Master's thesis Econometrics and Management Science
Operations Research and Quantitative Logistics

Extensions of a High-Performance Local Search Framework for Practical Vehicle Routing Problems

Adding Time-Dependent Travel Times and Break Scheduling to a Hybrid Genetic Search for the Capacitated Vehicle Routing Problem with Time Windows

Abstract

In this thesis, two methods are introduced to extend the Local Search (LS) framework of the Hybrid Genetic Search (HGS) by Vidal et al. (2013) with the aim of solving advanced, practical Capacitated Vehicle Routing Problems with Time Windows. Firstly, a general Iterative Method (IM) is introduced where the LS ignores the added practical constraints, such that it deals with a relaxation of the full problem. Instance data is changed iteratively for the LS to converge to a feasible solution. Secondly, a more specialized Local Search Method (LSM) is proposed, where the LS actively takes the practical constraints into account in a sophisticated way. After extending the distance-only objective with the route duration, the IM and LSM are applied by incorporating break scheduling and time-dependent travel times in the HGS. On a selection of 192 real-world instances, the two methods are compared with the algorithm ORTEC currently uses (OHD), to show their potential. The results show that, when adding a single practical constraint, the IM is very competitive with OHD after 20 minutes of running time and improves the average objective value of OHD by 1% to 2% after two hours. When more practical constraints are implemented using the IM, the results tend to fall short. The results of the LSM after 20 minutes heavily depend on the constraint(s) considered. However, the LSM outperforms both the IM and OHD after two hours of running time for all problem types considered.

Author:
J.C. Vellekoop
Student Number:
581335
Date:
October 27, 2022

Supervisor EUR:
Dr. R. Spliet
Second Assessor EUR:
Prof. Dr. A.P.M. Wagelmans
Supervisors ORTEC:
W. Kool
J. Olde Juninck

Erasmus University Rotterdam
Erasmus School of Economics

ORTEC
OPTIMIZE YOUR WORLD

Acknowledgments

This thesis marks the end of my six-year, academic journey that started at the TU Delft in 2016. After four years of Applied Physics and some Computer Science, I decided it was time to combine my love for mathematics and programming to solve logistical problems. Switching to Econometrics was a hard thing to do, especially since COVID-19 started at the same time as I started my pre-master at Erasmus University. In hindsight, I think it was the best decision I could have made. I enjoyed every single course, and I will never forget the projects and case studies I was able to do with an amazing group of friends.

First, I would like to thank Remy Spliet for providing me with constructive feedback in all our Friday afternoon meetings, guiding me in the process of this thesis, and always reminding me to write a concise report. Also, I want to thank Albert Wagelmans for taking the time to read my thesis as a second assessor.

I am especially grateful for my ORTEC supervisor Wouter Kool, who always reminded me to start at the basics, helped me countless times with re-structuring the code, and always took time when I had questions about the project. Also, I would like to thank Joep Olde Juninck for being my ORTEC supervisor for the first three weeks, and for answering all my questions about OHD. I would like to thank ORTEC for proving real-world customer data and making it possible to write my thesis on such an interesting, practical topic. I would like to thank my fellow optimizers at ORTEC for their enthusiasm and ideas during the monthly research projects meeting.

On the personal side, I am very thankful for having the support of my parents through every decision I made in my life. They always gave me the time and freedom to focus on studying, as well as other aspects of my life.

All good things come to an end, as does my journey as a student. With this thesis, I put the cherry on the cake of this academic adventure.

Contents

1 Introduction

Extensive research has been conducted on the Capacitated Vehicle Routing Problem (CVRP), where the goal is to find an optimal set of routes for a fleet of vehicles with limited capacity to satisfy all customer demand. Numerous different techniques, usually highly customized to solve a specific extension of the CVRP, have been used throughout time. A more general framework was introduced by Vidal et al. (2012). They presented a Hybrid Genetic Search with Adaptive Diversity Control (HGSADC), which is a Genetic Algorithm (GA) where each solution is potentially improved by a Local Search (LS). Key strengths of this algorithm are the adaptive control of the diversity of the solutions in the population, the use of infeasible solutions, and the fast LS on each offspring. To also use this algorithm to solve CVRPs with Time Windows (CVRPTW), the HGSADC was extended with time window support by Vidal et al. (2013). Recently, a new SWAP* neighborhood was added by Vidal (2022) and the earlier HGSADC for solving CVRPs was made open-source. On classical CVRPTW literature benchmarks, the HGSADC outperforms all current state-of-the-art approaches. Using an algorithm largely inspired by all three papers above, Kool et al. (2022) won the CVRPTW track of the 2022 DIMACS implementation challenge. This shows that the HGSADC is a state-of-the-art method to solve academic benchmark instances of the CVRPTW.

However, the HGSADC is currently unable to solve real-world problems, where practical constraints have to be taken into account. Hence, this research aims to extend the implementation of the HGSADC of Kool et al. (2022) with support for the most common and fundamentally challenging practical requirements, such that feasible solutions of good quality can be found for real-world CVRPTWs in a reasonable amount of time. The goal of this research is to show the potential of the framework of the HGSADC by Vidal et al. (2013) to solve advanced, practical CVRPTWs. The challenge is to solve those problems without losing the state-of-the-art performance of the HGSADC.

The scientific relevance of this study is twofold. Firstly, this research shows the versatility of the HGSADC and the potential to use this framework to solve practical, challenging CVRPTWs. Secondly, this research focuses on techniques and ideas where the generality of the HGSADC is kept intact and where implementation will often be the main challenge when adding more/other practical requirements. So, while current studies often only focus on CVRPs with only a specific set of one or two constraints, this research integrates *multiple practical requirements* into a *general* algorithm that already incorporates customer time windows. Besides its scientific relevance, this research is also of interest for practical applications. In today's world where large corporations face even bigger logistical challenges to satisfy the increasing customer demand, solving practical extensions of the CVRP is more relevant than ever before. As the transportation process generally accounts for 10% to 20% of the cost of a product according to Moghdani et al. (2021), relatively small improvements in the transportation process could lead to significant cost reductions. More specifically, ORTEC will use this research to determine if the HGSADC, introducing new and promising concepts compared to their current algorithm, has the potential to efficiently solve practical problems for their clients.

A selection is made of three practical requirements which are all essential to incorporate in the HGSADC to show its potential to solve practical CVRPTWs. The requirements are selected

based on their commonness in practical CVRPTWs, and inherent difficulty to add to the HGSADC. Firstly, it is important to add *route duration minimization* to the route evaluation of the HGSADC to take into account a driver's salary for example. Incorporating this requirement makes certain time-saving pre-checks in the LS of the HGSADC invalid. It also adds difficulty in combination with the next requirement, which is to include Time-Dependent Travel Times (TDTTs). This requirement, which will be called *congestion*, adds great complexity to the move evaluations in the LS of the HGSADC. Lastly, it is essential to incorporate *break scheduling*. Optimally scheduling breaks for move evaluations in the LS will increase the time complexity, which removes the strength of fast neighborhood searches of the HGSADC. This research focuses on scheduling at most one break per route, with a specified break duration, in a predefined break window. Other practical requirements, such as vehicle minimization and constrained route duration, will mainly be a matter of implementation and not of changing the fundamental structure of the algorithm. Therefore, they are not necessary to make conclusions about the potential of the HGSADC to solve practical CVRPTWs.

Adding the three selected practical requirements to the LS will result in expensive calculations to check the feasibility and solution value for each possible move. Therefore, the practical constraints are first only taken into account outside the LS, such that the LS deals with a relaxation of the problem. In this first, so-called *Iterative Method* (IM), feasibilities and solution values for the full constrained problem will be calculated only in between the so-called LS phases. Instance data is changed iteratively for the LS to converge to a feasible solution of the full constrained problem. This general approach is also of great interest to other Hybrid Genetic Search algorithms. A more specialized, so-called *Local Search Method* (LSM), is introduced where the LS considers the full constrained problem. To limit the increase in running time as much as possible, estimations (based on bounds on the travel durations) are used to reduce the number of exact move evaluations in the LS. In both methods, breaks are scheduled using a heuristic.

To show the potential of the extended HGSADC to solve practical CVRPTWs, both the IM and LSM are compared with the algorithm ORTEC currently uses (called ORTEC Home Delivery, or OHD), using real-world instances. ORTEC provided a data set of 222 instances from a major grocery chain in the United States. The average number of customers per instance is approximately 330, ranging between 200 and 880 customers. Each instance has one depot, and a specified maximum number of vehicles with a fixed capacity. Demands and vehicle capacities are on average 6.0 and 147.3 respectively, such that an average vehicle visits 25 customers on its route. On average, at least 14 vehicles need to be used based on their capacities.

The results show that, when adding a single practical constraint, the IM is very competitive with OHD after 20 minutes of running time and improves the average objective value of OHD by 1% to 2% after two hours. When both breaks and congestion are implemented using the IM, OHD finds solutions with better objective values on average. The results of the LSM after 20 minutes heavily depend on the constraint(s) considered. However, the LSM outperforms both the IM and OHD after two hours of running time for all problem types considered.

The remainder of this thesis is organized as follows. In Chapter 2, the problem considered in this research is formally defined. Chapter 3 is dedicated to explaining the starting algorithm of Kool et al. (2022) of this research. The relevant literature is presented and discussed in Chapter 4, followed by the methodology in Chapter 5. A description of the data is presented in Chapter 6. The results are reported and discussed in Chapter 7, after which Chapter 8 concludes. In the Appendix in Chapter 9, more details on the results are given, as well as an explanation of a suggestion for further research.

2 Problem Description

In this chapter, the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW) is formally introduced in Section 2.1, after which the three selected practical requirements are described in Sections 2.2, 2.3, and 2.4.

2.1 Capacitated Vehicle Routing Problem with Time Windows

Consider the CVRPTW, which is formally defined on a complete, directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V}$ and $\mathcal{A}$ are the sets of vertices and arcs of $\mathcal{G}$ respectively. Vertex $v_0 \in \mathcal{V}$ stands for the single depot of the instance, and all other vertices $v_i \in \mathcal{V}^C$, for $\mathcal{V}^C = \mathcal{V} \setminus \{v_0\}$ and $i \in \{1, ..., n\}$, represent $n \in \mathbb{N}^+$ customers. Each customer v_i is associated with demand $q_i \in \mathbb{N}^+$, service duration $s_i \in \mathbb{N}^+$, and time interval $[b_i, e_i]$, where $b_i < e_i$, and where $b_i \in \mathbb{N}^+$ and $e_i \in \mathbb{N}^+$ are the begin- and end time of the time interval at which service for customer i has to start. Vehicles arriving at some customer i before b_i have to wait until b_i before starting service. The planning horizon[a] is denoted by $[0, T)$, where $T \in \mathbb{N}^+$ is finite. For the single depot, $q_0 = s_0 = b_0 = 0$ and $e_0 = T - 1$. For all customers i, $b_i < T$ and $e_i < T$. At the single depot, a fleet of $m \in \mathbb{N}^+$ identical vehicles is positioned, each having capacity $Q \in \mathbb{N}^+$. Each vehicle can be used for only one route per instance. A feasible route r can be constructed by a path in $\mathcal{G}$, starting and ending at v_0, where all customers on the route are serviced in their specific time window, and where the total demand of customers on the route does not exceed Q. Subroutes are defined as possibly non-feasible sequences of visits to customers and the depot. $R(s)$ is the set of non-empty feasible routes that together form solution s. Routes use arcs $(i, j) \in \mathcal{A}$, for $i, j \in \{1, ..., n\}$ and $i \neq j$, where each arc is associated with constant distance $c_{ij} \in \mathbb{N}^+$ and constant travel time $\tau_{ij} \in \mathbb{N}^+$. The goal is to find a set of at most m feasible routes with a minimal total cost, such that the vehicles satisfy the demand of all customers, by visiting them once in their specific time windows. Each customer demand must be satisfied by one vehicle. The sum of customer demand for each route has to be at most Q. The total cost of a solution, $Z(s)$, equals the sum of the route distances over all routes $r \in R(s)$, as can be seen in Equation (1):

$$Z(s) = \sum_{r \in R(s)} \alpha C(r) \tag{1}$$

where $C(r)$ is the route distance of route r in meters, and α is the weight of the route distance.

The CVRPTW is $\mathcal{NP}$-complete, as proven by restriction to the Traveling Salesman Problem and the Single Machine Scheduling Problem by Kallehauge (2008). All input data in this report is deterministic and static input data. Processes outside the routing of vehicles are not considered.

[a] Time is discrete in this research. For applications where T is for example one day/week/month, the planning horizon is an half-open interval to not cover boundaries of the time horizon twice. This is also the reason why the last time the depot is open is just before T, so $e_0 = T - 1$.

2.2 Route Duration Minimization

The first practical requirement is to add the route duration to the objective function that can be seen in Equation (1). The new objective function is then given by Equation (2):

$$Z(s) = \sum_{r \in R(s)} (\alpha C(r) + \beta D(r)) \tag{2}$$

where $D(r)$ is the route duration of route r in seconds, and β is the weight of the route duration.

Without break scheduling, this requirement comes down to minimizing travel duration and waiting time. With break scheduling, waiting time is not necessarily disadvantageous, since waiting time can sometimes be used to take the necessary break.

2.3 Congestion

2.3.1 Notation

The notation used for TDTTs will now be formally introduced to include congestion in the model. The planning horizon $[0, T)$ is split into $\mu \in \mathbb{N}^+$ time intervals. The choice of μ is a trade-off between the amount of data needed to be stored and the accuracy with which congestion needs to be captured. The time intervals are denoted by t_k, where $k \in K = \{1, ..., \mu\}$. For every $k \in K$, the corresponding time interval $t_k = [s_k, f_k)$ starts at time $s_k \in \mathbb{N}^0$ and finishes at time $f_k \in \mathbb{N}^+$, where $s_k < f_k$, $s_1 = 0$, and $f_\mu = T$. Since $f_i = s_{i+1}$ for $i \in \{1, ..., \mu - 1\}$, $t_1 \cup t_2 \cup ... \cup t_\mu = [s_1, f_1) \cup [s_2, f_2) \cup ... \cup [s_\mu, f_\mu) = [s_1, f_\mu) = [0, T)$. In general, any travel time function can be considered, but piecewise constant functions are used since this is the format of the data available.

For each arc $(i, j) \in \mathcal{A}$ and each time interval $k \in K$, the congestion is modeled using the so-called congestion factor b_{ijk}, which is a real number greater than 1. A higher congestion factor means more congestion, so a longer travel time. Also, the discrete departure time $t_{dep} \in \mathbb{N}^0$ is used, where $t_{dep} < T$. The time-dependent travel time functions are then defined as $\tau_{ij}(t_{dep}) = b_{ijk}\tau_{ij}$, where k such that $t_{dep} \in t_k$. The arrival time functions will be defined as $\alpha_{ij}(t_{dep}) = t_{dep} + \tau_{ij}(t_{dep})$, where $\alpha_{ij}(t_{dep})$ is the arrival time when the departure from customer i to customer j takes place at time t_{dep}. Adding the congestion to the CVRPTW results in a Time-Dependent CVRPTW (TDCVRPTW).

2.3.2 FIFO Property

The First-In-First-Out (FIFO) property is also known as the non-passing property since it holds when two vehicles traveling on the same arc will arrive at the destination in the same order as they left at the departure point. This property must hold for two reasons. Firstly, real-world traffic can often be seen as a queue, so this property also holds in reality. Without the FIFO property, useless waiting before departure may be beneficial in the model, as shown by Ichoua et al. (2003). Since this is never the case in real-world situations, it may lead to route schedules that are not attainable in practice. Secondly, this property restricts the number of possible travels. Waiting

before the travel is never beneficial, so the departure will be planned immediately after servicing the customer. For the FIFO property to hold, a few restrictions are introduced to the data. Firstly, all discontinuities of the function $\tau_{ij}(t)$ have a higher value after the discontinuity. Together with the restriction that the slope of the travel time functions $\tau_{ij}(t) > -1$ at all times, this guarantees the FIFO property. This leads to strictly increasing arrival time functions $\alpha_{ij}(t_{dep})$. An example of a travel time function $\tau_{ij}(t_{dep})$ where the FIFO property holds can be seen in Figure 1. Details on the data transformation that is used in this research to guarantee the FIFO property can be found in Section 9.1.

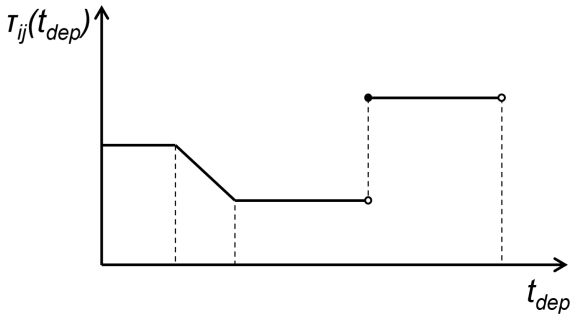


Figure 1: Example of a travel time function $\tau_{ij}(t_{dep})$ from customer i to customer j as a function of departure time t_{dep}.

2.3.3 Computational Complexity

Lemma 1. *The CVRPTW remains $\mathcal{NP}$-complete after adding TDTTs.*

Proof. A reduction by restriction is used to prove this statement. Let $\tau_{ij}(t_{dep}) = \tau_{ij}$, such that $\tilde{\tau}_{ijk} = 0$ for all $k \in K = \{1, ..., \mu\}$. Then this specific choice of parameters results in an instance of the TDCVRPTW without congestion and without time-dependency. Therefore, for this instance, the TDCVRPTW reduces to the CVRPTW. Since CVRPTW is $\mathcal{NP}$-hard (since it is $\mathcal{NP}$-complete), the TDCVRPTW is also $\mathcal{NP}$-hard by proof of reduction by restriction. Furthermore, CVRPTW is in $\mathcal{NP}$ since CVRPTW is $\mathcal{NP}$-complete. Time-dependency of travel times does not change the time complexity of checking if a possible solution is feasible; While iterating over the customer visits in the routes to check feasibility concerning to time windows, the appropriate travel time can be read in constant time when the TDTTs are for example stored in a three-dimensional matrix. Therefore, TDCVRPTW is in $\mathcal{NP}$ as well. Since TDCVRPTW is $\mathcal{NP}$-hard and in $\mathcal{NP}$, this proves TDCVRPTW is $\mathcal{NP}$-complete (unless $\mathcal{P} = \mathcal{NP}$). □

2.4 Break Scheduling

To show the potential of the HGSADC to solve practical problems, the specification of a feasible break should be commonly used in real-world problems. Firstly, this study focuses on the break specification of the working hours regulations, imposed by Directive 2002/15/EC by European Union (2002), since they are used in most countries in the European Union. Countries outside the

European Union often use the same structure of break regulations. European Union (2002) impose a break of a specific duration before a certain duration of working time. In practice, the break specification also includes a minimum duration of working time before starting the break. This results in a break window in which the break has to start. Formally, this results in the definition that feasible breaks have a specific duration Δt_B and start in a certain break window $[\Delta t_s, \Delta t_f]$. Δt_s and Δt_f are the start and finish time of the break window respectively, as measured from the start time of the route at the depot. A maximum of one break is added per route, and this break should only be scheduled when the route duration exceeds x. The maximum route duration, as also imposed by European Union (2002), is not taken into account, as this is mainly an implementation issue and will likely not change the conclusion about the potential of the HGSADC.

Driving hours regulations, imposed by European Community Regulation No 561/2006 by European Union (2006), are not taken into account, as they impose breaks with a similar structure as European Union (2002). The only significant difference is that breaks should start before a certain amount of driving time instead of working time (which also includes the service times). Since this is mainly an implementation issue, the driving hours regulations are not taken into account in this research. By only looking at the total working time instead of the driving time, the FIFO property still holds. When breaks should (also) be based on the accumulated driving time, it could be possible that waiting five minutes results in driving one minute less in the case of congestion. This may consequently lead to avoiding the need of scheduling a break, which may result in the FIFO property being violated.

In one specific situation, where the deliverer is strictly speaking not able to take a feasible break. This is the case when first taking a break will result in starting servicing the customer too late, while first servicing the customer will result in taking a break too late. This is an edge case of break scheduling that is considered feasible since the service times will be relatively small in many practical situations such as home delivery.

3 Background: Hybrid Genetic Search with Adaptive Diversity Control

The high-performance implementation of the Hybrid Genetic Search with Adaptive Diversity Control (HGSADC) from Kool et al. (2022) is explained in this chapter. It is the foundation this research is building on. Kool et al. (2022) added time window support from Vidal et al. (2013), a construction heuristic, crossover operator, and LS neighborhood to the HGS of Vidal et al. (2012).

3.1 The Problem to Solve in the DIMACS Challenge

The algorithm was created by Kool et al. (2022) to solve CVRPTWs, where the objective function is the sum of the route distances traveled by the vehicles. The distances between customers are Euclidean distances truncated to one decimal place. A non-restricting number of identical vehicles is given for each instance. The algorithm was scored on the Primal Integral, which is the area under the objective value versus time graph. This measure rewards finding quick solutions of good quality, as well as improving the solution in later stages.

3.2 Time Window Support using Time Warps

Time window support was included using the time warp principle of Vidal et al. (2013). When a vehicle arrives too late at a customer i, it "travels back in time" (the time warp) to e_i. An illustration of a time warp can be seen in Figure 2 at customer 3. Time t is on the horizontal axis and route distance is on the vertical axis. The blue horizontal lines are the time windows of the four customers, and the orange dotted lines are the travels of the vehicle between customers. The slope of an orange line is steeper when the vehicle travels at a higher speed.

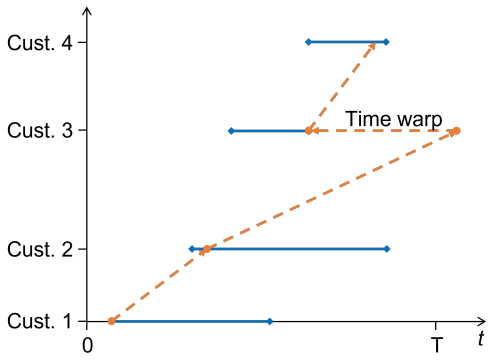


Figure 2: Illustration of a time warp at customer 3.

The total amount of time warp of a solution is multiplied by a penalty and added to the objective value. This time warp penalty is adjusted every 100 iterations, based on the fraction of feasible solutions with respect to time windows after LS. The penalty is increased/decreased when fewer/more solutions are feasible with respect to time windows after the LS.

3.3 Initial Population

To start with, 100 solutions are created to form the initial population. To start with a diverse population, 85% of the individuals are created by using the SPLIT algorithm, introduced by Prins (2004) and improved by Vidal (2015), on a random ordering of customers. To quickly find feasible solutions of relatively good quality, which is important to lower the Primal Integral as much as possible, three construction heuristics are used as well. Each heuristic accounts for 5% of the individuals in the initial population. Firstly, the construction heuristics "nearest" and "farthest" are used to create routes one by one by starting with the unassigned customer closest to and farthest away from the depot respectively. Then, unvisited customers are added to the optimal position in the route one at a time until no unvisited customers can be added without violations. Multiple (nearly feasible) solutions are constructed with this deterministic heuristic, by allowing for randomly selected, relatively small violations of the time windows and capacities. The third construction heuristic, called "sweep", constructs routes one by one by adding unvisited customers, sorted based on their angle with the depot, such that the capacity constraint is never violated. Then, the orders of the customers within the routes are changed based on arrival times and route distance. Time windows may be violated in this process. Multiple solutions are constructed by randomly reducing the capacity up to 40%.

3.4 Evolution of Generations

Both feasible and infeasible solutions are used in the HGSADC, where they are separated into two subpopulations. To create offspring, two parents, either feasible or infeasible, are selected by using a binary tournament. Each selected parent had the best fitness value of two randomly chosen solutions. The fitness value is based on both solution quality and diversity. The two selected parents will create a new solution using one of two possible crossovers. Firstly, an ordered crossover (OX) is used, which was introduced by Oliver et al. (1987). This procedure selects a subsequence of customers from the first parent and adds missing customers in the order of the second parent. Since depot visits are left out in this process, they are re-inserted by using the SPLIT algorithm. Since SPLIT does not take into account the time windows, any time window violations will be taken care of in the LS. Secondly, a Selective Route Exchange crossover (SREX) is used, which was introduced by Nagata and Kobayashi (2010). SREX randomly selects a set of routes from one parent and replaces them with a set of similar routes from the other parent. Missing nodes are inserted in the best position based on route distance. Customers that are visited twice are removed from one route. Since depot visits are not removed, the SPLIT algorithm is not needed and there is no need to fix time window violations. After adding the new offspring of both OX and SREX to the population, the best ω individuals are selected as a new generation. The value of ω is increased for every specific number of iterations.

3.5 Local Search

LS is used to improve the quality of all new offspring. Six different moves are considered:

RELOCATE: Relocate customer(s) to a different position in the same route. Complexity: $\mathcal{O}(n^2)$.

RELOCATE*: Relocate customer(s) to a different position in another route. Complexity: $\mathcal{O}(n^2)$.

SWAP: Swap two (pairs of) customers. Complexity: $\mathcal{O}(n^2)$.

SWAP*: Swap two (pairs of) customers from different routes without an insertion in place. So the selected customers are placed in the best position in the other route. Complexity: $\mathcal{O}(n^3)$.

2-OPT: Replace two arcs (u, x) and (v, y), where u and v are in the same route, with two arcs (u, v) and (x, y). Complexity: $\mathcal{O}(n^2)$.

2-OPT*: Replace two arcs (u, x) and (v, y), where u and v are in different routes, with two arcs (u, v) and (x, y). Complexity: $\mathcal{O}(n^2)$.

Each corresponding neighborhood can be explored in $\mathcal{O}(n^2)$, as shown by Vidal et al. (2013), Vidal (2022), and Kool et al. (2022). To search through the SWAP* neighborhood in $\mathcal{O}(n^2)$, the top three positions to insert a customer (only based on distance) are pre-computed for every route. This is a heuristic in case customers have time windows. For each move, only the Γ nearest neighbors are considered. This reduces the complexity of the neighborhoods. The value of Γ is based on the symmetric measure $\hat{\gamma}(u, v) = \min\{\gamma(u, v), \gamma(v, u)\}$, where $\gamma(u, v)$ is a measure taking into account spatial and time proximity. To reduce the running time even more, only pairs of routes with overlapping circle sectors are considered.

A penalty for violations of the capacity constraint is automatically adjusted such that at least a certain percentage of solutions from the LS satisfy this constraint. Also, the population size ω and the neighborhood size Γ are adjusted after every certain number of iterations. After a specific large number of iterations without improvement, the algorithm is restarted without resetting the parameters.

The evaluation of a move is discarded when the route(s) involved are feasible and the total route(s) distance after the move has not been reduced. This pre-check saves a significant number of move evaluations using the framework above, by skipping moves that will not improve the solution.

3.6 Efficient Move Evaluations in the Local Search

The framework that is implemented by Kool et al. (2022) and largely inspired by Vidal et al. (2013) allows for evaluating concatenations of subroutes in constant time. This is the heart of the HGSADC, where most of the running time is spent. Properties are defined for each subroute σ, where a subroute is a subsequence of a possibly non-feasible route. The four properties of a subroute σ are the minimum duration $D(\sigma)$, the minimum time warp $TW(\sigma)$, and the earliest and latest time of departure at the first visit ($E(\sigma)$ and $L(\sigma)$ respectively) allowing a schedule with duration $D(\sigma)$ and time warp $TW(\sigma)$. For a subroute σ^0 consisting of only one visit, the above-mentioned properties are initialized as $D(\sigma^0) = \tau_i, TW(\sigma^0) = 0, E(\sigma^0) = e_i$, and $L(\sigma^0) = l_i$.

Let $\sigma = (\sigma_i, \ldots, \sigma_j)$ and $\sigma' = \left(\sigma'_{i'}, \ldots, \sigma'_{j'}\right)$ be two subroutes consisting of depot- and customer visits. Then, for the concatenation $\sigma \oplus \sigma'$ it holds that:

$$D\left(\sigma \oplus \sigma'\right) = D(\sigma) + D\left(\sigma'\right) + \delta_{\sigma_j \sigma'_{i'}} + \Delta_{WT} \tag{3}$$

$$TW\left(\sigma \oplus \sigma'\right) = TW(\sigma) + TW\left(\sigma'\right) + \Delta_{TW} \tag{4}$$

$$E\left(\sigma \oplus \sigma'\right) = \max\left\{E\left(\sigma'\right) - \Delta - \Delta_{WT}, E(\sigma)\right\} \tag{5}$$

$$L\left(\sigma \oplus \sigma'\right) = \min\left\{L\left(\sigma'\right) - \Delta + \Delta_{TW}, L(\sigma)\right\} \tag{6}$$

where

$$\Delta = D(\sigma) - TW(\sigma) + \delta_{\sigma_j \sigma'_{i'}} \tag{7}$$

$$\Delta_{WT} = \max\left\{E\left(\sigma'\right) - \Delta - L(\sigma), 0\right\} \tag{8}$$

$$\Delta_{TW} = \max\left\{E(\sigma) + \Delta - L\left(\sigma'\right), 0\right\} \tag{9}$$

This framework considers instances where the travel duration is used as travel distance. Therefore, the travel durations $\delta_{\sigma_j \sigma'_{i'}}$ in Equation (7) have the values of the travel distances $c_{\sigma_j \sigma'_{i'}}$. This equation considers Δ, which is the travel duration up to the arrival at $\sigma'_{i'}$ (the first visit of the second subroute) excluding any possible time warp. Next, Equation (8) calculates Δ_{WT}, the minimum waiting time at $\sigma'_{i'}$. If the latest arrival at $\sigma'_{i'}$ (calculated by $L(\sigma) + \Delta$) is before the earliest start of σ', the difference between the two is the added waiting time Δ_{WT}. In a similar way, Equation (9) calculates Δ_{TW}, the minimum time warp at $\sigma'_{i'}$. If the earliest arrival at $\sigma'_{i'}$ (calculated by $E(\sigma) + \Delta$) is after the latest start of σ', the difference between the two is the added time warp Δ_{TW}. Those three equations are used to calculate the properties of $\sigma \oplus \sigma'$ in Equations (3)-(6). Equation (3) shows that the route duration after concatenation is calculated by summing the route durations of the two subroutes, the duration of the travel from σ_j to $\sigma'_{i'}$ and the minimum additional waiting time introduced at visit $\sigma'_{i'}$. Again, the travel durations $\delta_{\sigma_j \sigma'_{i'}}$ have the values of the travel distances $c_{\sigma_j \sigma'_{i'}}$ here. The time warp of $\sigma \oplus \sigma'$ is calculated in Equation (4) by summing the time warps of the two subroutes σ and σ' and adding the minimum additional time warp introduced at visit $\sigma'_{i'}$. $E\left(\sigma \oplus \sigma'\right)$ and $L\left(\sigma \oplus \sigma'\right)$ equal $E(\sigma)$ and $L(\sigma)$, but are made more strict when the backward calculation from $\sigma'_{i'}$ gives a later or earlier starting time of $\sigma \oplus \sigma'$ respectively.

(Sub)route distances are evaluated depending on the type of move considered, and are left out of this framework. Compared to Vidal et al. (2013), some equations of the framework have been rewritten for clarity. Non-trivial equations of the framework above are proven by Vidal et al. (2013).

Prior to each LS, the properties of subsequences that start or end at the depot, called the prefix and suffix subsequences respectively, are pre-processed and cached. This allows for constant time evaluations of an insertion or removal of a customer. Consequently, feasibility checks and objective value calculations can be performed in constant time to evaluate moves in the LS. By also pre-computing and caching the properties of subsequences between one of every four customers, the so-called seed customers, simultaneous insertions, and removals can be evaluated efficiently.

4 Literature Review

Firstly, a survey is given in Section 4.1 of the different techniques for solving the CVRPTW[b]. In Section 4.2, an overview is given of the research done on the three practical requirements that are studied in this research. This will show the gap in research filled by this thesis, and the current state-of-the-art methods dealing with the three practical requirements.

4.1 Techniques for Solving Vehicle Routing Problems with Time Windows

Finding the exact solution for the CVRPTW was first done by Kolen et al. (1987). Using a branch-and-bound method, up to several minutes of CPU time were needed to solve instances of up to 15 customers. A breakthrough algorithm by Desrochers et al. (1992) was capable of optimally solving several 100-customer problems. However, this often required hours of running time, if the optimal solution was found at all. This illustrates why heuristic-, and metaheuristic approaches are often used to solve (the majority of practical) CVRPTWs.

Firstly, trajectory-based metaheuristics such as Tabu Search have been used with several specialized diversification and intensification mechanisms. For example, Moccia et al. (2012) use a general Tabu Search framework, but it takes a relatively long time to solve relatively small instances. Secondly, population-based metaheuristics such as Particle Swarm Optimization, Ant Colony Optimization, and Artificial Bee Colony Optimization are shown to be great methods to solve CVRPs by Marinakis et al. (2019), Ding et al. (2012), and Mortada and Yusof (2021) respectively. Genetic Algorithms (GAs) became effective when the Giant Tour representation was introduced by Prins (2004). This chromosome representation omits travels to and from the depot. Individual routes are retrieved from a chromosome by the Split algorithm in $\mathcal{O}(n^2)$ using Bellman's algorithm. Vidal (2015) later proposed a faster Split algorithm that runs in $\mathcal{O}(n)$. After the introduction of a general HGSADC algorithm for the CVRP by Vidal et al. (2012), this algorithm was extended with time window support by Vidal et al. (2013). The resulting HGSADC could be used to solve a large class of CVRPTWs, including Multiple Depot-, Periodic- and, (Vehicle-)Site Dependent CVRPTWs. Especially Marinakis et al. (2019) show that the HGSADC outperforms all previously mentioned population-based metaheuristics on CVRPTW instances where $n > 100$ and where only the distance is used in the objective. Only the Hybrid Genetic Algorithm (HGA) of Nagata et al. (2010) is similar to the HGSADC in terms of solution quality for this type of CVRPTWs. However, this HGA relies on more problem-tailored components, which complicates extending the model. Recently, Vidal (2022) added a new SWAP* neighborhood to the HGSADC. On classical CVRP literature benchmarks, this algorithm outperforms current state-of-the-art approaches[c]. Kool et al. (2022) extended the open-source code of the HGSADC by Vidal (2022) with time window support

[b] The two frameworks by Current and Marsh (1993) and Eksioglu et al. (2009) can be read for an overview of the broader class of Routing Problems. More information on the different variants of the CVRP can be found in the taxonomy of Lahyani et al. (2015).

[c] Namely, the Hybrid Iterated LS by Subramanian et al. (2013), the Lin–Kernighan–Helsgaun heuristic by Helsgaun (2017), the Knowledge Guided LS by Arnold and Sörensen (2019), the Slack Induction by String Removals by Christiaens and Vanden Berghe (2020) and the Fast Iterative Localized Optimization by Accorsi and Vigo (2021).

inspired by Vidal et al. (2013). Details of this extended HGSADC can be found in Chapter 3. In conclusion, the HGSADC is a current state-of-the-art method to solve CVRPTWs in terms of solution quality and model flexibility.

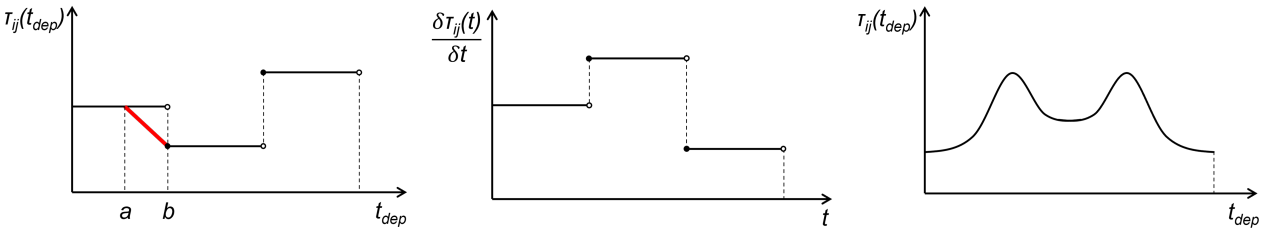
4.2 Incorporation of Practical Requirements

4.2.1 Route Duration Minimization

Firstly, a hierarchical objective function can be used to score route schedules based on multiple variables, as done by Conrad and Figliozzi (2011). In general, the primary objective is the number of routes. Then, for the minimal number of routes, the second objective is to minimize the total route distance or duration. However, there is no strict order for minimizing the total route distance and duration in this research. Therefore, a weighted sum can be used in the same way as in the GA by Ombuki et al. (2006). A third option is to use a Pareto ranking technique, which is also implemented by Ombuki et al. (2006). An advantage is that there is no bias introduced by poorly chosen weights in the objective since the criteria of the objective are treated as separate entities.

4.2.2 Congestion

The earliest papers that consider congestion, such as the research by Malandraki and Daskin (1992), model the travel time as a discrete function of departure time. The FIFO property is not guaranteed to hold, as can be seen in Figure 3a. Here, the FIFO property does not hold when the departure is in the interval (a, b) since a later departure at time b will result in an earlier arrival. The bold, red, diagonal line with a slope greater than -1 can be used in the interval (a, b) to guarantee the FIFO property. In later research, models are often used where the travel speed is a discrete function of time[d]. An example of this model, which was proposed by Ichoua et al. (2003), can be seen in Figure 3b. Setak et al. (2015) later proposed a procedure to transform this travel speed function into a continuous, FIFO travel time function of time. Thirdly, the travel time can be modeled as a differentiable function of the departure time, as done by Haghani and Jung (2005). An example can be seen in Figure 3c. Although this is the closest to real-life congestion, large amounts of data need to be stored for this model to capture congestion for real-world problems.



(a) Discrete travel time function of departure time t_{dep}.

(b) Discrete travel speed function of time t.

(c) Continuous travel time function of departure time t_{dep}.

Figure 3: Three different ways of incorporating congestion.

[d] Note the difference between time t and departure time t_{dep} here.

After describing different ways of modeling congestion, the focus is now on different state-of-the-art metaheuristics[e], specifically neighborhood searches, that improve solutions of TDCVRPTWs. For example, Hashimoto et al. (2008) used an Iterated LS algorithm that incorporates a Dynamic Programming (DP) algorithm to determine route schedules. Information from previous iterations is used for the next iteration of the DP algorithm to speed up computation. Routes are evaluated in pseudo-polynomial time. Next, Balseiro et al. (2011) introduced an Ant Colony algorithm to solve TDCVRPTWs. High-quality solutions are found in a reasonable amount of time for academic benchmark instances. The evaluation of moves is a computationally expensive procedure since travel times are propagated to the end of the route. To reduce running time, only moves with customers in a newly defined neighborhood are evaluated. Skipping move evaluations of the other customers likely leads to suboptimal solutions. Both methods do not take into account break scheduling.

Using neighborhood searches, it is crucial to be able to quickly check feasibilities and calculate solution values after applying a move. As shown by Vidal et al. (2015), feasibilities can be checked in $\mathcal{O}(n)$ for the TDCVRPTWs without total route duration in the objective, as well as for the constant travel time CVRPTWs where route duration is in the objective. Recently, Visser and Spliet (2020) proposed new methods to efficiently evaluate moves in neighborhood search heuristics for TDCVRPTWs with total route duration in the objective. Compositions of piecewise linear arrival time functions with a maximum of p breakpoints are calculated efficiently to allow for insertion- and exchange move evaluations in $\mathcal{O}(np)$. To the best of my knowledge, no method exists that can evaluate moves on feasibility and solution values for the TDCVRPTW with total route duration in the objective, using sub-pseudo-polynomial time.

4.2.3 Break Scheduling

Incorporating the driving hours regulations by European Union (2006) into a model was first done by Goel and Kok (2009). A depth-first-breadth-second technique was used to find a feasible route schedule in $\mathcal{O}(n^2)$ time if such a schedule exists. The same regulations were also used by Goel (2009), who proposed a large neighborhood search heuristic. A labeling algorithm is used to evaluate moves in the neighborhood. Kok et al. (2010a) improved the results by using a break scheduling algorithm within a DP heuristic, where the running time is still $\mathcal{O}(n^2)$. Taking into account both the driving periods by European Union (2006) and working periods by European Union (2002) was first done by Kok et al. (2010b) who used a DP approach to incorporate all the regulations in the TDCVRPTW. By combining congestion and break scheduling in one model, move evaluations in the LS become increasingly expensive. This is because moves have increasingly complex effects on the solution value and feasibility of the routes involved. This can for example be seen in the time-complexity of the DP algorithm by Kok et al. (2010b), which runs in $\mathcal{O}(n^{j+1}p)$. Here, j is the number of breaks to be scheduled, and p is the maximum number of breakpoints in the travel time function. Since p could be very large, especially for routes with many customers, this algorithm will likely be too slow

[e]Throughout time, exact methods have been used to solve TDCVRPTWs, but even recent studies, such as Dabia et al. (2013), are only able to solve instances with up to 100 customers to optimality. Therefore, the focus is again on studies that use metaheuristics to find good-quality solutions in a reasonable amount of time.

to schedule breaks in the LS in the HGSADC. Another method is to schedule breaks after solving a TDCVRPTW, for which the ILP model of Kok et al. (2011) could be used. However, this model again takes too much time to evaluate each move in the LS of the HGSADC. Prescott-Gagnon et al. (2010) incorporated both sets of regulations in the CVRPTW using a large neighborhood search method where all feasibility rules are modeled as resource constraints. This method outperformed the DP method of Kok et al. (2010a) when only a subset of the regulations was considered. Still, it takes between 10 and 90 minutes to find solutions for instances with just 100 customers.

Recently, Rincon-Garcia et al. (2020) proposed a state-of-the-art metaheuristic for the TD-CVRPTW including driving hours regulations. A "Scheduler" algorithm incorporates the driving hours regulations into a large neighborhood search that uses both destruction and recreation procedures. Although this algorithm can find good-quality solutions for practical instances, it needs about 36 minutes to generate a solution for problems with 200 customers.

In conclusion, no algorithm exists that can obtain good-quality solutions in a reasonable amount of time for practical problems that incorporate the selection of practical requirements of this research.

5 Methodology

This chapter starts by introducing a multi-objective function to the HGSADC in Section 5.1. Then, two different methods are introduced to add practical constraints to the HGSADC. Firstly, in Section 5.2, an Iterative Method (IM) is introduced, which uses an elegant framework that can be applied to most practical constraints. Secondly, in Section 5.3, the more specialized Local Search Method (LSM) is proposed, where the LS actively deals with constraints in a sophisticated way. The two selected practical constraints are added to the HGSADC using both the IM and LSM to demonstrate the implementation and to show the potential of both methods in Chapter 7.

5.1 Multi-Objective Function

5.1.1 General Concept

In practice, the weights of the objective function are often known prior to running the algorithm. This is why a weighted sum is used as the objective function instead of a Pareto Search or hierarchical objective function. For some objective functions, the framework of Equations (3)-(9) has to be extended to calculate the value of specific objectives. For example, when introducing a cost per route in the objective function, LS moves that add or remove a route will increase or decrease the objective function respectively. Most practical objective functions can be incorporated without significant running time increases. However, the time-saving pre-checks do not work anymore, as explained when adding Route Duration Minimization later.

5.1.2 Application: Route Duration Minimization

In this research, Route Duration Minimization has to be added to the distance-only objective. This results in the objective function given by Equation (2). To calculate values of this objective function efficiently in the LS, the framework of Equations (3)-(9) is extended[f] with the calculation of the cumulated distance $C\left(\sigma \oplus \sigma'\right)$ of the concatenation of two subroutes σ and σ'

$$C\left(\sigma \oplus \sigma'\right) = C(\sigma) + C\left(\sigma'\right) + c_{\sigma_j \sigma_{i'}} \tag{10}$$

where the cumulated distance after concatenation is calculated by summing the distances of the two subroutes σ and σ' and the distance of the travel from σ_j (the last visit of the first subroute) to $\sigma_{i'}$ (the first visit of the second subroute). For all subroutes σ^0 consisting of only one visit, $C\left(\sigma^0\right) = 0$.

Because of the differentiation between travel distance and travel duration, $\delta_{\sigma_j \sigma'_{i'}}$ now differ from $c_{\sigma_j \sigma'_{i'}}$ in Equations (3) and (7). This has a negative effect on the time-saving pre-checks, introduced by Kool et al. (2022), that were used to prevent unnecessary move evaluations. Namely, checking if the objective value decreases now requires evaluating Equations (3), (7), (8), and (10), instead of just Equation (10). Taking into account that pre-checks take time to evaluate as well, and will

[f] Equation (10) makes the framework equivalent to the framework by Vidal et al. (2013), although some equations of the framework have been rewritten for clarity.

only save running time for cases where move evaluations are prevented, pre-checks are removed for this method.

5.2 Constraints: Iterative Method

5.2.1 General Concept

In this IM, constraints are only added to the Genetic Algorithm (GA), while the LS deals with a relaxation of the problem. Instance data is changed iteratively between LS phases when the solution is unfeasible with respect to the full constrained problem. It is crucial to construct a procedure where solutions converge to feasibility in the process of iterating between the GA and the LS. As this is a very general framework, this technique could be implemented for most practical constraints, although suboptimal procedures may be necessary to guarantee convergence to feasible solutions. A disadvantage is that the quality of the solutions returned by the LS may be relatively low since the LS is unaware of the added constraints. However, the technique of concatenating subroutes, used for move evaluations in the LS, does not have to change significantly. This limits the speed reduction of the algorithm.

5.2.2 Application: Congestion and Break Scheduling

Congestion and break scheduling will now be added to the HGSADC in an elegant way using only a single framework. The analogy will be used that both congestion and break scheduling could be incorporated into a route by increasing either the travel times between customers or service times at customers. The amount of additional time needed for congestion or break scheduling is time-dependent for both constraints.

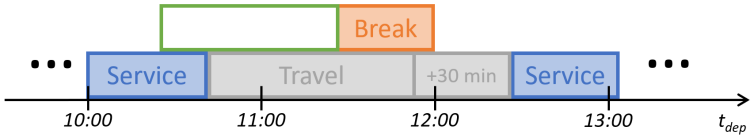
Congestion will be integrated by changing all travel times in routes with time warps. Those travel times will be updated such that they include the congestion concerned. The travel times will only be increased, such that convergence is guaranteed.

Breaks are scheduled using a heuristic; All breaks are scheduled as late as possible, except when a waiting time (independent of its duration) is encountered in the break window. The quality of this heuristic depends on the properties of the data, such as the width of the time windows of the customers. Optimal break scheduling would not increase the running time significantly in this case of scheduling only one break. However, when scheduling multiple breaks per route, an optimal method such as Dynamic Programming will take significantly more running time, as shown for example by Kok et al. (2010b). Therefore, this heuristic is used to schedule breaks in order to keep the results as general as possible.

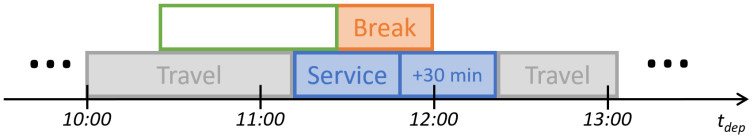
At the time a break is scheduled to start, one of three possible activities is being performed. Firstly, the break might start during a travel. In this case, the travel time of the concerned travel is increased by the total break duration, as can be seen in Figure 4a. Secondly, the break might be scheduled to start during service at a customer. In that event, the service time of the specific customer is increased by the total break duration, as can be seen in Figure 4b. Increasing the service

time is only possible for break scheduling, not for including congestion. Thirdly, the break might be scheduled to start during a waiting time at a customer. Then, the break uses as much waiting time as possible. Any remaining break duration is added to the service time of the next customer, as can be seen in Figure 4c. In all three figures, it can be seen that a similar framework is used as for congestion. Namely, times are increased such that a break of sufficient duration can be started in the break window. Again, travel- and service times will only be increased, such that convergence is guaranteed.

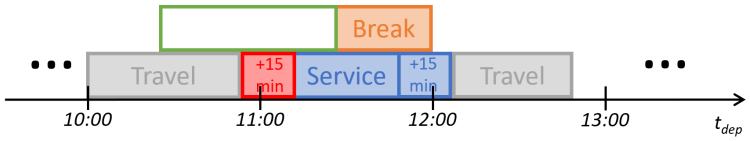
There is one edge case where this heuristic results in a solution where the break is scheduled such that it is impossible to respect both the customer's time window and the deliverer's break window. This is the case where a break is scheduled during a service time while the service time ends after the break window. In this case, the deliverer has to choose whether to service the customer first and start its break too late, or take the break first and start servicing the customer too late. This will not be a problem for home delivery problems, where service durations are relatively short and the drivers could just take their break after delivering the service. In practice, drivers are deciding when to take a break, and the route schedule should only provide the driver with time to take a break. Therefore, similar to ORTEC's algorithm, this is seen as a feasible break scheduling.



(a) The break extends the travel time by 30 minutes.



(b) The break extends the service time by 30 minutes.



(c) The break is partly in the waiting time (added to the first travel time) and partly extends the service time.

Figure 4: Three scenarios of break scheduling, where the break window is indicated by a green box.

The introduced framework uses uncongested travel times and the original service times in the first LS phase. For the solution after this first LS phase, two types of feasibility will be checked. Firstly, the so-called *basic feasibility* with respect to the unconstrained problem. Secondly, the *constraint feasibility* using the real-time, congested travel times and the break regulations. If the solution is both basic feasible and constraint feasible, the individual will be added to the pool of feasible solutions, the best-known solution will be updated, and the improvement process for this individual

will stop. In case the solution is only basic feasible, the travel- and service times will be updated[g] and the (higher) travel- and service times are used as new constant travel- and service times in the next LS phase. When the travel- and service times have been updated more than a certain number of times, the solution will be added to the pool of infeasible solutions and the improvement process for this individual will stop. If the solution is not basic feasible, both penalties for time warp and capacity excess will be increased[h], after which a new LS phase starts. When the penalties have been increased more than a certain number of times, the solution will be added to the pool of infeasible solutions and the improvement process for this individual will stop. For each individual, this iterative procedure will end with resetting penalties, and travel- and service times to their original values. The pseudocode for this algorithm can be seen in Algorithm 2 in Section 9.2.

An advantage of this implementation is the possibility to efficiently include route duration in the evaluation of moves in the LS. Another advantage of this method is that, although this research focuses on one break, this method can be easily extended to adding multiple breaks without increasing the time complexity. A disadvantage of this method is that the congestion will likely be over-estimated since travel times will never be decreased. This is also the case for break scheduling. Also, to create feasible routes, all vehicles start their route such that their arrival at the start of the service window of the first customer, so there is no start time optimization.

5.3 Constraints: Local Search Method

5.3.1 General Concept

In this LSM, the LS actively deals with constraints in a sophisticated way. The technique of concatenating subroutes, used for move evaluations in the LS, has to change drastically and this will slow down the algorithm. Heuristics, estimations, and/or bounds will be needed to limit the slowdown of the LS. However, depending on the use of the three above-mentioned techniques, the quality of the solutions returned by the LS will be higher than for the IM since the LS now directly deals with the constraints. As this is not a general framework, but rather a specialized method, it is likely that it will take a greater effort for a constraint to be incorporated in the HGSADC using this LSM. Also, adding more constraints will probably be increasingly difficult since each constraint needs to be incorporated in its own specialized way in a more and more specialized algorithm.

5.3.2 Application: Congestion

To actively take into account congestion in the LS, calculating the objective value for every possible move would slow down the algorithm too much. This is because removing/adding a customer from/to a route affects all other travel times[i]. Therefore, Lower Bounds (LBs) and Upper Bounds (UBs) on all travel times are used to make estimations on the objective value after moves in the

[g] Namely, for all routes with a time warp, all travel times will be updated to the real congested travel times. Also, new breaks are scheduled using the procedure of Figure 4 for all routes violating a break regulation.

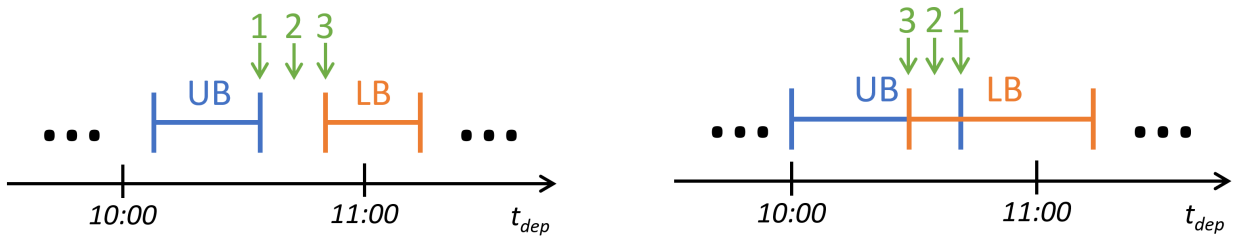
[h] Penalties will be multiplied by 10, similar to Kool et al. (2022)

[i] If the start of the route is fixed, only travel times of travels after the point of removal/addition will change.

LS. Instead of using one evaluation of a move, as in the IM, two evaluations are used; One using the LBs, and one using the UBs on the travel times. The objective value when using the LBs is always as least as low as when using the UBs. The two bounds on the objective value are then used to give an estimation of the objective value after the move. This estimation is given by

$$Z_{est}(s) = \gamma Z_{LB}(s) + (1-\gamma) Z_{UB}(s) \tag{11}$$

where $Z_{est}(s)$ is the estimated objective value[j] using $Z_{LB}(s)$ and $Z_{UB}(s)$, which are the objective value of solution s using the bounds on the travel times respectively. γ is a parameter that can be used to give preference to either $Z_{LB}(s)$ or $Z_{UB}(s)$. If $Z_{est}(s)$ is smaller than the current objective, the objective value is calculated with the real congestion, using the three starting times at the depot that can be seen in Figure 5: The latest departure from Equation (6) for the evaluation with the UBs (green arrow 1), the earliest departure from Equation (5) for the evaluation with the LBs (green arrow 3), and the average between the two (green arrow 2). Those start times are chosen because departing earlier/later for the evaluation with the UBs/LBs will never be beneficial.



(a) Non-overlapping intervals.

(b) Overlapping intervals.

Figure 5: Start time intervals of a route calculated with LBs (orange) or UBs (blue) on the travel times. The green arrows point to the start times used for route evaluation with the real congestion.

The start time that gives the lowest objective is the best start time. If the corresponding lowest objective improves the current objective, the move is performed and the start time is saved for that route. Otherwise, the move is canceled.

The lowest and highest congestion factors are cached at the start of the algorithm to use the bounds on the travel time efficiently. For every possible combination of two customers and a time interval, the lowest and highest congestion factor is saved for all ranges from that time interval to all later time intervals. To bring the bounds on the travel times as close as possible, only travel times of feasible departure times are taken into account when calculating $Z_{LB}(s)$ and $Z_{UB}(s)$.

As opposed to the IM, there is travel time optimization in the LSM. Another advantage is that travel times are not increased for the LSM. However, an iteration of the LS will take more time.

[j] For move evaluations, only the change in $Z(s)$ has to be calculated for routes in s that are affected by the move.

5.3.3 Application: Break Scheduling

To actively schedule breaks in the LS, an important observation is made. In the HGSADCs by Vidal et al. (2013) and Kool et al. (2022), routes were evaluated by concatenating subroutes in constant time. The resulting route was independent of the order of concatenation of its ordered subroutes. For example, to construct $\sigma \oplus \sigma' \oplus \widetilde{\sigma}$, a route consisting of the subroutes σ, σ', and $\widetilde{\sigma}$ in this order, it holds that $(\sigma \oplus \sigma') \oplus \widetilde{\sigma} = \sigma \oplus (\sigma' \oplus \widetilde{\sigma})$. This property is now used to fix the order of concatenation to a chronological order. As an example, take the situation where a move of one customer to another position in the same route needs to be evaluated. The four subroutes that need to be concatenated for the evaluation of this move can be seen in Figure 6.

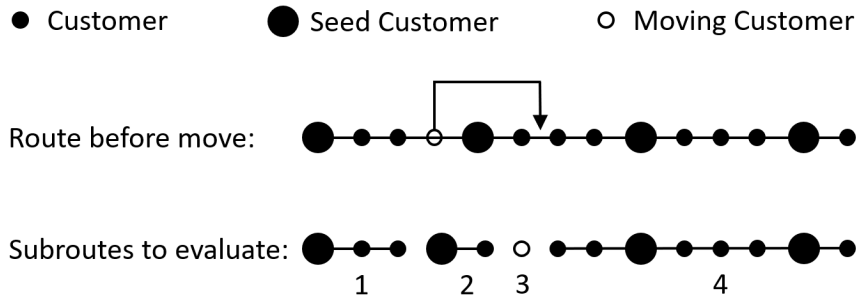


Figure 6: Schematic illustration of the four subroutes that need to be concatenated for a move evaluation where one customer is moved to another position in the same route. Subroutes are numbered. The arrow indicates the change in position of the moving customer.

If subroutes 3 and 4 are combined first, the route duration up to the start of subroute 3 is still unknown. Therefore, there is no indication of the location of the break window and it is impossible to take break scheduling into account during this concatenation. When the subroutes are combined chronologically, the first subroute of each concatenation always starts at the depot. Therefore, without slowing down the algorithm, the duration of the route can now always be calculated during the process of concatenating subroutes, using Equation (3). Constraints that are dependent on the departure time of the vehicle can now be evaluated with each concatenation. This is essential for checking if the break window has been reached.

At the start of the LS for a specific solution, breaks are scheduled in the routes of that solution. By the same arguments as for the IM, a heuristic is used for break scheduling in the LSM. However, breaks are always scheduled as late as possible now, independent of waiting time earlier in the break window. This is because later breaks will lead to more fast concatenations, as shown later. During this initial break scheduling, a Boolean $BREAK\_INCLUDED$ is cached for each possible subroute starting from the depot, indicating if a break is included in that subroute. After this pre-processing, the LS starts to concatenate subroutes in the process of evaluating moves. For every two subroutes that have to be concatenated, the framework of Equations (3)-(10) is used first to concatenate without scheduling breaks. Then, a check is performed if the resulting concatenation is valid with respect to break scheduling. The result is not valid when the break window went by completely

but no break was scheduled[k]. In that case, the concatenation is performed differently. This time, customers of the second subroute are added individually to the first subroute, to find the position where the break has to be scheduled. Caches of subroute information between seed customers are used to speed up this process. Scheduling breaks as late as possible, also when a waiting time is encountered earlier in the break window, leads to more valid concatenation. This will limit the number of subroutes where customers of the second subroute need to be added individually to the first subroute, thus saving running time. When the single customer is found where the break should have been scheduled during its concatenation, that specific customer will be concatenated to the first subroute using the constant time procedure that can be seen in Algorithm 1.

Algorithm 1: concatenateWithBreak(σ, σ', PARAMS)

Result: concatenation $\sigma \oplus \sigma'$ including a new break

Input:
σ, the first subroute, starting from the depot;
σ', the second subroute, consisting of one customer;
PARAMS, stores values of Δt_B, Δt_s, and Δt_f;

Initialization:
DELTA_BREAK_INCLUDED $\leftarrow$ **false**;

Iterative Process:
calculate Δ using Equation (7);
if $\Delta > \Delta t_f$ **then**
  increase $\delta_{\sigma_j \sigma'_{i'}}$ by Δt_B and recalculate Δ using Equation (7);
  DELTA_BREAK_INCLUDED $\leftarrow$ **true**;
end
calculate Δ_{WT} using Equation (8);
calculate Δ_{TW} using Equation (9);
if $\Delta + \Delta_{WT} - \Delta_{TW} + D(\sigma') > \Delta t_f$ **and not** *DELTA_BREAK_INCLUDED* **then**
  DELTA_START_BREAK $\leftarrow \max\{\Delta, \Delta t_s\}$;
  DELTA_WT_AVAILABLE $\leftarrow \max\{0, E(\sigma') - DELTA\_START\_BREAK - L(\sigma)\}$;
  BREAK_DURING_WT $\leftarrow \min\{DELTA\_WT\_AVAILABLE, \Delta t_B\}$;
  BREAK_DURING_SERVICE $\leftarrow \Delta t_B$ - BREAK_DURING_WT;
  increase $\delta_{\sigma_j \sigma'_{i'}}$ by BREAK_DURING_SERVICE and recalculate Δ using Equation (7);
  DELTA_BREAK_INCLUDED $\leftarrow$ **true**;
end
calculate $C(\sigma \oplus \sigma')$, $E(\sigma \oplus \sigma')$, $L(\sigma \oplus \sigma')$, and $TW(\sigma \oplus \sigma')$ using Equations (4)-(6), (10);
BREAK_INCLUDED $\leftarrow$ (BREAK_INCLUDED **or** DELTA_BREAK_INCLUDED);

[k] This check is performed in constant time using the Boolean $BREAK\_INCLUDED$ and the route duration from Equation (3).

6 Data Description

To show the potential of using the HGSADC to solve practical problems, ORTEC provided real-life data from a major grocery chain in the USA. The 222 selected instances can be divided into three, equally sized groups based on their number of customers. These groups are used in the Computational Experiments. Group 1, 2 and 3 consider the instances where $200 \leq n < 275$, $275 \leq n < 350$, and $350 \leq n \leq 880$ respectively. The distribution of the instances over the groups can be found in Figure 7. Every customer of every instance is assigned to a representative. Representatives cluster customers with similar congestion characteristics to limit the number of congestion factors b_{ijk}. Namely, $b_{ijk} = b_{i'j'k}$ when i and i' belong to the same representative and j and j' do as well. Information on the transformation of this data such that the FIFO property holds can be found in Section 9.1. The relationship between the number of available vehicles and the number of customers can be seen in Figure 8. For each instance, the number of routes is calculated by $\lfloor \frac{\sum_{i=1}^{n} q_i}{Q} \cdot 1.1 + 1 \rfloor$, which might result in some infeasible problems, especially when considering breaks and congestion. Based on the number of customers and available vehicles, routes visit on average 14.2 customers in the range [6.7, 21.5]. Based on the vehicle capacities and the customer demands, available vehicles are on average filled for 62.2% of their capacity, in the range [29.8%, 94.5%].

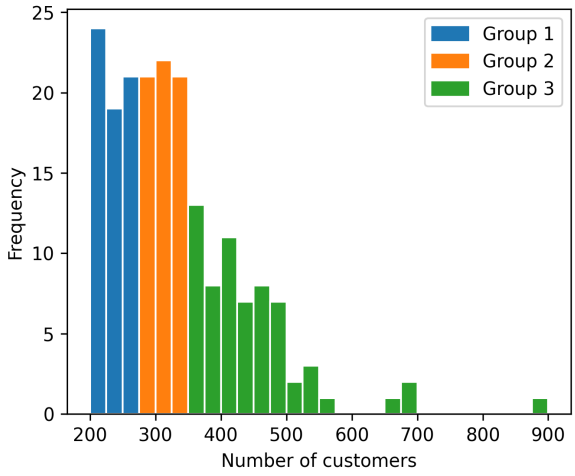


Figure 7: Distribution of the number of customers for the three groups.

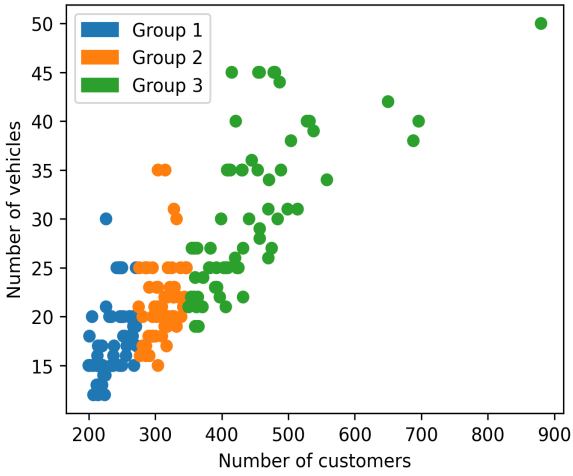


Figure 8: Relationship between the number of vehicles and the number of customers.

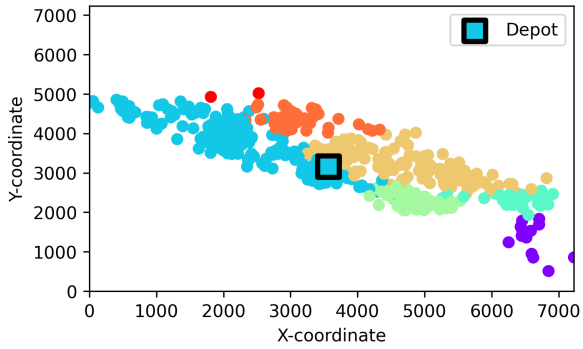
Without loss of generality, this research focuses on intra-day route scheduling since route scheduling is often done once a day in practice. In the specific time horizon, each pair of representatives has a so-called congestion profile, which is the change of b_{ijk} over time. The congestion factors b_{ijk} are stored efficiently[l] for every 15-minute time interval. This data is not necessarily symmetric, as can be seen in the example in Figure 9d. The lowest and highest values for b_{ijk} over all instances are 1.0 and 2.44 respectively. The range, mean and average of other important instance properties can be seen in Table 1. Here, the depot is filtered from the data, to only show customer statistics.

[l] Rounded values of $\frac{100}{b_{ijk}}$ between 0 (infinite congestion) and 100 (no congestion) are stored as integers .

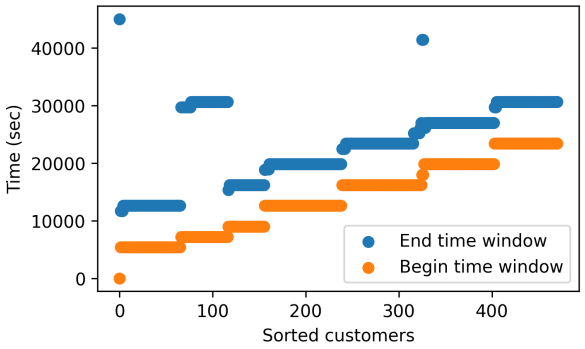
Table 1: Ranges, means, and averages of instance properties.

	Range	**Mean**	**Median**
# Customers	$[200, 880]$	331.2	313.5
# Representatives	$[1, 12]$	3.0	3.0
# Vehicles	$[12, 50]$	23.3	21.0
Demand	$[1, 100]$	6.0	6.0
Capacity	$[125, 200]$	147.3	145.0
Service duration	$[120, 5580]$	577.0	540.0
Time window duration	$[3300, 24000]$	9907.0	7500.0

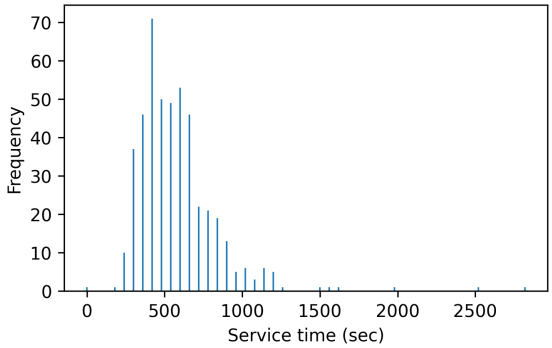
A diverse group of instances is selected to evaluate the algorithms on general, practical CVRPTWs. For example, the depot location is in between the customers for some instances, and further away from all customers for other instances. Also, some instances only consider the afternoon such that there is only one peak in the congestion profile. An example of an instance where the depot is located in between the customers, and the time horizon considers two congestion peaks, can be seen in Figure 9. For each instance, there are groups of customers sharing the same time window. However, some customers have a unique time window, as can be seen in the example in Figure 9b. For the service times, only specific values are used, as can be seen in the example in Figure 9c.



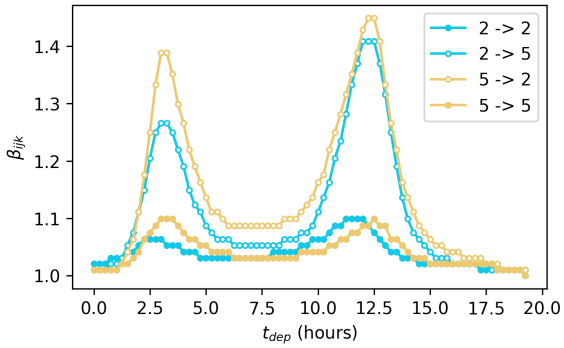
(a) Map of customers, colored by the representative.



(b) Sorted time windows.



(c) Sorted service times.



(d) Congestion profiles for two representatives.

Figure 9: Illustration of most important properties of one specific instance with 470 customers, 27 available vehicles, and 7 representatives.

7 Computational Experiments

In this chapter, the results are presented of the experiments that are conducted to show the potential of the HGSADC to solve practical problems. All methods are implemented in C++. All instances are run in parallel, using a single thread per instance. Multiple machines are used (with 2 up to 16 cores and 4 up to 32 GB of RAM) with 3^{rd} Generation Intel® Xeon® Platinum 8370C's with 3.4 GHz (turbo frequency of 3.7 GHz). The running time is measured from the start of the algorithm using `std::clock()`.

This chapter starts with choosing and tuning the values of the most important parameters in Section 7.1. Using the tuned parameters, experiments were conducted to assess the performance of the introduced methods. The results of the implementation of the multi-objective function can be found in Section 7.2, and the results of the Iterative Method (IM) and Local Search Method (LSM) relative to the algorithm ORTEC uses (OHD) can be seen in Section 7.3.

7.1 Parameters

The values of parameters that were also used in the algorithm by Kool et al. (2022) are not changed. This is to avoid overfitting and to focus on general execution instead of problem-tailored performance. For the IM, the parameter MAX_P, the maximum number of penalty increases that is used in Algorithm (2), is set to a value of 5. This is the maximum value while limiting the risk of integer overflows of the penalized objective value for the largest problems. For tuning, ten instances are selected from each group that was defined in Figure 7. A running time of 20 minutes is used since this is often the available running time in practice. The weights of Equation (2) are set to $\alpha = \frac{2}{1000}$ and $\beta = \frac{1}{60}$, which is a cost of 2 per kilometer and 60 per hour. Those values correspond to practical cases. The multi-objective is used for all runs in this section.

The first tuned parameter is MAX_TT, the maximum number of travel time increases that is used in Algorithm (2). The specific values used for tuning MAX_TT, as well as the average gap in the objective value to the best OHD solution over time for the three problem types, can be found in Figure 15 in Section 9.4. Although the results are relatively similar, $MAX\_TT = 40$ is used for the problems with congestion, and $MAX\_TT = 20$ is used for the other two problem types. In Figure 16 in Section 9.4, the results are split over the groups defined in Figure 7. It is concluded that it is reasonable to take the same value of MAX_TT independent of the group of an instance.

The parameter MAX_TT is also made dynamically adjustable, to fix the fraction of feasible solutions resulting from the LS to a specific value. This fraction is denoted by the parameter $FEAS\_LS$. The previously tuned values of MAX_TT are used as starting values. The chosen values used for tuning $FEAS\_LS$, and all the objective values over time, can be found in Figures (17) and (18) in Section 9.4. Although the results are relatively similar, $FEAS\_LS = 0.6$ is used for all problem types considered, independent of the group of an instance. The parameters MAX_TT and $FEAS\_LS$ are compared in Table 2. Although there is only a relatively small difference in performance, the dynamic parameter $FEAS\_LS$ is used in further experiments.

Table 2: Comparison of static and dynamic parameter. The average gap to the best OHD solution per problem type after 20 minutes of running time. A lower gap is better.

	Breaks	**Congestion**	**Breaks and congestion**
Using dynamic parameter $FEAS\_LS$	2.66	1.39	5.87
Using fixed parameter MAX_TT	2.51	1.18	5.83

Lastly, the parameter γ of Equation (11) for the LSM is tuned. This parameter is only used for problems including congestion. From the results of Figures (19) and (20) in Section 9.4, it is concluded that $\gamma = 1.0$ is used for the other experiments. This means that the algorithm is very optimistic. Namely, by only using the objective value calculated using the LBs on the travel times, moves are often evaluated with the real congestion to try to improve the route. Calculating the estimation with the UBs on the travel times is still necessary to obtain the start times that are used to evaluate the move with the real congestion.

7.2 Multi-Objective Function

The performance of the extended HGSADC when using the distance-only objective is compared to using the multi-objective function where route duration is added. For this, the average gap to the Best Known Solution (BKS) of OHD for the considered problem, is plotted over time. This measure will be referred to as *the average gap*. If the average gap is smaller than zero, the extended HGSADC finds on average better solutions than OHD. The average gap over time for the two different objective functions can be seen in Figure 10. This is for the problem type where both congestion and break scheduling are not taken into account, the so-called *basic problem*. The algorithm is run for 20 minutes since this is the running time that is often available to solve problems in practice.

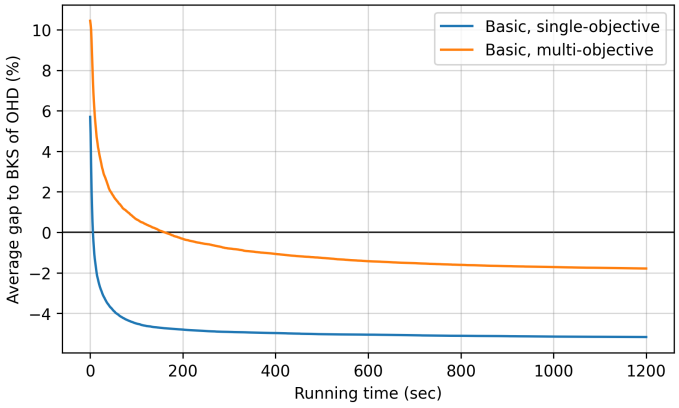
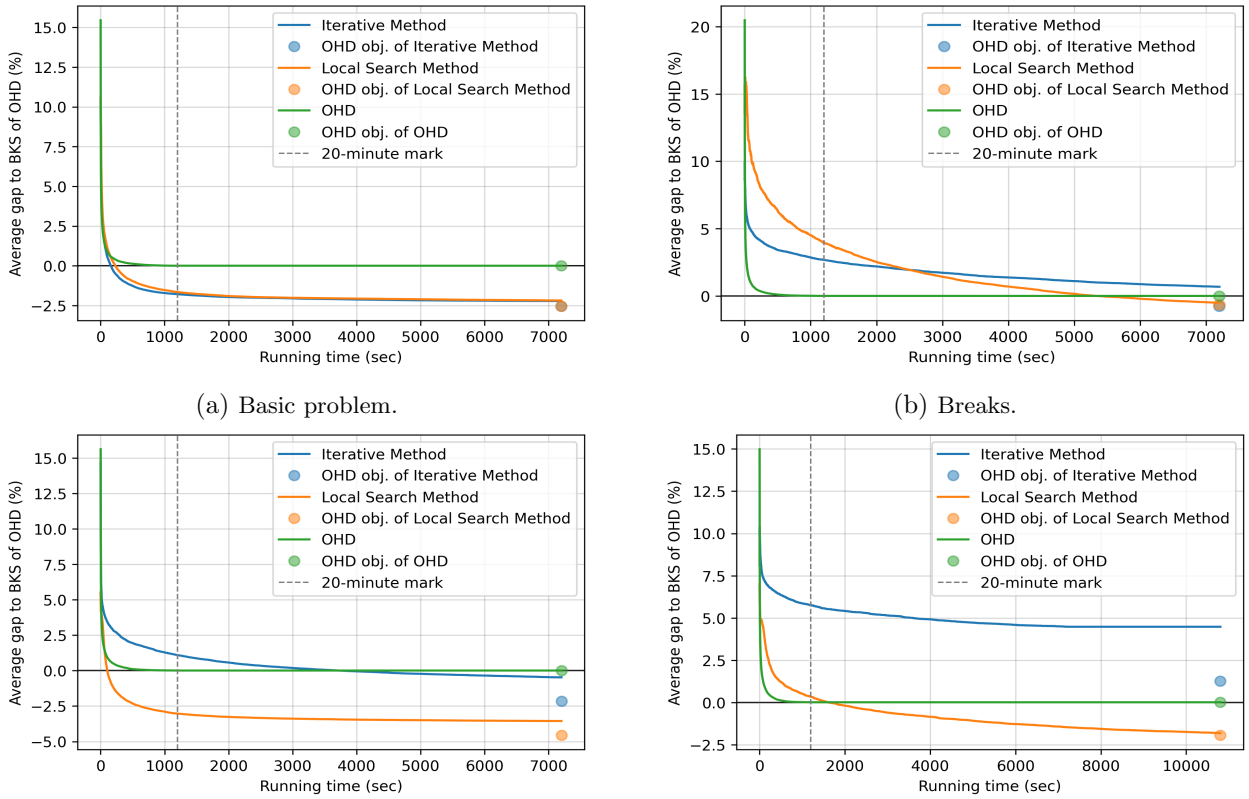


Figure 10: The average gap to the BKS of OHD over time for the basic problem, using the single- and multi-objective function.

It can be seen that the single-objective function performs approximately 3% to 5% better than the single-objective at all times. The single-objective is better than OHD after a few seconds, while the multi-objective outperforms OHD after approximately 150 seconds.

7.3 Constraints

The performance of the IM and the LSM is analyzed, using the same metric as in Section 7.2. Now, also the gap between the OHD objective value and its BKS over time is plotted to show its convergence to its BKS. The results can be seen in Figure 11, where the algorithms are run for 2 hours for the problems with up to one added constraint, and 3 hours when both break scheduling and congestion are included. The algorithms are given more time compared to Section 7.2, such that the potential of the HGSADC can also be seen when more time is available. A dashed, grey, vertical line indicates 12 minutes of running time. OHD is given the same amount of time as the HGSADC, but OHD stops on average after approximately 12 minutes since it is unable to improve its solution.



(a) Basic problem.

(b) Breaks.

(c) Congestion.

(d) Breaks and congestion.

Figure 11: The average gap to the BKS of OHD over time for the IM, LSM, and OHD. Results are grouped by the problem type considered. All problem types use the multi-objective function. The circles indicate the average solution value given by OHD for the best solutions of all individual methods. The dashed, grey line indicates the 20-minute mark.

The best solution of both the IM and the LSM, which is an ordered set of customers for each vehicle, is also given to OHD. OHD then checks if the solution is feasible and also gives the solution a value, without changing the solution. A solution's objective value given by OHD may differ from the objective value given by the HGSADC, due to different break scheduling, the absence of the

FIFO property in OHD, and other start time optimization. The OHD objective values of the best HGSADC solutions are indicated in Figure 11 by the colored circles. For the sake of completeness, the OHD objective value according to OHD is indicated with a green circle at an average gap of 0.0%. The results are combined per problem type to compare the relative performance of the two methods for each of the four problems type. The multi-objective is used for all problem types since the single-objective is rarely used in practical problems.

For the basic problem, the results can be seen in Figure 11a. Those results are equivalent, except that the LSM is on average 2.5 times slower than the IM. When only considering break scheduling, the IM on average still outperforms the LSM after 20 minutes, as can be seen in Figure 11b. However, the LSM is on average better than the IM after 2499 seconds and also outperforms OHD after 5338 seconds. Although the IM solution is on average 0.7% worse than the OHD solution after 2 hours, it is on average 0.8% better according to the OHD solver itself. This is in fact on average 0.2% better than the LSM method, according to the OHD solver. In Figure 11c, the results can be found for problems with congestion. The LSM significantly outperforms OHD at all times. The IM is on average worse at the 20-minute mark. However, it is very competitive with OHD when using the OHD objective values of its solution. After 3712 seconds, it surpasses OHD and the IM has on average 2.1% better solution values after two hours when considering the OHD objective values of the solutions. Finally, the results for the most advanced problems, which consider breaks and congestion, can be seen in Figure 11d. The LSM is on average better than OHD after 1592 seconds. The IM on average does not find solutions of comparable quality, although the OHD objective value of its solutions is competitive with the average OHD solutions after three hours.

The results of Figure 11, but clustered per method and using a logarithmic x-axis, can be found in Figure 21 in Section 9.5.

In Table 3, more detailed data can be found about the results of Figures (10) and (11). For each problem type considered, every used HGSADC method is given a row with information on the solution values. Namely, the number of best solutions over the different methods, as well as the average gap to the BKS of OHD are given for three possible scenarios; after 20 minutes, at the end of the available running time, and at the end of the available running time when using the OHD objective of the HGSADC solutions.

Table 3: Detailed data on the solutions of OHD and the used HGSADC methods, clustered per problem type. For each combination of problem type and used solver, the results are given after running for 20 minutes, for the total available running time, and for the total available running time when assessing the HGSADC solutions with the OHD solver. The results consist of the percentage of best solutions found over the different methods used, as well as the average gap to the BKS of OHD. Lower values are better.

		20 min		**End of run**		**Using OHD obj.**	
		Best	**Gap**	**Best**	**Gap**	**Best**	**Gap**
Standard single-obj.	IM	100.0 %	-5.2 %	100.0 %	-5.3 %	100.0	-5.3 %
	OHD	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %
Standard multi-obj.	IM	56.9 %	-1.8 %	52.1 %	-2.2 %	45.7 %	-2.6 %
	LSM	39.4 %	-1.7 %	47.4 %	-2.2 %	54.3 %	-2.5 %
	OHD	3.7 %	0.0 %	0.5 %	0.0 %	0.0 %	0.0 %
Breaks	IM	3.2 %	2.7 %	10.6 %	0.7 %	40.4 %	-0.8 %
	LSM	19.1 %	3.9 %	64.4 %	-0.5 %	41.0 %	-0.6 %
	OHD	77.7 %	0.0 %	25.0 %	0.0 %	18.6 %	0.0 %
Congestion	IM	0.0 %	1.1 %	0.0 %	-0.5 %	1.1 %	-2.1 %
	LSM	100.0 %	-3.0 %	100.0 %	-3.6 %	98.9 %	-4.6 %
	OHD	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %
Breaks and congestion	IM	0.0 %	5.7 %	0.5 %	4.5 %	12.3 %	1.3 %
	LSM	37.4 %	0.3 %	84.0 %	-1.5 %	77.0 %	-1.6 %
	OHD	62.6 %	0.0 %	15.5 %	0.0 %	10.7 %	0.0 %

From this table, it can for example be seen that the LSM finds the highest number of best solutions after the available running time when assessing the solutions with the OHD solver.

There are a few instances that could not be solved with an HGSADC method after 20 minutes of running time. This concerns the two largest instances for the problem type with breaks, and one smaller instance for the problem type with breaks and congestion. However, solutions were found for all instances at the end of the available running time. OHD was able to find solutions for all instances for all problem types within 20 minutes.

8 Conclusion

8.1 Main Findings

The potential of the framework of the HGSADC by Vidal et al. (2013) to solve practical CVRPTWs is shown in this research. Two methods are proposed to extend the framework of the HGSADC. Firstly, the IM is proposed where practical constraints are not directly incorporated into the LS. The LS only deals with a relaxation of the full problem. The instance data used in the LS is iteratively updated in such a way that the solutions of the LS will converge to a feasible solution for the full problem. This is a general framework that could be applied to most practical constraints. Secondly, the LSM is introduced, where the practical constraints are actively taken into account in the LS in a sophisticated way. Adding constraints to the HGSADC using the LSM will often require more specialized techniques, as this is not a general framework, but rather a specialized technique. In practice, it would likely be increasingly difficult to add more practical constraints to the HGSADC using the LSM, as each individual constraint requires its own specialized way of incorporation.

First, the distance-only objective is changed to a multi-objective function by incorporating the route duration. Then, two constraints, break scheduling and congestion, are implemented using both the IM and the LSM. Those two constraints are selected because they are the most commonly used constraints in practical problems, and they are expected to be the most challenging constraints to incorporate in the HGSADC. A selection of 192 real-world instances is used to show the potential of both methods to solve practical problems. A comparison is made with the algorithm that ORTEC currently uses (OHD).

The results show that the HGSADC still finds approximately 2% better solutions compared to OHD when the multi-objective function is implemented. Also, the IM is very competitive to OHD after 20 minutes of running time when adding a single practical constraint. After two hours, the IM improves the average objective value of OHD by approximately 1% to 2%. However, the results of the IM tend to fall short when multiple practical constraints are implemented using this method. The results of the LSM after 20 minutes heavily depend on the constraint(s) considered. However, the LSM outperforms both the IM and OHD after two hours of running time for all problem types considered.

8.2 Discussion and Further Research

As can be seen in Figure 10, the relative performance of the HGSADC compared to OHD is significantly better when using the single-objective to solve the basic problem. There are two reasons for this significant difference. Firstly, the HGSADC was created to solve the single-objective problem originally, using for example time-saving pre-checks that are removed when using the multi-objective function. This removes one of the competitive edges of the HGSADC. Secondly, OHD is not created to solve single-objective problems, since practical problems always involve optimizing multiple objectives.

When looking at the results of Figure 11a, the IM and LSM perform exactly the same, except

that the LSM is on average 2.5 times slower than the IM. This is because the LSM already uses the hierarchical concatenation of subroutes, where inefficiencies in the structure of the C++ code slow down the algorithm. In further research, this factor 2.5 could likely be reduced significantly using a more efficient structure of the code. Taking this into account when considering breaks in Figure 11b, the IM and LSM have a similar performance after 20 minutes, as both methods perform on average about 2% worse than OHD. The fact that the LSM does not perform significantly better than the IM, while this is the case for the other problem types, is possibly due to scheduling the breaks using a very basic heuristic in the LSM. Dynamical Programming could be used in further research to optimally schedule the breaks in the LSM, which would likely improve the solution significantly while using the same running time. When incorporating breaks and congestion, or congestion only, the LSM performs on average approximately 4% to 6% better than the IM. This strongly indicates that the procedure of updating the instance data leads to highly suboptimal solutions due to service and travel times being increased too much. This could be improved in further research by decreasing the travel- and service times to their real value after a certain number of iterations. This would likely result in improving the solutions, although it could not be guaranteed that the same improvements are found repeatedly after each decrease in travel- and service times.

To conclude, this research shows the great potential of the HGSADC to be used to solve practical CVRPTWs. Firstly, the IM method, which could be used for most practical constraints, shows to be highly competitive to OHD. A disadvantage of this method is that incorporating more practical constraints seems to reduce the competitiveness of the HGSADC. Secondly, the LSM shows great results after two hours of running time by finding significantly better solutions than OHD for all tested combinations of practical constraints. However, this method is more specialized, and its results after 20 minutes depend on the practical constraints considered.

It should be noted that OHD is an algorithm that is capable of solving practical CVRPTWs with many other constraints. Therefore, the results of this research do not show the incapabilities of OHD, but only use it to show the potential of the HGSADC.

This study tested the two proposed methods with two constraints. Further research could be focused on adding more/other constraints using the two methods. Also, the results could be compared to other solvers, to obtain a better idea of the quality of the solution of the two methods. The methods themselves could be further improved as well. Firstly, there are multiple possibilities on how to use the bounds in the LSM to decide if a move is evaluated with the real congested travel time. It is also possible to make an estimation on the travel time, based on a linear interpolation (as can be found in Section 9.3) and add this to the information of the two bounds to make a better estimation of whether to evaluate of move. Secondly, the decision to evaluate a move or not could be directly based on the LB and UB. For example, the move is evaluated when the UB before the move is better/lower than the LB after the move. But also when no better solution is guaranteed by the bounds the move could be decided to be evaluated, so for example when the LB after the move is better than before the move.

This study focused on showing the potential to solve practical CVRPTWs with the HGSADC,

by targeting the largest obstacle; dealing with the constraints in the LS. However, more improvements can be made by developing specialized construction heuristics, LS neighborhoods, and SPLIT algorithm etc. Currently, the mentioned aspects of the HGSADC are only based on travel distance.

8.3 Advice for ORTEC

As the results of this research show that the HGSADC has great potential to solve practical CVRPTWs, the next step is to investigate if it is possible to add all other practical constraints to this algorithm, using either the IM or LSM. It would be recommended to incorporate at least a portion of the practical constraints using the LSM, as the quality of the results of the IM tends to decrease when adding more constraints using the IM. When this is possible, the HGSADC is able to find solutions of excellent quality compared to OHD, especially for cases where a running time of several hours is available. When less time is available, this algorithm can for example be used as an extension of OHD. Namely, OHD is often able to find solutions of good-quality in a relatively short amount of time. After a few minutes, the HGSADC could be used to improve the solution of OHD, by using an initial population largely inspired by the solution of OHD.

References

Accorsi, L., & Vigo, D. (2021). A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems. *Transportation Science*, *55*(4), 832–856.

Arnold, F., & Sörensen, K. (2019). What makes a VRP solution good? The generation of problem-specific knowledge for heuristics. *Computers & Operations Research*, *106*, 280–288.

Balseiro, S., Loiseau, I., & Ramonet, J. (2011). An ant colony algorithm hybridized with insertion heuristics for the time dependent vehicle routing problem with time windows. *Computers & Operations Research*, *38*(6), 954–966.

Christiaens, J., & Vanden Berghe, G. (2020). Slack induction by string removals for vehicle routing problems. *Transportation Science*, *54*(2), 417–433.

Conrad, R. G., & Figliozzi, M. A. (2011). The recharging vehicle routing problem. *Proceedings of the 2011 industrial engineering research conference*, *8*.

Current, J., & Marsh, M. (1993). Multiobjective transportation network design and routing problems: Taxonomy and annotation. *European Journal of Operational Research*, *65*(1), 4–19.

Dabia, S., Ropke, S., van Woensel, T., & Kok, T. D. (2013). Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation Science*, *47*(3), 380–396.

Desrochers, M., Desrosiers, J., & Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, *40*, 342–354.

Ding, Q., Hu, X., Sun, L., & Wang, Y. (2012). An improved ant colony optimization and its application to vehicle routing problem with time windows. *Neurocomputing*, *98*, 101–107.

Eksioglu, B., Vural, A. V., & Reisman, A. (2009). The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, *57*(4), 1472–1483.

European Union. (2002). Directive 2002/15/EC of the European Parliament and of the Council of 11 March 2002 on the organisation of the working time of persons performing mobile road transport activities. Official Journal of the European Communities L 080, 23.03.2002.

European Union. (2006). Regulation (EC) No 561/2006 of the European Parliament and of the Council of 15 March 2006 on the harmonisation of certain social legislation relating to road transport and amending Council Regulations (EEC) No 3821/85 and (EC) No 2135/98 and repealing Council Regulation (EEC) No 3820/85. Official Journal of the European Communities L 102/1, 11.04.2006.

Goel, A. (2009). Vehicle scheduling and routing with drivers' working hours. *Transportation Science*, *43*(1), 17–26.

Goel, A., & Kok, A. (2009). Efficient scheduling of team truck drivers in the European Union. *Flexible Services and Manufacturing Journal*, *24*, 81–96.

Haghani, A., & Jung, S. (2005). A dynamic vehicle routing problem with time-dependent travel times. *Computers & Operations Research*, *32*(11), 2959–2986.

Hashimoto, H., Yagiura, M., & Ibaraki, T. (2008). An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *Discrete Optimization*, *5*(2), 434–456.

Helsgaun, K. (2017). An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 24–50.

Ichoua, S., Gendreau, M., & Potvin, J.-Y. (2003). Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, *144*(2), 379–396.

Kallehauge, B. (2008). Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers & Operations Research*, *35*(7), 2307–2330.

Kok, A. L., Meyer, C. M., Kopfer, H., & Schutten, J. M. J. (2010a). A dynamic programming heuristic for the vehicle routing problem with time windows and European Community Social Legislation. *Transportation Science*, *44*(4), 442–454.

Kok, A., Hans, E., & Schutten, J. (2011). Optimizing departure times in vehicle routes. *European journal of operational research*, *2011*(201), 579–587.

Kok, A., Hans, E., Schutten, J., & Zijm, W. (2010b). A dynamic programming heuristic for vehicle routing with time-dependent travel times and required breaks. *Flexible services and manufacturing journal*, *2010*(22), 83–108.

Kolen, A. W. J., Kan, A. H. G. R., & Trienekens, H. W. J. M. (1987). Vehicle routing with time windows. *Operations Research*, *35*(2), 266–273.

Kool, W., Juninck, J. O., Roos, E., Cornelissen, K., Agterberg, P., van Hoorn, J., & Visser, T. (2022). Hybrid genetic search for the vehicle routing problem with time windows: A high-performance implementation. Retrieved April 14, 2022, from https://wouterkool.github.io/pdf/paper-kool-hgs-vrptw.pdf

Lahyani, R., Khemakhem, M., & Semet, F. (2015). Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research*, *241*(1), 1–14.

Malandraki, C., & Daskin, M. (1992). Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation Science*, *26*, 185–200.

Marinakis, Y., Marinaki, M., & Migdalas, A. (2019). A multi-adaptive particle swarm optimization for the vehicle routing problem with time windows. *Information Sciences*, *481*, 311–329.

Moccia, L., Cordeau, J.-F., & Laporte, G. (2012). An incremental tabu search heuristic for the generalized vehicle routing problem with time windows. *Journal of the Operational Research Society*, *63*.

Moghdani, R., Salimifard, K., Demir, E., & Benyettou, A. (2021). The green vehicle routing problem: A systematic literature review. *Journal of Cleaner Production*, *279*, 123691.

Mortada, S., & Yusof, Y. (2021). A neighbourhood search for artificial bee colony in vehicle routing problem with time windows. *Int J Intell Eng Syst*, *14*(3), 255–266.

Nagata, Y., Bräysy, O., & Dullaert, W. (2010). A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, *37*(4), 724–737.

Nagata, Y., & Kobayashi, S. (2010). A memetic algorithm for the pickup and delivery problem with time windows using selective route exchange crossover. *6238*, 536–545.

Oliver, I. M., Smith, D. J., & Holland, J. R. C. (1987). Study of permutation crossover operators on the traveling salesman problem. *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, 224–230.

Ombuki, B., Ross, B., & Hanshar, F. (2006). Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, *24*, 17–30.

Prescott-Gagnon, E., Desaulniers, G., Drexl, M., & Rousseau, L.-M. (2010). European driver rules in vehicle routing with time windows. *Transportation Science*, *44*, 455–473.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, *31*(12), 1985–2002.

Rincon-Garcia, N., Waterson, B., Cherrett, T. J., & Salazar-Arrieta, F. (2020). A metaheuristic for the time-dependent vehicle routing problem considering driving hours regulations – an application in city logistics. *Transportation Research Part A: Policy and Practice*, *137*, 429–446.

Setak, M., Habibi, M., Karimi, H., & Abedzadeh, M. (2015). A time-dependent vehicle routing problem in multigraph with FIFO property. *Journal of Manufacturing Systems*, *35*, 37–45.

Subramanian, A., Uchoa, E., & Ochi, L. S. (2013). A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, *40*(10), 2519–2531.

Vidal, T. (2015). Technical note: Split algorithm in O(n) for the vehicle routing problem. *Computers & Operations Research*, *69*.

Vidal, T. (2022). Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, *140*, 105643.

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, *60*, 611–624.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, *40*(1), 475–489.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2015). Timing problems and algorithms: Time decisions for sequences of activities. *Networks*, *65*.

Visser, T., & Spliet, R. (2020). Efficient move evaluations for time-dependent vehicle routing problems. *Transportation Science*, *54*.

9 Appendix

9.1 Data Transformation

The structure of the data that is used in this research does not guarantee the FIFO property, which is explained in Section 2.3.2. This property is violated when discontinuities in the travel time function allow for an earlier arrival at a later departure. Those discontinuities will be called downward discontinuities in this thesis since the travel time has a lower value after the discontinuity.

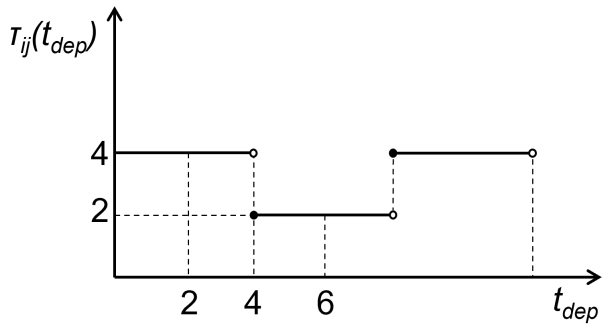
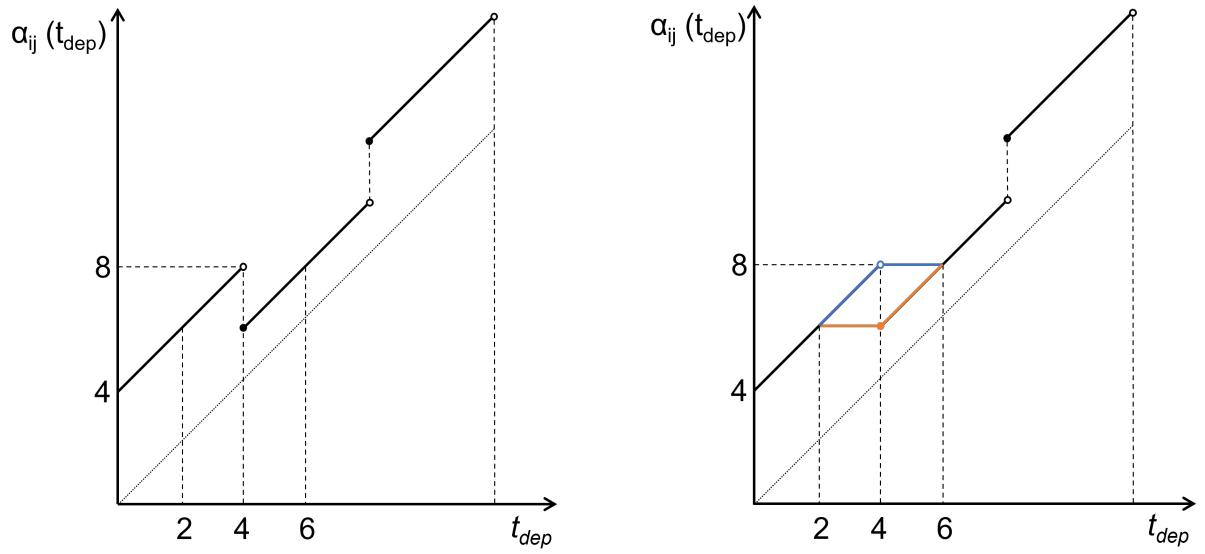


Figure 12: Travel time as a function of departure time t_{dep}. An example of a violation of the FIFO property caused by the downward discontinuities at departure time $t_{dep} = 4$.

An example of a downward discontinuity can be seen in Figure 12. The FIFO property is violated, since a departure between $t_{dep} = 2$ and $t_{dep} = 4$ will result in a later arrival compared to waiting with the departure until $t_{dep} = 4$. The same conclusion can be drawn using the corresponding arrival time function that can be seen in Figure 13a.

Different transformations of the data are possible to guarantee the FIFO property. Two options will be considered, illustrated in Figure 13b. The first option is to use the minimum arrival time. In this case, indirectly, it is allowed to wait at a customer and depart in a future time interval if that results in an earlier arrival time. The resulting arrival time function is given by the orange and black lines in Figure 13b. The second option is to increase the travel times directly after downward discontinuities in the travel time function. By increasing the travel time by the amount of waiting time, waiting will never result in earlier arrivals. The resulting arrival time function is given by the blue and black lines in Figure 13b. For both methods, the horizontal lines may be changed slightly (to a slightly increasing line) to guarantee the inverse of the arrival time function exists, and that the FIFO property holds. Other transformations are possible as well, as long as the resulting arrival time function is increasing.



(a) FIFO violation example due to the downward discontinuities at departure time $t_{dep} = 4$.

(b) Two possible data transformations to remove the FIFO violation.

Figure 13: Arrival time as a function of departure time t_{dep}. An example of a violation of the FIFO property and a possible data transformation to remove that violation.

The first option, corresponding to the orange and black lines in Figure 13b, is mathematically the best way to optimize the vehicle routes with respect to total duration. However, this might result in routes that are impossible to drive in the real world, since the travel times are underestimated in the model at the start of the orange line. Also, using this option might result in a difficult comparison between the results of this thesis and the results of OHD. Namely, OHD ignores the FIFO property such that it could make use of downward discontinuities in the travel time function. Although OHD is not tuned to find such opportunities, it does make use of it when encountered. Therefore, the second option, corresponding to the blue and black lines in Figure 13b, is used in this thesis to guarantee that a lower objective value of the HGSADC compared to OHD is not caused by the data transformation used. As a result, route duration could be slightly overestimated compared to OHD.

9.2 Pseudocode

The pseudocode of the IM can be seen in Algorithm 2. In this algorithm, "update T_S_TIMES;" depends on the problem type considered. When congestion is considered, this updates the travel times. If breaks are (also) considered, this updates travel- and service times (as well).

Algorithm 2: iterativeMethod(PENS, INDIV, T_S_TIMES, MAX_P, MAX_TT)

Result: (possibly improved) INDIV is added to the correct pool of solutions)

Input:

PENS, values to penalize infeasibilities (time warps and capacity-excess);

INDIV, new individual created using crossover operators;

T_S_TIMES, constant travel- and service times without congestion and breaks;

MAX_P, maximum number of increases of PENS;

MAX_TT, maximum number of increases of T_S_TIMES;

Initialization:

PENS_INCREASES ← 0; T_S_TIMES_INCREASES ← 0;

LOOP ← **true**;

BEST_FEAS_SOL ← `null`;

Iterative Process:

INDIV ← result after LS on INDIV using T_S_TIMES and PENS;

while *LOOP* **do**

  if *INDIV basic feasible* **then**

    if *INDIV congestion feasible* **then**

      add INDIV to pool of feasible solutions and update BEST_FEAS_SOL;

      LOOP ← **false**;

    else if *T_S_TIMES_INCREASES = MAX_TT* **then**

      put INDIV in pool of infeasible solutions;

      LOOP ← **false**;

    else

      update T_S_TIMES;

      T_S_TIMES_INCREASES ← T_S_TIMES_INCREASES + 1;

      INDIV ← result after LS on INDIV using T_S_TIMES and PENS;

    end

  else

    if *PENS_INCREASES = MAX_P* **then**

      put INDIV in pool of infeasible solutions;

      LOOP ← **false**;

    else

      PENS ← PENS · 10;

      PENS_INCREASES ← PENS_INCREASES + 1;

      INDIV ← result after LS on INDIV using T_S_TIMES and PENS;

    end

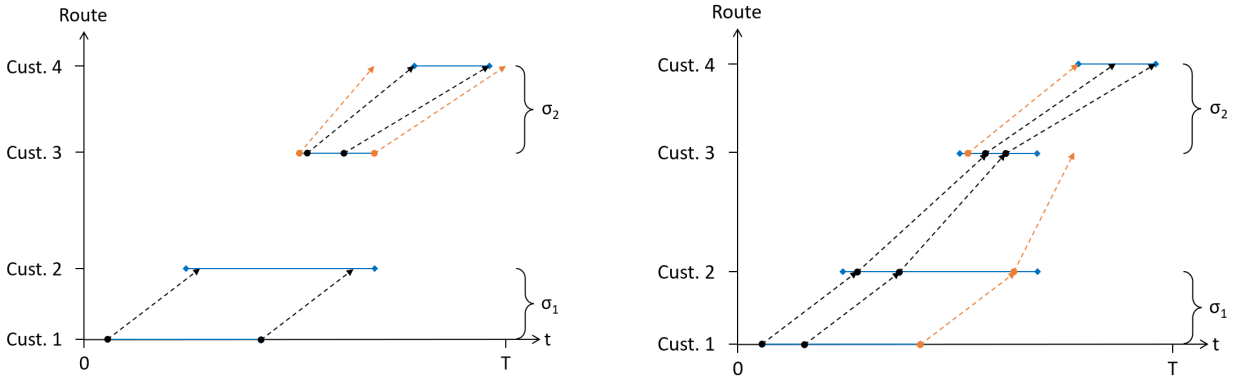
  end

end

reset PENS and T_S_TIMES;

9.3 Congestion with the LSM: Linear Interpolation

Another idea for the LSM to incorporate congestion uses linear interpolation of the congestion factors. This technique will be explained using the example that can be seen in Figure 14. In Figure 14a, customers 1 and 2 are concatenated equivalently to the implemented LSM. For customer 3, the so-called *potential* earliest and latest travels to customer 4, indicated by the orange arrows, arrive too early or too late at customer 4. A constant time backwards calculation procedure, using linear interpolation of the congestion factor, can be used to calculate the earliest and latest departure at customer 3 such that the vehicle arrives in the time window of customer 4. Those so-called *possible* travels are indicated by the black arrows. The congestion of the black arrows is linearly interpolated between the two congestion factors of the orange arrows. The same procedure is used in Figure 14b to concatenate subroutes σ_1 and σ_2. The result is a smaller "trail" through space-time of possible routes with a minimum waiting time and time warp. This linear interpolation of congestion factors will likely be an oversimplification of the problem for long routes, but it will probably give a good indication of the duration for short (sub)routes.



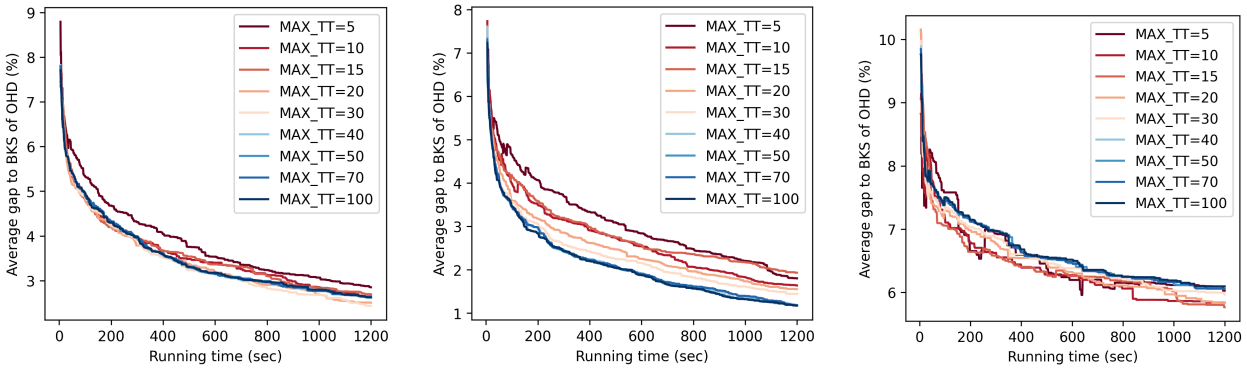
(a) Customers 1 and 2 are concatenated as subroute σ_1, customers 3 and 4 are concatenated as subroute σ_2.

(b) Subroutes σ_1 and σ_2 are concatenated using linear interpolations of the travel time.

Figure 14: One example where customers and subroutes are concatenated in two steps using the linear interpolation method. The blue, horizontal lines represent the time windows of the customers. The orange arrows show the *potential* travels from a first to a second customer when starting at the earliest or latest departure time at the first customer. The black arrows show the *possible* travels from a first to a second customer when starting at the first or latest departure time at the first customer, such that the waiting time and time warp is minimized.

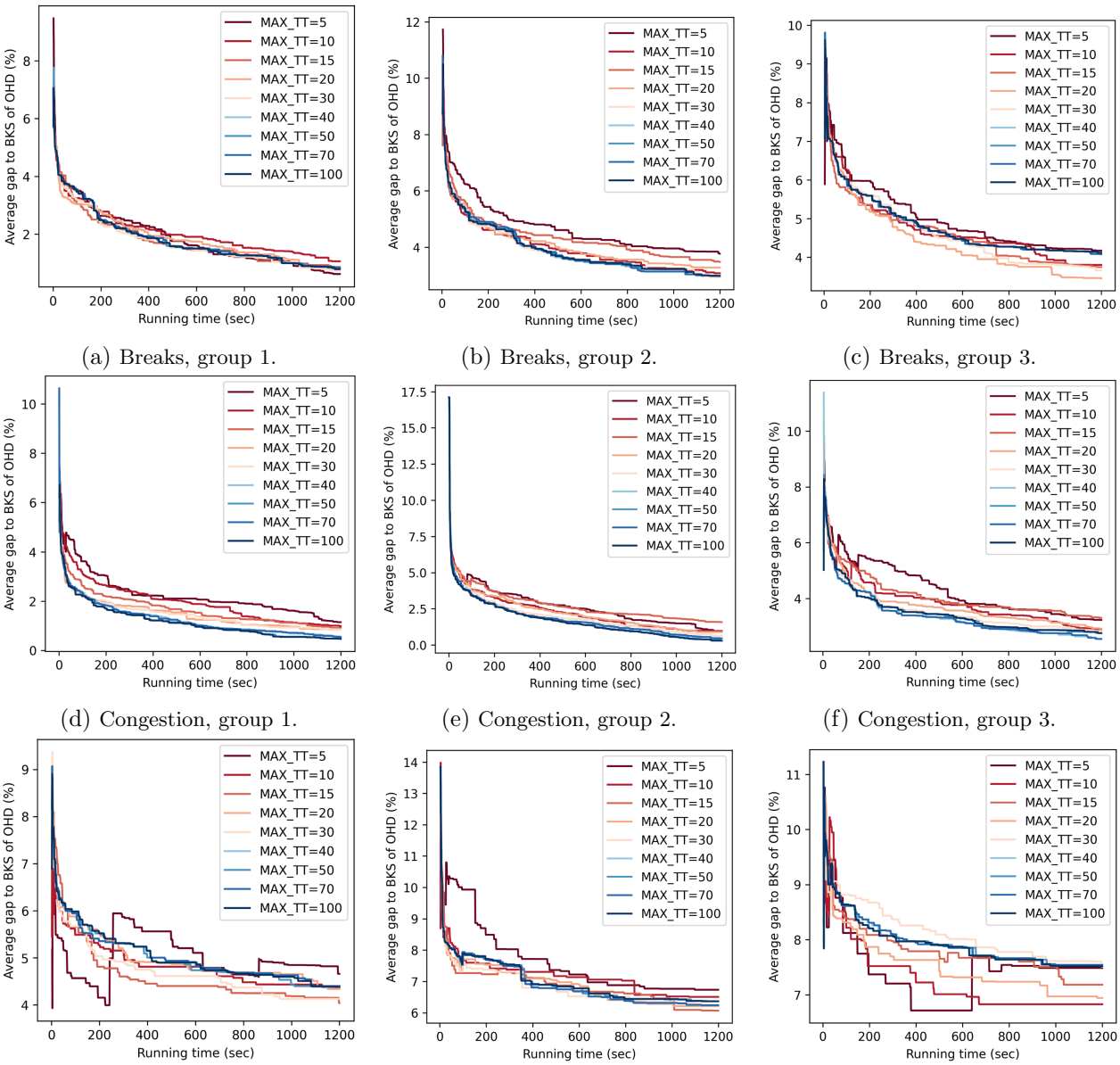
9.4 Parameter Tuning

The results of tuning the parameter MAX_TT for the problems with breaks, congestion, and both constraints, can be found in Figure 15. Equivalent results, but split over the three groups of Figure 7 can be seen in Figure 16. The same data can be seen in Figures 17 and 18 for the tuning of the parameter $FEAS\_LS$, and in Figures 19 and 20 for tuning of the parameter γ.



(a) Including breaks. (b) Including congestion. (c) Including breaks and congestion.

Figure 15: Tuning results of MAX_TT. Average gap to best OHD solution over time, for three different problem types.

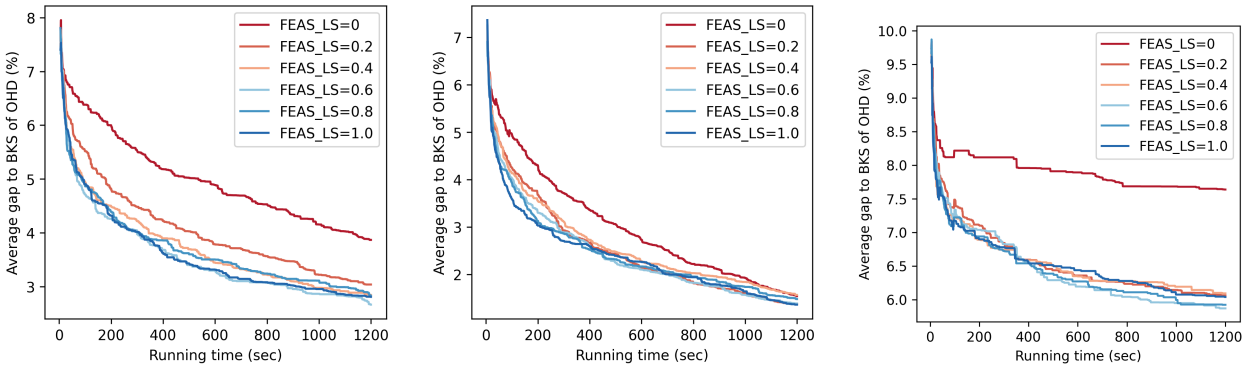


(a) Breaks, group 1. (b) Breaks, group 2. (c) Breaks, group 3.

(d) Congestion, group 1. (e) Congestion, group 2. (f) Congestion, group 3.

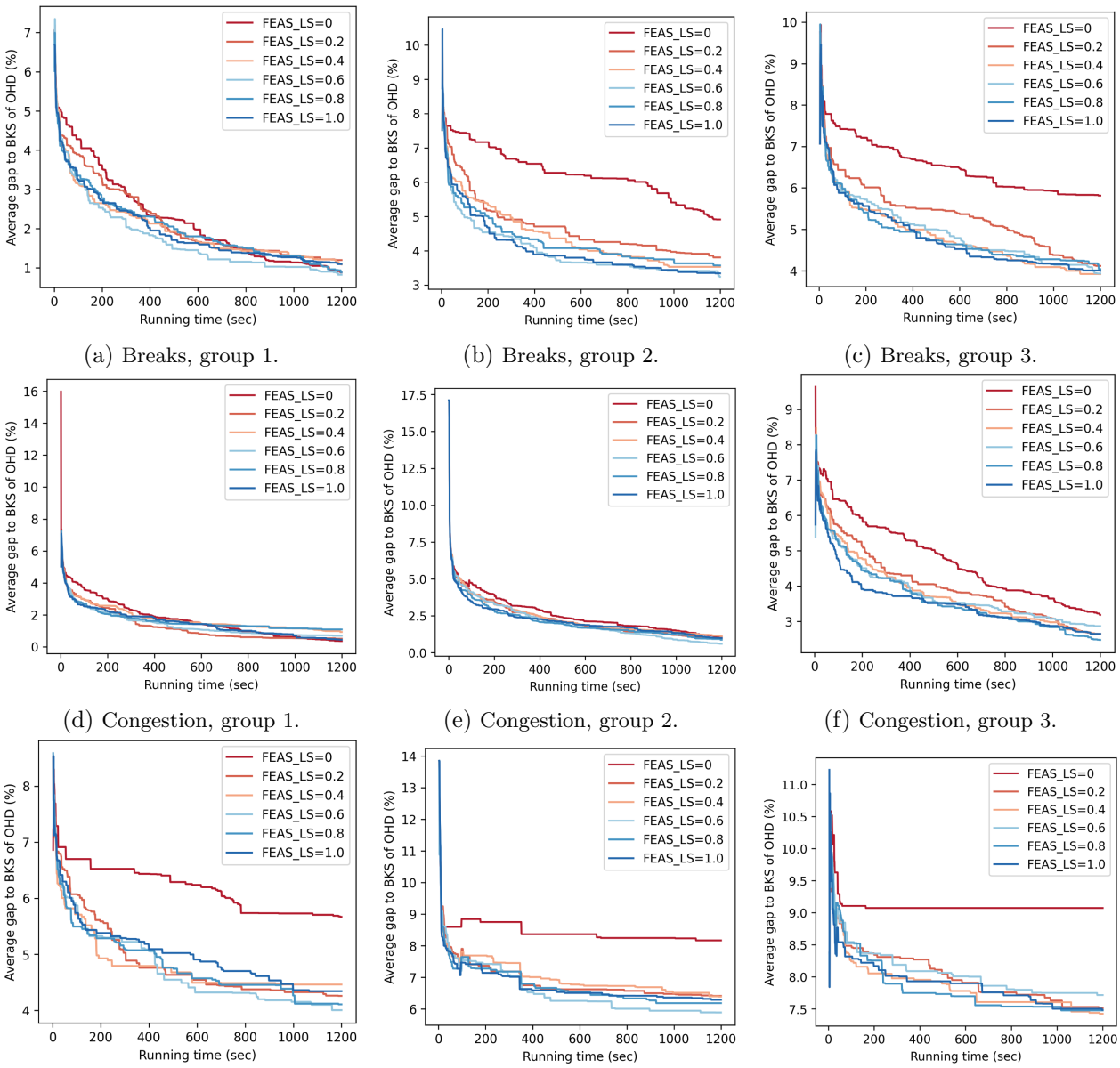
(g) Breaks and congestion, group 1. (h) Breaks and congestion, group 2. (i) Breaks and congestion, group 3.

Figure 16: Tuning results of MAX_TT. Average gap to best OHD solution over time, for three different problem types, split over the three groups.



(a) Including breaks. (b) Including congestion. (c) Including breaks and congestion.

Figure 17: Tuning results of *FEAS_LS*. Average gap to best OHD solution over time, for three different problem types.

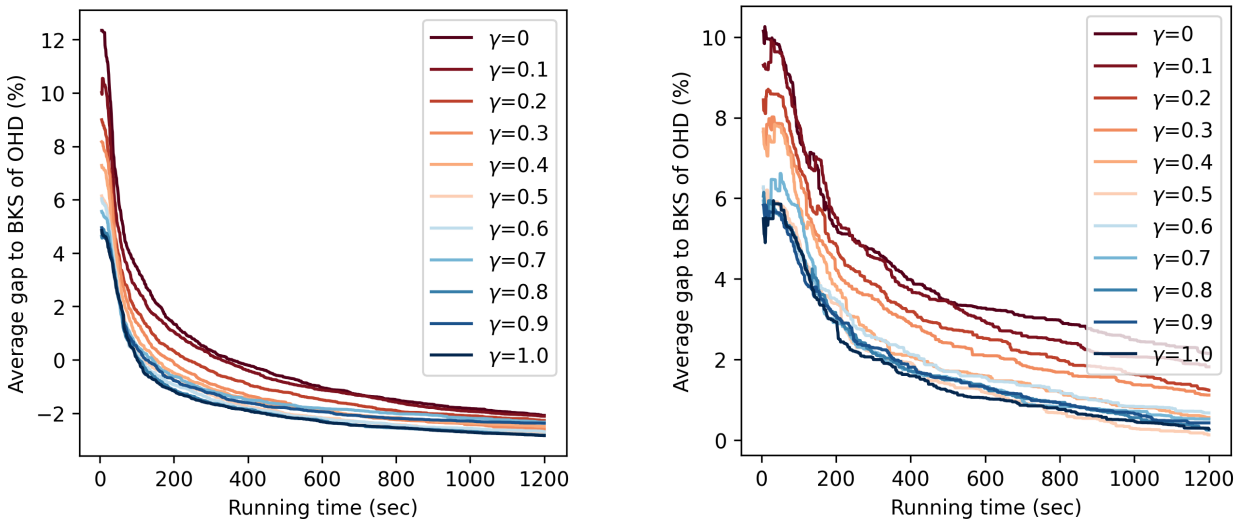


(a) Breaks, group 1. (b) Breaks, group 2. (c) Breaks, group 3.

(d) Congestion, group 1. (e) Congestion, group 2. (f) Congestion, group 3.

(g) Breaks and congestion, group 1. (h) Breaks and congestion, group 2. (i) Breaks and congestion, group 3.

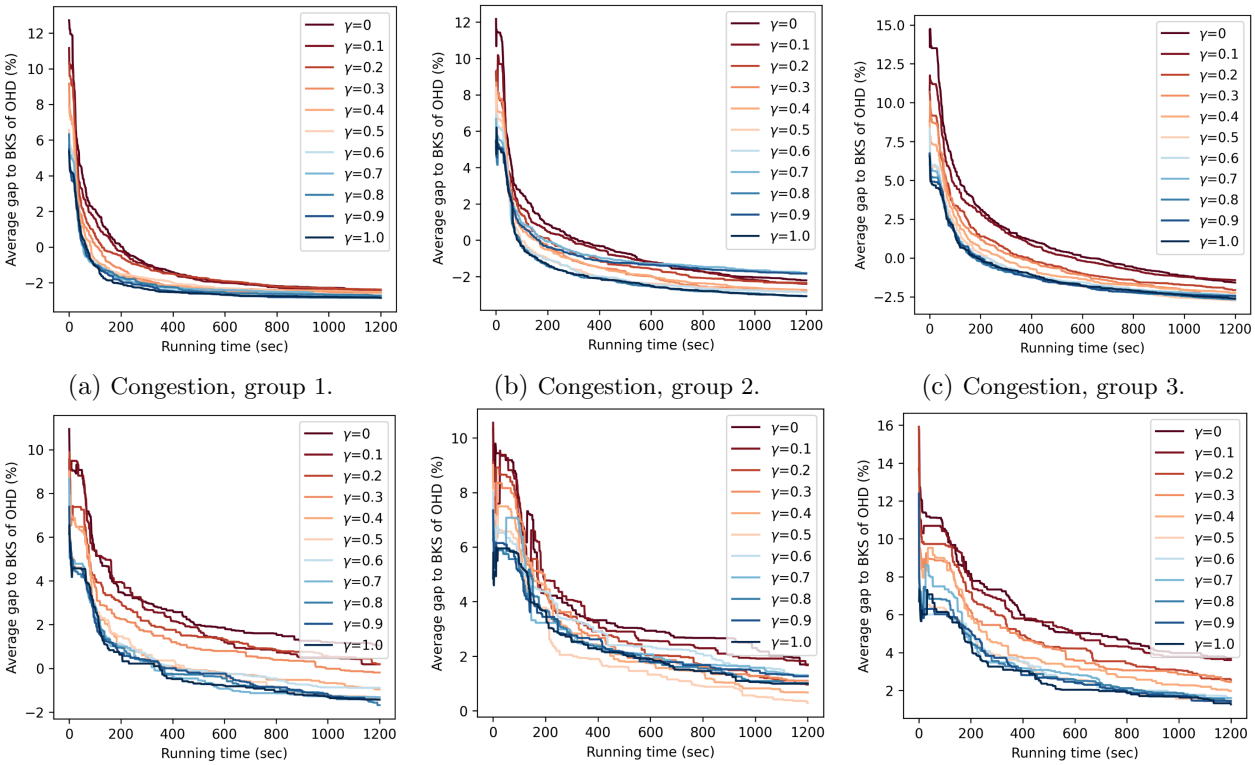
Figure 18: Tuning results of $FEAS\_LS$. Average gap to best OHD solution over time, for three different problem types, split over the three groups.



(a) Including congestion.

(b) Including breaks and congestion.

Figure 19: Tuning results of γ. Average gap to best OHD solution over time, for three different problem types.



(a) Congestion, group 1.

(b) Congestion, group 2.

(c) Congestion, group 3.

(d) Breaks and congestion, group 1.

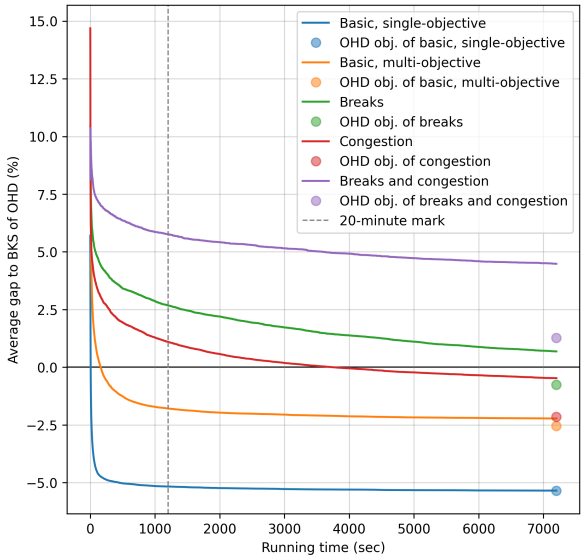
(e) Breaks and congestion, group 2.

(f) Breaks and congestion, group 3.

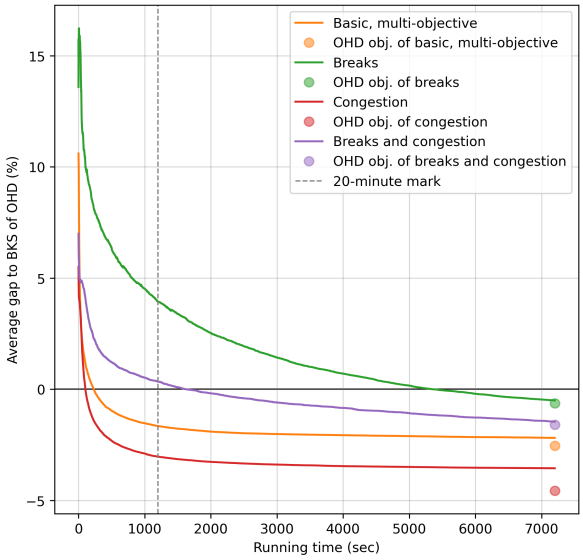
Figure 20: Tuning results of γ. Average gap to best OHD solution over time, for two different problem types, split over the three groups.

9.5 Results

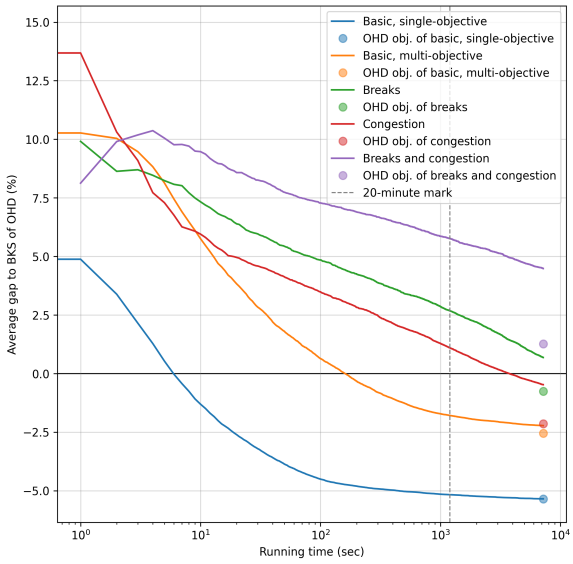
The gap to the BKS of OHD over running time can be seen in Figures 21a and 21b for the IM and LSM respectively. Those are the same results as in Figure 11 but now clustered per method. Figures 21c and 21d show equivalent results but now using a logarithmic horizontal axis. The linear trends in Figures 21c and 21d show that, after two hours of running time, the objective values are logarithmically improved over time for some problem types.



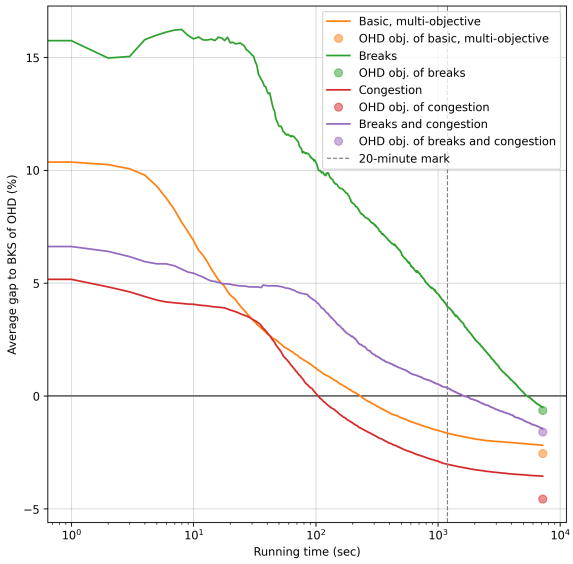
(a) IM, linear horizontal axis.



(b) LSM, linear horizontal axis.



(c) IM, logarithmic horizontal axis.



(d) LSM, logarithmic horizontal axis.

Figure 21: The average gap to the BKS of OHD over time for the IM and LSM. The circles indicate the average solution value given by OHD for the best solutions of all individual methods. The dashed, grey line indicates the 20-minute mark.