PolygonVoronoi90, PolygonNone90, PolygonV1_90, PolygonV2_90, and PolygonV3_90), the outcomes are very similar to the simulations with 0% home drivers. While the introduction of home drivers results in an

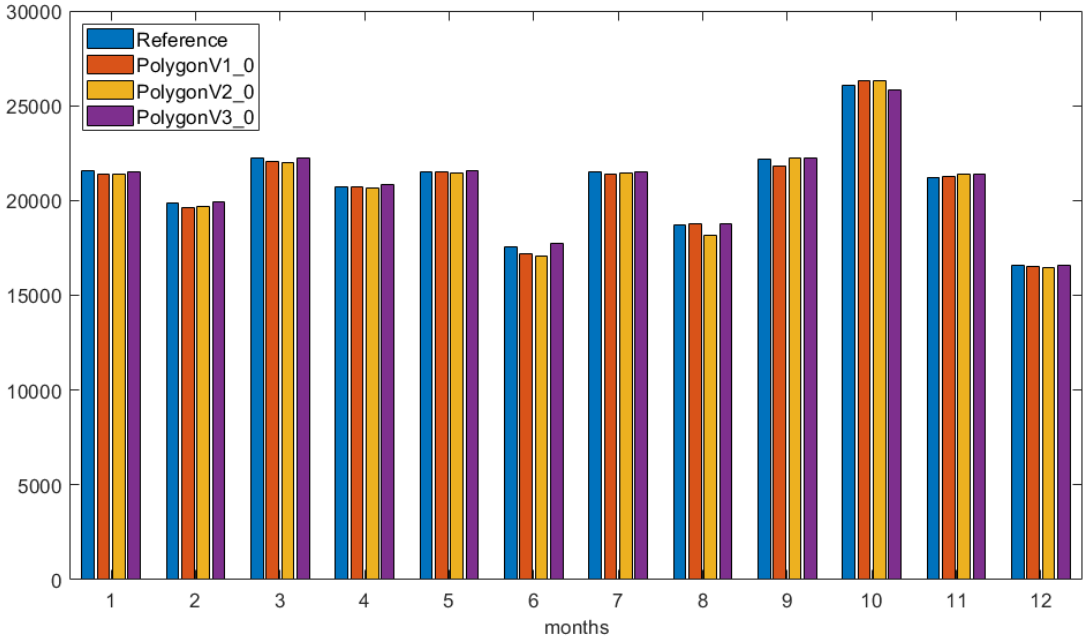


Figure 5.14.: Monthly distribution of executed tasks for Reference, PolygonV1_0, PolygonV2_0, and PolygonV3_0 scenario

overall increase in task execution, the relative performance between the different polygon configurations remains consistent with the observations from the scenarios without home drivers. For an accurate evaluation of these outcomes, the exact places of residence of the field service agents would be necessary, only with this precise information we can determine valid simulation results for operational reality.

Our analysis shows the importance of considering natural geographical and infrastructural boundaries in the division of the service area. The results show that the use of strictly geometric divisions, as generated by Voronoi polygons regardless of existing physical barriers or paths, does not yield efficient outcomes in task execution. Instead, configuring polygon boundaries that follow natural landmarks such as streets will lead to more practical and effective results. Additionally, our analysis of variations 1 – 3 indicates that not only the service area within Vienna but also the surrounding area should be divided into smaller polygons in order to achieve a higher number of completed tasks.

5.4. Anticipated Future Tasks

In this section, we describe the simulation results for the anticipated future tasks in the year 2027, as detailed in Table 3.3. Due to the increase of the number of tasks by 10%, as outlined in Section 3.3, we have a total of 292,656 tasks. Given the previously drawn conclusion that the best results are achieved with 90% home drivers, this parameter value remains constant for all simulations in this section, and we focus on varying the qualification distribution of the field service representatives.

We conduct our basis simulation with all tasks in the data with a generation date between December 27, 2018 and December 31, 2019, resulting in a total of 251,648 tasks. Parameter values for this simulation are specified in the ReferenceFuture scenario in Table 5.1, additionally we assume that only 1% of today's RSVK meter readings need to be carried out, as this will be the case in the future.

It is important to note that compared to the ReferenceFuture scenario we have a 16.3% increase in task volume in the year 2027, without consideration of the RSVK meter readings.

The number of executed tasks per month for 90% home drivers and different qualification distributions is shown in Figure 5.15. We observe a significant increase in the number of executed tasks compared to the ReferenceFuture scenario, which is because 11.3% of all tasks are completed at the headquarters in Erdberg, therefore no travel time is required. Additionally, task attributes such as the execution periods and the locations are drawn randomly from the data, contributing to the observed difference.

A detailed analysis of the number of executed tasks under the future task list yields that with the qualification distribution identical to the ReferenceFuture scenario, i.e. with the Future1 scenario, there is a 15.6% increase in executed tasks. Adjusting the distribution according to the parameters of the Future2 and Future3 scenario results in an increase of 15.3% and 14.5%, respectively. In other words, the task execution efficiency is reduced by 0.9% when altering the qualification distribution of the field service agents from 15% Q2, 15% Q3 light, and 70% Q3 (Future1 scenario) to 40% Q2, 15% Q3 light, and 45% Q3 (Future3 scenario). The reason for this minor difference is the qualification requirements of the tasks themselves, as we already observed and discussed in the previous chapter. In the case of the future task list, 54% of tasks require at least Q2, 8% demand Q3 light, and the remaining 38% require a Q3 qualification. As discussed earlier, the crucial parameter for determining the required qualifications of the field service agents is the execution time, for this value the distribution adjusts to 49% Q2, 14% Q3 light, and 36% Q3.

It is important to note that we do not consider mass installations in this scenario, as explained in Section 3.3.

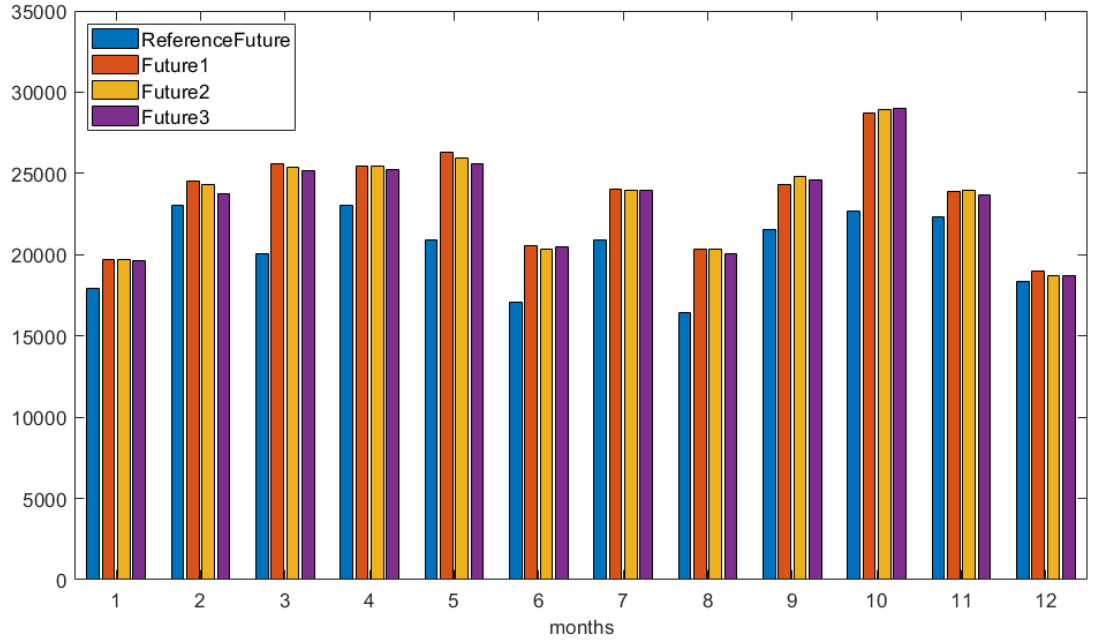


Figure 5.15.: Monthly distribution of executed tasks for ReferenceFuture, Future1, Future2, and Future3 scenario

The backlog for the ReferenceFuture scenario compared to the Future1 scenario, where both have the same qualification distribution but a different task list, is depicted in Figure 5.16. We observe that the backlog is considerably larger for the future task list, despite a higher total number of tasks executed. This result is due to the fact that the task list for the future contains 16.3% more tasks than in the ReferenceFuture scenario, and since only 15.6% more tasks are executed, this results in an increased number of remaining tasks. Due to the definition of the backlog as stated at the beginning of Chapter 5 (mainly the fact that generated tasks with future start dates of the execution period are not included in the backlog) and the quite different attributes of the task lists the discrepancy of the backlog appears to be much higher than it actually is, namely only around 2,000 additional tasks remain uncompleted at the end of the year. Further analysis of the backlog across different qualification distributions shows that the Future2 scenario results in around 600 more tasks (or a 6.2% increase) remaining at the end of the year in contrast to the Future1 scenario, as shown in Figure 5.17. Moreover, the Future3 scenario results in around 2,000 more tasks (or a 27.8% increase) remaining uncompleted, as illustrated in Figure 5.18.

From our observations, we conclude that due to the increase in the number of tasks an expanded field service workforce structure would be necessary in order to avoid an increase

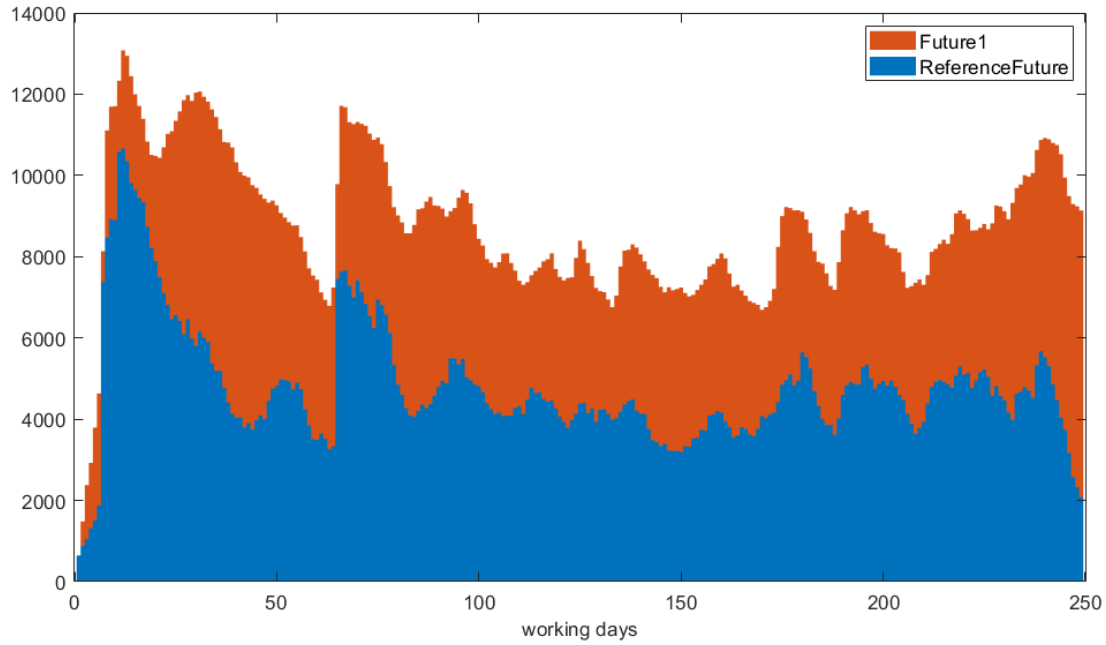


Figure 5.16.: Backlog for ReferenceFuture scenario compared to Future1 scenario

in backlogs. In addition, a reduction of Q3 employees by 10% - from 70% to 60% - would not have a significant impact on the task allocation process, demonstrating a degree of flexibility in managing qualification allocations without substantially affecting operational outcomes.

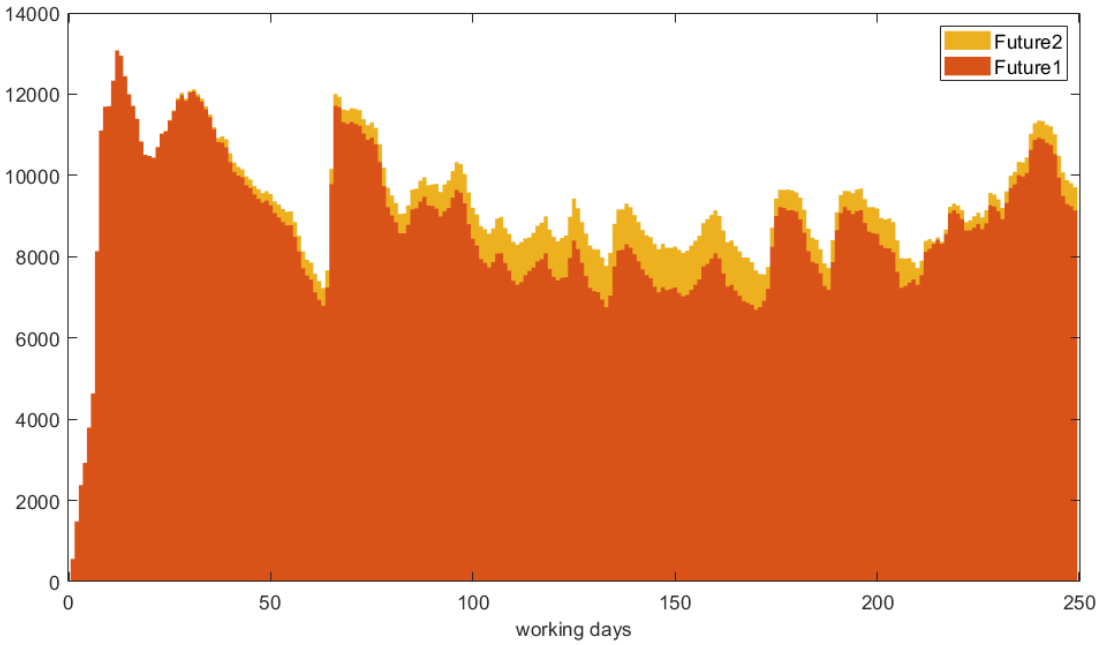


Figure 5.17.: Backlog for Future2 scenario compared to Future1 scenario

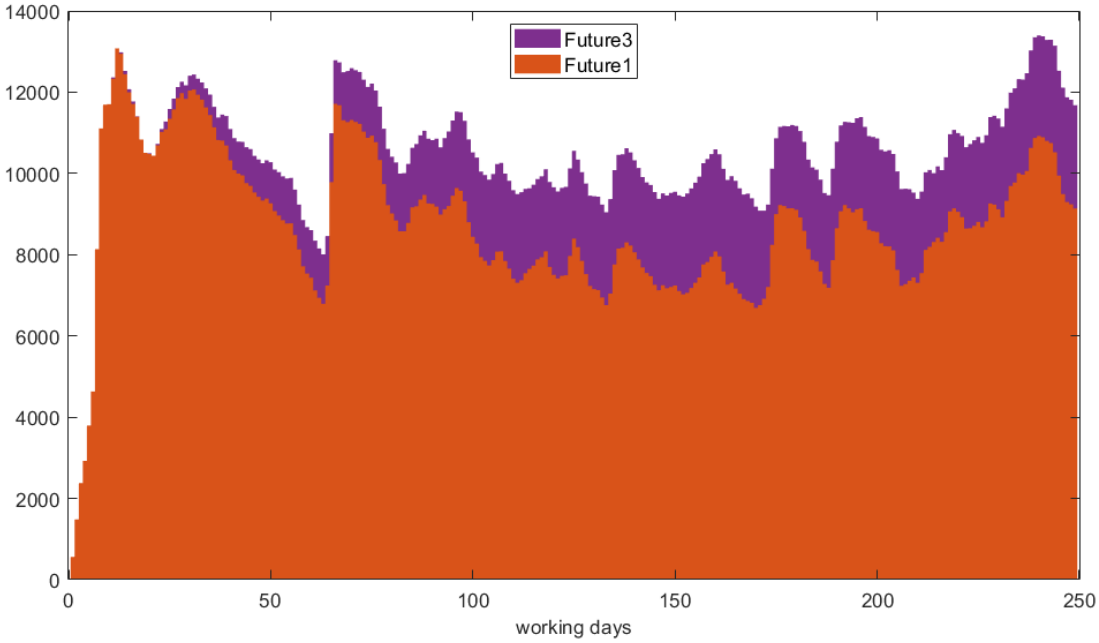


Figure 5.18.: Backlog for Future3 scenario compared to Future1 scenario

6. Conclusion and Outlook

In this thesis, an agent-based simulation model was developed to optimize the task planning processes within Wiener Netze. By using agent-based modeling alongside the vehicle routing problem with heuristic techniques, we were able to simulate and evaluate different scenarios to determine their impact on the efficiency of task allocation and execution.

The simulation experiments yielded several important findings. Increasing the percentage of home drivers leads to a significant increase in operational efficiency, highlighting the potential of flexible working arrangements. Nevertheless, the model cannot depict potential advantages of employees being present in person in the company in the morning. Furthermore, analyzing different qualification distributions of field service representatives revealed that adjustments of the qualifications are coupled to the skill requirements of the tasks themselves. More precisely, as long as the qualification distribution of service technicians is within the range of the requirements of the tasks, these adjustments do not have a significant impact. The spatial partitioning experiments emphasized the importance of designing service areas based on natural landmarks rather than a strict geometric layout, which offers more practical and effective results. Moreover, our study indicates that a refined division of the current polygons could lead to a more efficient task execution in the field service.

The results of our research should be interpreted with a nuanced understanding of the scope and limitations of the model. The model has a wide range of potential applications and proves extremely useful in scenarios for which it was specifically designed. However, it is essential to recognize the limitations of the model, as its applicability diminishes in scenarios that diverge significantly from its basic design principles.
Moreover, due to various simplifications we made, the numbers and percentages presented in Chapter 5 should not be taken too precisely and there will be a quantitative deviation from reality. However, the model is designed to provide a qualitative understanding of the behavior of the system under different conditions.
A main limiting factor were insufficient data and anonymized data due to data protection. If accurate and complete data would be provided the scenario of home drivers could be evaluated with real places of residence of the employees and the outcome would therefore be a more accurate depiction of reality. Furthermore, with precise coordinates of task

locations accurate initialization of mass installations would be possible.

Building on the foundations laid in this thesis, there are several opportunities for future research that will allow the model's capabilities to be extended and new applications to be explored.
In the modeling process we have made a number of simplifications. These inaccuracies could be improved in a future research project, including deadline exceptions, cancelled tasks, and the presence of at least one Q3 technician for mass installations. Moreover, with additional data and knowledge from Wiener Netze, a feasible assignment of polygons to employees could be implemented. With this, the fact that three field service agents are always available for all polygons can be taken into account.
Regarding the variation of the spatial division of the service area, a more detailed analysis could unveil a truly efficient polygon configuration.

Finally, while the computational complexity of the problem limited the scope of our analysis to a limited set of scenarios within the complete parameter set, access to high-performance computing resources in a future research could allow a more comprehensive examination of these scenarios.
We conclude, that our modeling and simulation approach was well suited for accurately depicting the research problem. It demonstrated a significant potential for improving the task allocation processes at our company partner Wiener Netze and has potential to be generalized to other applications.

Bibliography

[1] Martin Bicher, Christoph Urach, and Niki Popper. GEPOC ABM: A GENERIC AGENT-BASED POPULATION MODEL FOR AUSTRIA. In *2018 Winter Simulation Conference (WSC)*, pages 2656–2667, Gothenburg, Sweden, December 2018. IEEE.

[2] Andrei Borshchev and Alexei Filippov. From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools, 2004.

[3] Ann Melissa Campbell and Martin Savelsbergh. Efficient Insertion Heuristics for Vehicle Routing and Scheduling Problems. *Transportation Science*, 38(3):369–378, August 2004.

[4] G Clarke. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, 12(4):568–581, 1964.

[5] Koen H. Dam, Igor Nikolic, and Zofia Lukszo, editors. *Agent-Based Modelling of Socio-Technical Systems*. Springer Netherlands, Dordrecht, 2013.

[6] G. B. Dantzig and J. H. Ramser. The Truck Dispatching Problem. *Management Science*, 6(1):80–91, October 1959.

[7] Ahmed El-Nashar. *Multi-Vehicle Dispatching And Routing With Time Window Constraints And Limited Dock Capacity*. PhD thesis, University of Central Florida, 2012.

[8] GraphHopper Routing Engine. `https://github.com/graphhopper/graphhopper`, October 2023.

[9] Documentation - GraphHopper Directions API. `https://docs.graphhopper.com/`.

[10] Routing via Java API. `https://github.com/graphhopper/graphhopper/blob/master/docs/core/routing.md`.

[11] Volker Grimm, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K. Heinz, Geir Huse, Andreas Huth, Jane U. Jepsen, Christian Jørgensen, Wolf M. Mooij, Birgit Müller, Guy Pe'er,

Cyril Piou, Steven F. Railsback, Andrew M. Robbins, Martha M. Robbins, Eva Rossmanith, Nadja Rüger, Espen Strand, Sami Souissi, Richard A. Stillman, Rune Vabø, Ute Visser, and Donald L. DeAngelis. A standard protocol for describing individual-based and agent-based models. *Ecological Modelling*, 198(1-2):115–126, September 2006.

[12] Volker Grimm, Uta Berger, Donald L. DeAngelis, J. Gary Polhill, Jarl Giske, and Steven F. Railsback. The ODD protocol: A review and first update. *Ecological Modelling*, 221(23):2760–2768, November 2010.

[13] Kai Hormann and Alexander Agathos. The point in polygon problem for arbitrary polygons. *Computational Geometry*, 20(3):131–144, November 2001.

[14] Manar Hosny. Heuristic Techniques for Solving the Vehicle Routing Problem with Time Windows. *IACSIT Press*, 13:6, 2011.

[15] Nicholas R Jennings, Katia Sycara, and Michael Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, 1:7–38, 1998.

[16] C M Macal and M J North. Tutorial on agent-based modelling and simulation. *Journal of Simulation*, 4(3):151–162, September 2010.

[17] Atsuyuki Okabe, editor. *Spatial Tessallations: Concepts and Applications of Voronoi Diagrams*. Wiley Series in Probability and Mathematical Statistics. Wiley, Chichester Weinheim, 2. ed edition, 2000.

[18] Jean-Yves Potvin and Jean-Marc Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, May 1993.

[19] Jean-Yves Potvin and Jean-Marc Rousseau. An Exchange Heuristic for Routeing Problems with Time Windows. *The Journal of the Operational Research Society*, 46(12):1433–1446, December 1995.

[20] Documentation for QGIS 3.34 — QGIS Documentation documentation. `https://docs.qgis.org/3.34/en/docs/index.html`.

[21] Willkommen beim QGIS Projekt! `https://www.qgis.org/de/site/`.

[22] Tourenplanung und mobiles Workforce Management. `https://reisewitz.com/`.

[23] Marius M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, 35(2):254–265, April 1987.

[24] Regionalstatistische Rastereinheiten - data.gv.at. `https://www.data.gv.at/katalog/dataset/ca996648-ed34-3dcd-a99f-68619f680241`.

[25] Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, Pa, 2002.

[26] Junjie Wu. *Advances in K-means Clustering: A Data Mining Thinking*. Springer Theses. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

A. Algorithms

Algorithm 5 VRPTW part 1

```
1:  function VRPTW(N, R, T, Employees, Tasks, polygon)
2:      remove multiple visited tasks from N except one
3:      ▷ step 1: mass installations in N ◁
4:      N_mass ← tasks in N which are mass installations
5:      while N ≠ ∅ do
6:          N_mass1 ← tasks with mass installation group index of N_mass(i)
7:          pr ← indices of employees with any priority assignment for polygon
8:          qu ← indices of employees in polygon with qualification Q3 light or Q3
9:          sect ← intersection of pr and qu
10:         for k = 1, ..., number of columns of R do ▷ number of columns of R is always 8
            due to choice of t_start and t_end
11:             x ← 0
12:             for j ∈ sect do
13:                 if employee j is present then
14:                     x ← x + free working time of employee j
15:                 end if
16:             end for
17:             if x > ∑ N_mass1.ExecutionTime + 20 then ▷ x is free working time of all
                employees on day k, 20 is estimation for min. driving time (in minutes)
18:                 for j ∈ sect do
19:                     if employee j is present then
20:                         if free working time of employee j > 5 *
                            N_mass1.ExecutionTime(1) + 10 then ▷ 10 is estimation
                            for min. driving time (in minutes)
21:                             p* ← 0
22:                             while p* > −5000 & N_mass1 ≠ ∅ do
23:                                 p* ← −5000
24:                                 for l = 1, ..., number of tasks in route R(j, k) − 1 do
25:                                     p ← PROFIT(N_mass1(1), T, R(j, k)(l), R(j, k)(l + 1))
26:                                     [ej, lj, possible] ← FEASIBLE(N_mass1, T, R(j, k)(l),
                                        R(j, k)(l + 1), workdays(k))
27:                                     if ej ≤ lj & p > p* & possible then
```

Algorithm 6 VRPTW part 2

```
                                    ej* ← ej
                                    lj* ← lj
                                    j* ← j
                                    k* ← k
                                    l* ← l
                                    p* ← p
                                end if
                            end for
                            if p* > −5000 then
                                insert task N_mass1(1) in route R(j*, k*) between l* and l* + 1
                                delete task N_mass1(1) from N_mass1
                                [Tasks, Employees] ← UPDATE(ej*, lj*, R(j*, k*), l*, Employees(j*), Tasks, workdays(k*))
                            end if
                        end while
                    end if
                end if
                if N_mass1 ≠ ∅ then
                    break
                end if
            end for
        end if
        if N_mass1 ≠ ∅ then
            break
        end if
    end for
    delete planned tasks from N_mass
end while
▷ step 2: all other tasks in N ◁
N ← all tasks in N except mass installations
while N ≠ ∅ do
    p* ← −5000
    for pr = 1, 2 do
        for qu = N(1).Skill, ..., 3 do
            sect ← indices of employees with priority pr and qualification qu
            for j ∈ sect do
                for k = 1, ..., number of columns of R do ▷ number of columns of R is always 8 due to choice of t_start and t_end
                    for l = 1, ..., number of tasks in route R(j, k) − 1 do
                        p ← PROFIT(N_mass1(1), T, R(j, k)(l), R(j, k)(l + 1))
                        [ej, lj, possible] ← FEASIBLE(N_mass1, T, R(j, k)(l), R(j, k)(l + 1), workdays(k))
```

Algorithm 7 VRPTW part 3

```
67:                                  if ej ≤ lj & p > p* & possible then
68:                                      ej* ← ej
69:                                      lj* ← lj
70:                                      j* ← j
71:                                      k* ← k
72:                                      l* ← l
73:                                      p* ← p
74:                                  end if
75:                              end for
76:                          end for
77:                      end for
78:                      if p* > −5000 then
79:                          break
80:                      end if
81:                  end for
82:                  if p* > −5000 then
83:                      break
84:                  end if
85:              end for
86:              if p* = 5000 then
87:                  delete task N(1) from N
88:              else
89:                  insert task N(1) in route R(j*, k*) between l* and l* + 1
90:                  delete task N(1) from N
91:                  [Tasks, Employees] ← UPDATE(ej*, lj*, R(j*, k*), l*, Employees(j*),
                     Tasks, workdays(k*))
92:              end if
93:          end while
94:          return Employees, Tasks
95:      end function
```

Algorithm 8 PROFIT

```
1: function PROFIT(N, T, l, l1)
2:     ▷ N ... task to be inserted ◁
3:     ▷ l ... reference number of task/employee prior to insertion position ◁
4:     ▷ l1 ... reference number of task/employee after insertion position ◁
5:     p = −(T_{l,N} + T_{N,l1} − T_{l,l1} + N.ExecutionTime)
6:     return p
7: end function
```

Algorithm 9 FEASIBLE

```
1: function FEASIBLE(N, T, l, l1, day)
2:     ▷ N ... tasks to be planned, i.e. all unplanned tasks ◁
3:     ▷ l ... reference number of task/employee prior to insertion position ◁
4:     ▷ l1 ... reference number of task/employee after insertion position ◁
5:     ▷ day ... current working day ◁
6:     if l is employee then
7:         early ← StartEnd(1) of employee l
8:         earliest ← max(N.StartTimeWindow(1), early + T_{l,1})
9:     else ▷ l is task
10:        early ← N.StartTimeWindow(l)
11:        earliest ← max(N.StartTimeWindow(1), early + T_{l,1} + N.ExecutionTime(l))
12:    end if
13:    if l1 is employee then
14:        late ← StartEnd(4) of employee l1
15:        latest ← min(N.EndTimeWindow(1), late − T_{1,l1} − N.ExecutionTime(1))
16:    else ▷ l1 is task
17:        late ← N.EndTimeWindow(l1)
18:        latest ← min(N.EndTimeWindow(1), late − T_{1,l1} − N.ExecutionTime(1))
19:    end if
20:    possible ← false
21:    if day > N.StartExecutionPeriod(1) then
22:        if N.EndExecutionPeriod(1) exists & day < N.EndExecutionPeriod(1)
           then
23:            possible ← true
24:        else if N.EndExecutionPeriod(1) does not exist then
25:            possible ← true
26:        end if
27:    end if
28:    return earliest, latest, possible
29: end function
```

Algorithm 10 UPDATE

function UPDATE($ej, lj, r, l^*, emp, Tasks, day$)
  ▷ l^* ... position in r prior to inserted task ◁
  ▷ emp ... employee executing route r ◁
  ▷ day ... current working day ◁
  $StartTimeWindow(r(l^*+1)) \leftarrow ej$
  $EndTimeWindow(r(l^*+1)) \leftarrow lj$
  $Appointment(r(l^*+1)) \leftarrow day$
  for $k = l^*, ..., 1$ **do**
    $EndTimeWindow(r(k)) \leftarrow \min(EndTimeWindow(r(k)),$
    $EndTimeWindow(r(k+1)) - T_{r(k),r(k+1)} - ExecutionTime(r(k)))$
  end for
  $StartEnd(2) \leftarrow \min(StartEnd(2), EndTimeWindow(r(1)) - T_{emp,r(1)})$
  for $k = (l^*+2), ..., |r|$ **do**
    $StartTimeWindow(r(k)) \leftarrow \max(StartTimeWindow(r(k)),$
    $StartTimeWindow(r(k-1)) + T_{r(k),r(k-1)} + ExecutionTime(r(k-1)))$
  end for
  $StartEnd(3) \leftarrow \max(StartEnd(3), StartTimeWindow(r(end)) + T_{r(end),emp} +$
  $ExecutionTime(r(end)))$
  return $Tasks, emp$
end function

B. Tables

ID	Service product English	Service product German
1	New installation	Neuaufstellung
2	Removal	Wegnahme
3	Replacement	Wechslung
4	Inspection	Überprüfung
5	Periodic change	Turnuswechsel
6	Sampling	Stichproben
7	Special meter reading	Sonderablesung
8	Shutdown control	Abschaltekontrolle
9	Commercial shutdown	kaufmännische Abschaltung
10	Commercial reactivation	kaufmännische Wiederinbetriebnahme
1N	Special meter reading [Electricity]	Sonderablesung [Strom]
2N	Special meter reading [Electricity] Smart Meter	Sonderablesung [Strom] Smart Meter
3N	Annual periodic reading (Ferraris meter)	Turnusablesung Ferraris - 1x jährlich
4N	Annual periodic reading (Smart Meter)	Turnusablesung Smart Meter - 1x jährlich
5N	Installation	Montage
6N	Device replacement	Gerätewechsel
7N	Periodic change (recalibration)	Turnuswechsel (Nacheichung)
8N	Configuration/QC/Circuits	Parametrierung/QS/Schaltungen
9N	Dismantling	Abmontage
10N	Sealing	Plombierung
11N	Preparations for firmware updates	Vorbereitungen für Firmware-Updates

Table B.1.: English-German translation of service products

List of Figures

List of Algorithms

List of Tables