Erasmus University Rotterdam
Erasmus School of Economics

Master Thesis Econometrics and Management Science

The Team Orienteering Problem with Service-Time-Dependent Profits and Time Windows

Abstract

The Team Orienteering Problem consists of selecting and visiting a subset of nodes such that the obtained profit is maximised and a time budget is not exceeded. The Team Orienteering Problem with Service-Time-Dependent Profits and Time Windows is a new generalisation of the Team Orienteering Problem, where the obtained profit depends on the duration of the visit at each of the nodes and each node has a time window during which profit can be collected. A Hybrid Adaptive Large Neighbourhood Search method is proposed, decomposing the problem into route generation and service time scheduling components. The Service-Time Scheduling Problem is solved using a general purpose solver and a proposed Modified Coordinate Search method. The proposed algorithm where the Service-Time Scheduling Problem is solved exactly shows to be a reliable method for the researched problem, showing steady computational times and strong solutions, especially for larger instances. When the Modified Coordinate Search method is used for the Service-Time Scheduling Problem, both solving times and solution quality significantly decrease compared to the exact method.

Name student: V.T. Schouten

Student ID number: 449249

Supervisor: M.H. Akyuz

Second assessor: Prof. Dr. Rommert Dekker

Date final version: December 5, 2021

Contents

1 Introduction

Tsiligirides (1984) describes the origin of the Orienteering Problem (OP) coming from the sport of orienteering, more specifically in the form of a score orienteering event. This event has its participants navigate through a forest using a map and compass, arriving at an end point before a given time and obtaining points by visiting control points of different worth. The aim of the participants is to determine which control points are feasible to visit within the given time limit, while maximising their obtained total score in order to win the event.

The Team OP (TOP) was first examined by Butt and Cavalier (1994), although defined as the Multiple Tour Maximum Collection Problem. This problem generalises the OP by requiring multiple distinct tours to be determined. The inspiration for examining the MTMCP was the recruitment program for the football team of Earlham College in Richmond, U.S.A. A proven approach for recruitment was to meet with senior members of the teams from other schools. Earlham college wanted to visit as many schools as possible within a 100-mile radius while maximising its recruitment potential. The recruiter was not allowed to stay the night, and thus had to start and end the tour of the day at the same location, i.e. the depot. Students were only available for contact during school hours, resulting in an upper bound on the amount of visiting hours on a day. Lastly, the recruiter only had a limited number of days for visitation, which in the context of the MTMCP means that multiple tours have to be determined.

In the context of OP with Service-Time-Dependent Profits (OPSTP), Erdoğan and Laporte (2013) discuss its applications in various sectors. This problem generalises the OP by letting collected profits being dependent on the duration of the visit at each of the nodes. In the fishing industry, fish populations may vary per location and fishing itself is a time-consuming operation, with additionally there often being a legal time limit on fishing. Another example is given for the entertainment industry, where various attractions can be visited on a day and thus, the satisfaction gained can safely be assumed to be time-dependent. One example from the literature comes from J. Yu, Aslam, Karaman, and Rus (2015), discussing optimal schedules for mobile sensing robots. In this problem, the routing of a mobile sensing robot through geographic Points Of Interests (POIs) needs to be determined. As these robots have limited fuel capacity and both traveling and information gathering cost time, a balance has to be struck between traveling to nodes and spending time at nodes. As Gunawan, Ng, Kendall, and Lai (2018) note, the aforementioned examples can easily be justified in the context of the Team OPSTP (TOPSTP) as well.

In this research, a new generalisation of the TOP is investigated, namely the TOPSTP with Time Windows (TOPSTPTW). Individually, the TOPSTP and the TOP with Time Windows (TOPTW) have been researched in the academic literature to varying degrees, but a conjunction of the two has not yet surfaced. Introducing time windows to the TOPSTP could have interesting theoretical implications, and could also be seen as a natural assumption in some applications. Consider the entertainment industry example from before, it may realistically be assumed that the operating hours of attractions may differ from one another. Would it be worth it to arrive too early and have to wait before being able to enter an attraction, even though you could have just as well visited another attraction during that time? Or does the addition of time windows significantly simplify the decision of which attraction are to be visited and in which order, or more importantly for how long. The aim of this research is to construct a solution method for the TOPSTPTW which is able to find solutions of good quality for large benchmark instances within a reasonable amount of time.

Given that research on the TOPSTPTW does not exist yet, benchmark instances have to be generated. Following Erdoğan and Laporte (2013) and Gunawan et al. (2018), TOPSTP instances are generated from available instances of Traveling Salesman Problem (TSP). Then, time windows are added following the procedure of Kantor and Rosenwein (1992), resulting in TOPSTPTW instances.

A mathematical formulation for the TOPSTPTW is presented, inspired by the works of Erdoğan and Laporte (2013), Gunawan, Lau, and Lu (2015b), Gunawan et al. (2018), and Labadie, Mansini, Melechovsky, and Calvo (2012). Moreover, a Hybrid Adaptive Large Neighbourhood Search (HALNS) heuristic is proposed as a solution approach, taking inspiration from Gunawan et al. (2018), Hammami, Rekik, and Coelho (2020), Labadie et al. (2012), Q. Yu, Adulyasak, Rousseau, Zhu, and Ma (2021), and Q. Yu, Fang, Zhu, and Ma (2019). The HALNS framework divides the problem into a routing and scheduling problem, where the former determines which locations are visited in which order, while the latter determines how long each of the selected locations should be visited. The Service-Time Scheduling Problem (SSP) is solved using the general purpose solver `CPLEX` and an adjusted version of the Modified Coordinate Search (MCS) heuristic proposed by Q. Yu et al. (2021).

It is found that the HALNS solution approach where the SSP is solved exactly is a reliable method to solve the TOPSTPTW, and finds improved solutions compared to `CPLEX` for virtually all test instances within four minutes. The HALNS algorithm where the SSP is solved with the MCS heuristic is a quick solution method, finding solutions within 30 seconds for instances with 25 or more nodes. However, the solution quality is far more inferior than when the SSP is solved exactly. Additionally, the MCS heuristic is not successful for instances with less than 25 nodes, showing significantly increased solving times for some instances. Due to the computational difficulty of the TOPSTPTW, the upper bounds obtained by `CPLEX` are rather weak, and thus no real conclusions can be made about the relative optimality gap of the obtained solutions.

This paper is outlined as follows. A detailed description of the TOPSTPTW is given in Section 2, followed by a discussion of the relevant literature in Section 3. Subsequently, the mathematical formulation and heuristic solution approaches are presented in Section 4, and Section 5 provides an explanation on the generation procedure of the required benchmark instances. Then, the obtained results are presented and discussed in Section 6, after which Section 7 concludes the paper with conclusions and a discussion.

2 Problem Description

A broad description of the goal of the TOPSTPTW would be to determine multiple distinct tours over a subset of nodes in a graph, each starting and ending at the depot and being restricted by a maximum travel time. Simultaneously, the visitation duration at each of the visited nodes has to be determined in order to maximise obtained profit, and additionally each of the nodes has a time window during which profits can be collected.

To be more precise, let us first represent the problem on an undirected graph $G = (V, E)$, where $V = \{1, 2, \ldots, |V|\}$ denotes the set of nodes and E denotes the corresponding set of edges. The set of nodes $V \setminus \{1\}$ can potentially be visited, while node 1 functions as the depot. Each edge $(i, j) \in E$ is associated with a travel cost $t_{ij} \geq 0$, which are assumed to satisfy the triangle inequality and to be symmetrical, i.e. $t_{ij} = t_{ji}$. Each node $i \in V \setminus \{1\}$ is associated with a maximum attainable profit $r_i \geq 0$, a profit collection parameter $\beta_i \geq 0$, an earliest available time $e_i \geq 0$, and a latest available time $l_i \geq e_i$. Upon visiting a node, a percentage of the maximum attainable profit r_i is collected dependent on the profit collection function $f_i(z_{ip})$, where $z_i \geq 0$ denotes the duration of the visit at node $i \in V \setminus \{1\}$ for tour $p \in P$. To clarify, the set P consist of the required amount of tours which need to be determined, and each tour $p \in P$ should adhere to the imposed time limit $T_{max} \in \mathbb{N}$. The choice has been made for the concave increasing function $f_i(z_i) = 1 - e^{-\beta_i z_i}$, which captures the desired characteristic of diminishing returns over time. Due to the time windows, a theoretical upper bound on z_i is imposed, i.e. $z_{ip} \leq l_i - e_i$.

Ultimately, for each tour $p \in P$ it has to determined which nodes it should visit and for how long such that the travel and visitation time does not exceed T_{max}, while maximising the profits obtained over all visited nodes of all tours in P and adhering to the imposed time windows at each of the nodes. Note that each tour must start and end at the depot, and all nodes in $V \setminus \{1\}$ can be visited at most once.

3 Literature Review

The specific topic of the TOPSTP is a recent development in the academic literature, and consequently rather scarce. The OPSTP was first introduced in 2013 by Erdoğan and Laporte (2013), and the TOPSTP was only just introduced in 2018 by Gunawan et al. (2018). Subsequently, this literature review will also cover the Orienteering Problem (OP) and its generalisations, in order to get a better grasp of the subject and its history. Section 3.1 introduces the classical OP, and its generalisation where multiple tours are considered. Subsequently, Section 3.2 covers literature on the generalisation of the (T)OP where the collection of profits is dependent on the amount of time spent at each of the nodes. Section 3.3 extends on its preceding section by discussing literature where it is additionally imposed that the collection of profits is dependent on the arrival time at each of the nodes. Lastly, Section 3.4 treats literature where the collection of profits is not dependent on service time or arrival time, yet at each of the nodes time windows are present within which profit can be collected.

3.1 (Team) Orienteering Problem

The first mention of the OP may be attributed to Tsiligirides (1984), and it is still relevant to this day due to its proposed benchmark instances. Tsiligirides (1984) argues that the OP is similar to a Generalised Traveling Salesman Problem (GTSP), where not all cities have to be visited and the salesman can only travel a certain distance. Each city is associated with an expected amount of sales the salesman can make there, and the objective is to maximise the total amount of sales. Assuming travel time varies proportionally with distance, this GTSP formulation may be used to express the OP. Note that the GTSP is *NP*-hard, and subsequently the OP is *NP*-hard as well (Golden, Levy, and Vohra, 1987).

Tsiligirides (1984) provides two heuristic methods to solve the OP. A stochastic algorithm is proposed, which relies on a Monte Carlo method to generate routes. The second solution method is a deterministic algorithm based on a procedure for the vehicle-scheduling problem proposed by Wren and Holliday (1972). This approach restricts routes to sectors, within which the aim for a route is to visit all nodes, while minimising the total distance traveled and not violating the maximum allowed travel distance. The results of Tsiligirides (1984) show that the stochastic algorithm outperforms the deterministic algorithm. With the implementation of a three-step improvement phase, both algorithms show similar results and runtimes.

Research on the OP was kick-started at this point, such as Golden, Wang, and Liu (1988) incorporating the stochastic concept of Tsiligirides (1984) and centre of gravity idea of Golden et al. (1987), while simultaneously introducing learning capabilities. Exact methods for the OP also started to gain traction, with Laporte and Martello (1990) being the first using a branch-and-bound method, and Pillai (1992) claiming to be able to find the optimal solutions to the benchmark instances of Tsiligirides (1984) with her branch-and-cut framework. Perhaps more notable were Chao, Golden, and Wasil (1996a), introducing new benchmark instances and proposing a new improved heuristic.

The OP is a special case of the TOP in which only a single tour is considered, and as such the TOP is also *NP*-hard. Due to this difficulty, exact solution approaches have been relatively scarce. Butt and Ryan (1999) were the first with a set-partitioning formulation, which was solved using a combination of column generation and constraint branching. The other main approaches throughout the years have been branch-and-price (Boussier, Feillet, and Gendreau, 2007; Keshtkaran, Ziarati, Bettinelli, and Vigo, 2016), branch-and-cut (Bianchessi, Mansini, and Speranza, 2018; Dang, El-Hajj, and Moukrim, 2013), cutting planes (El-Hajj, Dang, and Moukrim, 2016), and branch-cut-and-price (Pessoa, Sadykov, Uchoa, and Vanderbeck, 2019; Poggi, Viana, and Uchoa, 2010). However, even in the most recent works, the exact algorithms are still time-consuming and resource-hungry. Moreover, numerous well-performing heuristic approaches have been proposed throughout the years, ranging from tabu searches (Archetti, Hertz, and Speranza, 2007; Tang and Miller-Hooks, 2005), to variable neighbourhood searches (Archetti et al., 2007; Vansteenwegen, Souffriau, van den Berghe, and van Oudheusden, 2009), ant colony optimisation (Ke, Archetti, and Feng, 2008), memetic algorithms (Bouly, Dang, and Moukrim, 2010), simulated annealing (Lin, 2013), and greedy randomised adaptive search procedures (Souffriau, Vansteenwegen, van den Berghe, and van Oudheusden, 2010), just to name some.

Butt and Cavalier (1994) construct an Integer Program (IP) and a heuristic for the Multiple Tour Maximum Collection Problem. The heuristic considers node pairs instead of single nodes. Each node pair is assigned a weight defined by a convex combination of two different weight criteria, which is then used in generating the routes.

In order to determine the quality of the solutions obtained with the heuristic, the IP was solved to obtain optimal solutions. The heuristic was able to obtain the optimal solution for 76% of the instances for which an exact solution was obtained, and the worst-case optimality gap was 5.7% for the remaining the instances. Overall, the heuristic was able to obtain a solution for instances with up to 100 nodes within approximately 70 seconds.

Although Butt and Cavalier (1994) did solve the TOP, they did not publish their test instances. Thus Chao, Golden, and Wasil (1996b) generated and published TOP instances by taking OP instances of Tsiligirides (1984) and Chao et al. (1996a) and dividing the time limit by $m \in \{2, 3, 4\}$ team members.

Dang, Guibadj, and Moukrim (2013b) propose a Particle Swarm Optimisation (PSO) inspired algorithm, based on the PSO memetic algorithm for the TOP by Dang, Guibadj, and Moukrim (2013a). For their algorithm they report a relative percentage error of only 0.0005% on the set of known test instances by Chao et al. (1996b). Consequently, it seemed like there was not much room for algorithmic improvement for these cases, and as such Dang, Guibadj, et al. (2013b) introduce larger test instances with more nodes to accommodate future research on the topic. These larger benchmark instances are based on a subset of OP instances of Fischetti, González, and Toth (1998), and use the transformation procedure of Chao et al. (1996b) to obtain the desired TOP instances.

Ke, Zhai, Li, and Chan (2016) provide an interesting Pareto Mimic Algorithm (PMA) for the TOP, utilising Pareto dominance (Voorneveld, 2003) and the newly introduced mimic and swallow operators. Ke et al. (2016) show that their PMA is on a par with the PSO-inspired algorithm by Dang, Guibadj, et al. (2013b), with PMA finding all previously known best solutions, while also being able to find 10 new best known solutions. Additionally, average results over 10 execution runs are in favour of PMA in regards of objective value and running time.

A more recent attempt at solving the TOP comes from Hammami, Rekik, and Coelho (2020), proposing a Hybrid Adaptive Large Neighbourhood Search (HALNS). Their approach is to hybridise an Adaptive Large Neighbourhood Search (ALNS), which on its own is an extension of a Large Neighbourhood Search (LNS) for the Vehicle Routing Problem (VRP) (Shaw, 1998). Hybridisation is achieved by introducing a Sub-Route Optimisation Problem (SROP) to find a more profitable (sub)sequence in a given constructed route, which corresponds to solving an OP. The algorithm is further hybridised by solving a Set Packing Problem (SPP) to determine the combination of distinct routes resulting in the highest profit among all generated routes. First, nodes are eliminated which on their own in a tour already exceed the time limit, which is done using a procedure proposed by El-Hajj et al. (2016). An initial solution is obtained using the nearest neighbour algorithm of Keller et al. (1985). The structure of the rest of the algorithm consists of *run segments*, in which the ALNS procedure takes place.

Hammami et al. (2020) compare the performance of their HALNS algorithm to 26 other TOP algorithms from the academic literature, and report promising results. The HALNS algorithm was able to find all best-known solutions for the small-scale instances of Chao et al. (1996b), and even find a new best-known solution for the large-scale instances of Dang, Guibadj, et al. (2013b). Perhaps more importantly, HALNS showed to be significantly faster in solving the smaller instances compared to other competitive state-of-the-art heuristics. Additionally, it was able to find the best-known solutions the quickest for 78% of the larger instances.

3.2 (Team) Orienteering Problem with Service-Time-Dependent Profits

Erdoğan and Laporte (2013) were the first to touch upon the OP with Service-Time dependent Profits (OPSTP), which generalises the OP such that the collection of profits is dependent on the duration of a visit at a node. Mathematical formulations are presented for discrete and continuous interpretations, where the former relies on *passes* to model the duration of time. A pass is defined as a specified amount of time, and multiple passes can be made at a node to signify a longer visit. The proposed continuous formulation results in a nonlinear program, which for their branch-and-cut algorithm is linearised to allow the dynamic addition of valid inequalities. Due to the inherent characteristics of profits being collected at nodes as a nonlinear function of time, methods developed for the (T)OP cannot be applied directly to solve the problem.

Being the first to discuss the OPSTP, Erdoğan and Laporte (2013) present schemes to transform Traveling Salesman Problem (TSP) instances obtained from TSPLIB to OPSTP instances. It is shown that the Mixed Integer Programming (MIP) formulation with discrete passes is able to solve all instances with up to 100 nodes within 5 minutes. For the continuous interpretations, this drops to 94% and 86% of the instances in the case of a concave and convex profit functions, respectively. Additionally, the latter case takes approximately three times as much computational time than the former, showcasing the difficulty of properly linearising the convex case.

Sometime later, Guitouni and Masri (2014) formulate a Search And Rescue (SAR) problem as an OPSTP, which is then solved using the LINGO solver. The only noteworthy difference of their formulation compared to that of Erdoğan and Laporte (2013) is that they model the subtour elimination problem using the Miller-Tucker-Zemlin formulation (Miller, Tucker, and Zemlin, 1960). However, the LINGO solver appears to struggle to further improve potentially optimal solutions past 98%, after which each of the remaining percentage points shows a significantly increased runtime. Guitouni and Masri (2014) admit that linearisation of the problem as done by Erdoğan and Laporte (2013) and the use of the `CPLEX` solver may possibly reduce runtime.

J. Yu et al. (2015) discuss the OPST in the context of scheduling for mobile sensing robots, proposing both primal and dual MIP formulations supporting arbitrary profit functions through piece-wise linearisation. It is shown that a $(1+\epsilon/2)$ piece-wise linear approximation of any arbitrary profit function results in $(1+\epsilon)$-optimal solution to the original primal problem, given $\forall\epsilon \in \mathbb{R}^+$.

J. Yu et al. (2015) test their formulations on two sets of instances. The first set of instances consist of POIs located on the lattice points of a rectangular grid, while the second one consists of POI locations randomly generated in a rectangle using a uniform distribution. The latter set of instances appears to be more difficult to solve, with up to 40 POIs being attainable for those instances, while for the former set of instances up to 200 POIs is still feasible.

Q. Yu, Fang, Zhu, and Ma (2019) propose a two-phase matheuristic for the OPSTP. The problem is decomposed into routing and scheduling subproblems, which are solved iteratively. The goal of the routing problem is to determine the subset of nodes to be visited and the corresponding sequence. Consequently, the Service-Time Scheduling Problem (SSP) has to determine the optimal service time at each of the nodes in a given route. The SSP is similar to a continuous knapsack problem with nonlinear objective, and Q. Yu et al. (2019) develop an exact polynomial-time algorithm with the use of the Karush-Kun-Tucker conditions. The proposed algorithm consists of a tabu search to find feasible solutions for the routing problem, and the SSP is solved either with the nonlinear solver `IPOPT` or the proposed polynomial-time algorithm. The LS deterministic-removal operator is specifically designed for the OPSTP, as it removes any node having an optimal service time of zero. Due to the many executions of the SSP, it is important that it can be solved quickly for the entire algorithm to be efficient.

Results are compared to the algorithm of Erdoğan and Laporte (2013), and due to developments in general purpose solvers also a reimplementation thereof. The proposed two-phase matheuristic is able to obtain results for all instances with up to 100 nodes within 30 seconds. At times the (reimplementation of the) algorithm of Erdoğan and Laporte (2013) has a better objective, but that comes at the cost of a significantly higher runtime. On the other hand, Q. Yu et al. (2019) are able to obtain results for all instances within two minutes, with a maximum optimality gap of 10.7%. The algorithm is also compared to that of J. Yu et al. (2015) on their proposed test instances, and it shows that the two-phase matheuristic is able to outperform it in terms of obtained objective values and in runtime. Demonstrating the efficiency of their proposed polynomial-time algorithm for the SSP, its average runtimes are compared to that of `IPOPT` and shown to be equal to approximately a fraction of a second and seven seconds, respectively.

Only recently has research on the Team OPSTP (TOPSTP) surfaced, with the proposal of an ILS algorithm by Gunawan et al. (2018). The TOPSTP is first formulated as an IP, inspired by the discrete model by Erdoğan and Laporte (2013). The ILS follows the same structure and operators as that of Chao et al. (1996b). The *most efficient* criterion is introduced just for the TOPSTP, and is used to construct new tours during the initialisation and reinitialisation phases, and the first LS operator, as well as during the second LS operator to determine how many discrete passes should be made at a node. As this formulation is dependent on discrete passes, there is no SPP as with Q. Yu et al. (2019).

Since the TOPSTP has not been researched before, benchmark instances needed to be generated. As such, Gunawan et al. (2018) present formulae to transform the OPSTP instances of Erdoğan and Laporte (2013) to TOPSTP instances. In order to determine the relative performance of the ILS procedure, the proposed IP is solved with `CPLEX` with a time limit of 1,000 seconds to obtain upper bounds. The ILS procedure was able to obtain results for all instances with up to 200 nodes within approximately 12 seconds, having a maximum gap with respect to the upper bound of 37.92%.

3.3 Team Orienteering Problem with Service- and Arrival-Time-Dependent Profits

Under the umbrella of (T)OP with variable profits also falls the (T)OP with Arrival-Time Dependent Profits ((T)OPATP). Research on the (T)OPATP has been around for a longer time, with Erkut and Zhang (1996) introducing the OPATP, and Tang, Miller-Hooks, and Tomastik (2007), Ekici and Retharekar (2013) and Afsar and Labadie (2013) discussing the TOPATP. However, Q. Yu, Adulyasak, Rousseau, Zhu, and Ma (2021) propose a combination of the TOPATP and TOPSTP, ergo the TOP with Service- and Arrival-Time-Dependent Profits (TOP-TTP). The motivation for this specific problem comes from SAR operations, where the probability of finding survivors depends on when a SAR team arrives at a location and how long the team searches that location. Q. Yu et al. (2021) formulate a Mixed Integer Non-Linear Program (MINLP), propose a Benders Cranch-and-Cut (BBC) algorithm to solve it, and also provide an ILS approach. Just as Q. Yu et al. (2019), an SSP has to be solved during the procedure, and thus Q. Yu et al. (2021) propose a heuristic Modified Coordinate Search (MCS) algorithm for the SSP. This derivative-free method iteratively attempts to find and move to a search direction which will improve the objective value of a given solution.

TOP-TTP benchmark instances did not exist yet, so Q. Yu et al. (2021) used a subset of the Vehicle Routing Problem with Time Window (VRPTW) instances of Solomon (1987) to generate them. These instances have up to 25 nodes (excluding depots) and up to 5 vehicles, i.e. tours. The instances are also divided into two sets, based on the nodes being cluster-located or random-located. Q. Yu et al. (2021) show that their ILS with MCS is generally able to find better solutions than their proposed BBC algorithm, and in considerably less computational time as well. For the larger instance, BBC has runtimes of up to more than two hours compared to ten minutes for the ILS with MCS. In addition, an optimality gap of 181.51% is observed for the BBC for the largest instance, while this is merely 32.04% for the ILS with MCS. Looking at the SSP, the proposed MCS is able to construct solutions of similar quality as the `IPOPT` solver, but only needs 2.74% of the computational time.

3.4 (Team) Orienteering Problem with Time Windows

Kantor and Rosenwein (1992) were first to discuss the OPTW, proposing a tree search-based heuristic and an insertion heuristic, where the latter is an extension of the heuristic developed by Laporte and Martello (1990) for the OP. In order to test the performance of the proposed heuristics, Kantor and Rosenwein (1992) provide formulae to generate OPTW instances from the OP instances of Tsiligirides (1984). Given that the tree-search based heuristic explores a larger solution space, it proved to provide better results than the insertion heuristic, with acceptable runtimes for heavily-constrained, moderately-sized problems.

Labadie et al. (2012) propose an LP-based Granular Variable Neighbourhood Search (VNS) for the TOPTW. An insertion heuristic based on a custom criterion is used to generate an initial solution. The first VNS neighbourhood consist of classical routing operators to reduce travel time, while the second one aims to replace a sequence of visited nodes by unvisited ones in order to increase profitability. The LP-based Granular component of the proposed heuristic involves the dual formulation of an assignment problem to identify more promising edges, in order to reduce the size of the second VNS neighbourhood.

Gunawan et al. (2015b) present an ILS for the TOPTW. A greedy construction heuristic is proposed to obtain an initial solution, making use of an insertion benefit ratio. The LS component consists of six operations, both inter- and intraroute, in order to reduce travel time and improve the objective. The perturbation component consists of a route exchange and a shake procedure aims to escape local minima. Additionally, an intensification strategy is implemented, which reinitialises the algorithm using the best-found solution if no improvement has been found for a predetermined amount of iterations. This ILS framework is extended by Gunawan, Lau, and Lu (2015a) to incorporate an SA component, resulting in the possibility of worse solution being accepted based on a SA criterion, further aiding escaping local minima. The SA ILS shows promising results compared to other competitive algorithm from the literature, while also being able to find new best-known solutions for the TOPTW.

This chapter discussed the OP and some of its various generalisations. In the next chapter, the (linearised) objective function for the OPSTP put forward by Erdoğan and Laporte (2013) will form the basis of the TOPSTPTW formulation. Accommodating multiple routes in the TOPSTPTW formulation follows follows the structure of the TOPSTP formulation of Gunawan et al. (2018). Incorporating time windows in the formulation of the TOPSTPTW is inspired by the formulations for the TOPTW by Gunawan et al. (2015b) and Labadie et al. (2012). Subsequently, the HALNS framework for the TOP proposed by Hammami et al. (2020) will form the the basis for the heuristic solution method. To incorporate service-time-dependent profits, the construction of an initial solution takes inspiration from Q. Yu et al. (2021), Q. Yu et al. (2019). Due to presence of time windows, the proposed local search procedure takes notes from Gunawan et al. (2015b). The SSP is also present in the HALNS framework for the TOPSTPTW. An exact mathematical formulation for the SSP is proposed, as well as an MCS heuristic based on that of Q. Yu et al. (2021).

4 Methodology

This section will first introduce a mathematical formulation for the Team Orienteering Problem with TOPSTPTW inspired by the works of Erdoğan and Laporte (2013), Gunawan et al. (2015b), Gunawan et al. (2018), and Labadie et al. (2012). A solution approach based on the HALNS framework for the TOP of Hammami et al. (2020) is discussed. This framework is modified for the TOPSTPTW, taking notes from Gunawan et al. (2015b) and Q. Yu et al. (2021), Q. Yu et al. (2019). Furthermore, a mathematical formulation is presented for the SSP, and an adjusted version of the MCS proposed by Q. Yu et al. (2021) is introduced in order to solve it heuristically.

4.1 Mathematical Formulation

Given that we have to keep track of when service time starts at a node, it is necessary to define a virtual copy of the depot 1, denoted by v. To that extent, let us define the set of nodes containing both depots as $\hat{V} = V \cup \{v\}$, and the corresponding new set of edges $\hat{E}$ (Labadie et al., 2012). Furthermore, this brings the need to define the $e_i = 0, l_i = T_{max}$ for the nodes $i \in \{1, v\}$, as well as $t_{1v} = \infty$. Additionally, for convenience we set $t_{ij} = \infty$ if $e_i + t_{ij} > l_j$.

$$
\begin{aligned}
\max \quad & \sum_{p\in P} \sum_{i\in V\setminus\{1\}} f_i(z_{ip}) && && (1)\\
\text{s.t.} \quad & \sum_{j:(1,j),(v,j)\in\hat{E}} \sum_{p\in P} x_{1jp} = |P| && && (2)\\
& \sum_{i:(i,v),(i,1)\in\hat{E}} \sum_{p\in P} x_{ivp} = |P| && && (3)\\
& \sum_{p\in P} y_{ip} \leq 1 && \forall i \in V \setminus \{1\} && (4)\\
& \sum_{j:(k,j)\in\hat{E}, j\neq k} x_{kjp} = y_{kp} && \forall k \in V \setminus \{1\}, p \in P && (5)\\
& \sum_{i:(i,k)\in\hat{E}, i\neq k} x_{ikp} = y_{kp} && \forall k \in V \setminus \{1\}, p \in P && (6)\\
& \sum_{(i,j)\in\hat{E}} t_{ij} x_{ijp} + \sum_{i\in\hat{V}} (z_{ip} + w_{ip}) \leq T_{max} && \forall p \in P && (7)\\
& T_{ip} + z_{ip} + t_{ij} - T_{jp} \leq M^1_{ij}(1 - x_{ijp}) && \forall (i,j) \in \hat{E}, p \in P && (8)\\
& T_{jp} - (T_{ip} + z_{ip} + t_{ij}) \leq M^2_j(1 - x_{ijp}) + w_{jp} && \forall (i,j) \in \hat{E}, p \in P && (9)\\
& e_i y_{ip} \leq T_{ip} \leq l_i y_{ip} && \forall i \in \hat{V}, p \in P && (10)\\
& 0 \leq z_{ip} \leq l_i - T_{ip} && \forall i \in \hat{V}, p \in P && (11)\\
& y_{ip} = 1 && \forall i \in \{1, v\} && (12)\\
& w_{ip} \geq 0 && \forall i \in \hat{V}, p \in P && (13)\\
& y_{ip} \in \{0, 1\} && \forall i \in \hat{V}, p \in P && (14)\\
& x_{ijp} \in \{0, 1\} && \forall (i,j) \in \hat{E}, p \in P && (15)
\end{aligned}
$$

Equations (1)-(15) represent the mathematical formulation for the TOPSTPTW. The decision variable T_{ip} denotes the start of service time at node i on tour p; z_{ip} denotes the service time duration at node i for tour p; y_{ip} is equal to one if node i is visited on tour p, zero otherwise; x_{ijp} is equal to one when node j is visited directly after node i on tour p, zero otherwise. To incorporate a tour arriving at a node before the earliest available time, w_{ip} denotes the waiting time of a tour p before entering node i in case it arrives before the earliest available time e_i (Gunawan et al., 2015b). The MINLP represented in equations (1)-(15) may be solved using a Non-Linear solver such as `IPOPT`. However, it may be more efficient to linearise the MINLP following Erdoğan and Laporte (2013) and solve it with the general purpose solver `CPLEX`.

The objective function (1) maximises the total collected profit according to the service time at all visited nodes in all tours, where $f_i(z_{ip}) = r_i(1 - e^{-\beta z_{ip}})$. Constraints (2) and (3) ensure that all tours start and end at the depot, respectively. Constraints (4) states that each node, with exception of the depot, can at most be visited once. Constraints (5) and (6) enforce connectivity between edges and vertices. Constraints (7) ensure that the travel, service and waiting time of each tour does not exceed the given time limit. Constraints (8) ensure that the consecutively visited nodes have their starting times of service updated correctly. Similarly, constraints (9) ensure that the waiting time at a node is updated correctly, if it arrives before the earliest available time (Gunawan et al., 2015b). The aforementioned two sets of constraints also eliminate the creation of subtours, similar to the Miller-Tucker-Zemlin formulation for subtour elimination (Miller et al., 1960). Constraints (10) enforce the time windows at each of the nodes. Constraints (11) impose bounds on the service times. Lastly, constraints (13) ensures non-negativity of the waiting time, and constraints (14)-(15) are integrality constraints. Taking inspiration from Labadie et al. (2012), M^1_{ij} is set according to a theoretical upper bound on the left hand side of inequalities (8) for each $(i, j) \in \hat{E}$. This yields a stronger formulation compared to the case where arbitrarily high M-values are chosen. Subsequently, its values are $M^1_{ij} = 2l_i + t_{ij}$, since $T_{ip} + z_{ip} + t_{ij} - T_{jp} \leq l_i + (l_i - e_i) + t_{ij} - e_j \leq 2l_i + t_{ij}$, for each $(i, j) \in \hat{E}$. Analogously, it may be deduced for inequalities (9) that $M^2_j = l_j$, for each $(i, j) \in \hat{E}$.

4.2 Linearisation

In order to solve the TOPSTPTW with the general purpose solver CPLEX, the introduced formulation has to be linearised. Following the linearisation for the OPSTP proposed by Erdoğan and Laporte (2013), the auxiliary variable u_{ip} is added, accompanied by the valid inequalities shown in (16)-(17). The decision variable u_{ip} denotes the ratio of profit to be collected at node i for route p. Inequalities (16) define the tangents of each of the profit collection functions, defined for each integer value of the parameter z^*_{ip} on the interval $[0, l_i - e_i]$, for each $i \in V\{1\}, p \in P$. As Erdoğan and Laporte (2013) argue, this approach accurately approximates the chosen nonlinear profit collection functions. Additionally, inequalities (17) impose an upper bound on each of the profit collection functions according to the corresponding time windows.

$$u_{ip} \leq \beta_i e^{-\beta z^*_{ip}} z_{ip} + 1 - \beta_i e^{-\beta z^*_{ip}} - z^*_{ip} \beta_i e^{-\beta z^*_{ip}} \qquad \forall i \in V \setminus \{1\}, p \in P, z^*_{ip} \in [0, l_i - e_i] \tag{16}$$

$$u_{ip} \leq (1 - e^{-\beta(l_i - e_i)}) y_{ip} \qquad \forall i \in V \setminus \{1\}, p \in P \tag{17}$$

Then, the linearised formulation for the TOPSTPTW consists of

$$\max \sum\nolimits_{p \in P} \sum\nolimits_{i \in V \setminus \{1\}} r_i u_{ip} \tag{18}$$

$$\text{s.t. } 0 \leq u_{ip} \leq 1 \qquad \forall i \in V \setminus \{1\}, p \in P \tag{19}$$

and (2)-(17), where r_i denotes the profit associated with visiting node $i \in V \setminus \{1\}$.

4.3 Hybrid Adaptive Large Neighbourhood Search

The HALNS procedure consists of node selection strategies and removal/insertion operators, each of which is adaptively chosen based on past performance. Each strategy or operator k is associated a score π_{kq} during each run segment q of the algorithm, set equal to one for $q = 1$, and updated following $\pi_{k(q+1)} = \lambda\overline{\pi_{kq}}/n_k + (1-\lambda)\pi_{kq}$. A run segment is defined as a subsection of the algorithm during which N_{max} iterations of ALNS is performed, and the scores of each operator and strategy are updated per run segment. n_k denotes number of the times during a run segment q that a strategy or operator k has been chosen, $\overline{\pi_{kq}}$ the observed score, and $\lambda \in (0,1)$ a specified reaction factor. The observed score $\overline{\pi_{kq}}$ is initially set to zero and incremented according to ρ as per (20). Ultimately, each strategy and operator is chosen using roulette wheel selection, i.e. according to the probability $\pi_{kq}/\sum_{l=0}^{m}\pi_{lq}$, where m denotes the total available strategies/operators with respect to k.

$$\rho = \begin{cases} \rho_1 & \text{if new solution is a new best} \\ \rho_2 & \text{if the new solution value is better than the last admissible solution} \\ \rho_3 & \text{if the new solution does not improve the last admissible solution} \end{cases} \tag{20}$$

A general outline of the used HALNS framework is given in Algorithm 1. Iteratively, a random amount of nodes Γ to be removed from a current solution P according to the selected operator. Subsequently, a random amount of nodes Δ is selected and inserted according to the chosen selection strategy and operator. If the objective value of a modified solution P is greater or equal to the objective value of the admissible solution P_{adm}, a local search is performed with the aim to reduce travel time and increase obtained profits. A Simulated Annealing (SA) temperature parameter S is also employed to occasionally admit a solution with a worse objective value than the admissible solution. The SA parameter is initialised to S_0, and updated through multiplication of a cooling factor c during each ALNS iteration. If a new best solution is found, P_{best} is updated and the counter of consecutively generated non-improving solutions n_{best} is reset, otherwise the counter is incremented. If a modified solution P is admitted, P_{adm} is updated to it, regardless of whether it is a new best or not. Additionally, if the SA parameter value falls below a specified minimum S_{min}, a Set Packing Problem (SPP) is solved in order to determine the best combination of tours among all generated tours up to that point. The algorithm stops when no improved best-known solution has been found for $N_{best_{max}}$ iterations, or when Q_{max} run segments is reached. Then the SPP is solved one last time and the best-known solution is given as output. Note that the SSP is solved each time a solution is modified in order to obtain the corresponding objective value, and the objective value of a solution P is denoted by $v(P)$.

Algorithm 1: HALNS Framework for the TOPSTPTW

```
1  NodeElimination
2  $P \leftarrow$ TourInitialisation
3  $P \leftarrow$ SolveSSP($P$)
4  $P_{best} \leftarrow P, P_{adm} \leftarrow P, S \leftarrow S_0, q \leftarrow 0, n_{best} \leftarrow 0$
5  ScoreInitialisation
6  while $q < Q_{max}$ and $n_{best} < N_{best_{max}}$ do
7      $n \leftarrow 0$
8      while $n < N_{max}$ do
9          $P \leftarrow P_{adm}$
10         Generate $\Gamma$, $\Delta$
11         Select node selection strategy $\gamma$
12         Select removal and insertion operators $R$ and $I$
13         Remove $\Gamma$ nodes in $P$ using $R$
14         Insert $\Delta$ nodes in $P$ using $I$ according to $\gamma$
15         $P \leftarrow$ SolveSSP($P$)
16         Generate random number $\delta \in (0, 1)$
17         if $v(P) \geq v(P_{adm})$ or $\delta \leq e^{(v(P)-v(P_{adm}))/S}$ then
18             $P \leftarrow$ LocalSearch($P$)
19             $P \leftarrow$ SolveSSP($P$)
20             if $v(P) > v(P_{best})$ then
21                 $P_{best} \leftarrow P, n_{best} \leftarrow 0$
22             else
23                 $n_{best} \leftarrow n_{best} + 1$
24             $P_{adm} \leftarrow P$
25         else
26             $n_{best} \leftarrow n_{best} + 1$
27         ObservedScoreUpdate
28         if $S \leq S_{min}$ then
29             $S \leftarrow S_0$
30             $P \leftarrow$ SolveSPP
31             if $v(P) > v(P_{best})$ then
32                 $n_{best} \leftarrow 0$
33             $P_{best} \leftarrow P, P_{adm} \leftarrow P$
34         $S \leftarrow S \cdot c, n \leftarrow n + 1$
35     ScoreUpdate
36     $q \leftarrow q + 1$
37 $P_{best} \leftarrow$ SolveSPP
38 return $P_{best}$
```

4.3.1 Initialisation

The method `NodeElimination` eliminates nodes which on their own in a tour already exceed the time limit, i.e. nodes $i \in V$ s.t. $t_{1i} + t_{iv} \geq T_{max}$. An initial solution is then generated in the `TourInitialisation` method, in which $|P|$ tours are constructed simultaneously (Q. Yu et al., 2021). To incorporate variations in profit collection rates and travel distances, each node is associated with a score defined by $\zeta_{ip} = r_i \beta_i (l_i - e_i)/\Delta t_{ip}$, taking inspiration from Q. Yu et al. (2019). Δt_{ip} corresponds to the incremental increase in travel time of adding node i at the end of route p. An example would be that of adding node i to tour $p = \{1, \ldots, k, v\}$, resulting in tour $p = \{1, \ldots, k, i, v\}$. In that case, $\Delta t_{ip} = t_{ki} + t_{iv} - t_{kv}$. This method gives a higher priority to nodes having a larger profit, profit collection rate and/or time window, or those which minimally increase the total travel time. Each of the $|P|$ tours is first set to only contain the depot as start and end nodes, i.e. $\forall p \in P : p = \{1, v\}$. Then, iteratively it is tried to insert a node i with the highest score ζ_{ip} to each of the tours $p \in P$. If insertion results in an infeasible route exceeding the time limit without considering service time of node i, that route is considered complete. We may assume that service time at a node should be greater than zero, and Q. Yu et al. (2021), Q. Yu et al. (2019) argue that setting it to collect 50% of the available profit is reasonable during initialisation. Consequently, in initialising the service times, z_{ip} is set such that $f_i(z_{ip}) = 0.5$ if the time limit or time window permits it, otherwise z_{ip} is set to the remaining available time with respect to the time limit or time window. If the arrival time of a node is not within its time window, the next node with the highest score will be sought which does satisfy the time window. If no such node can be found, the node having the lowest waiting time among the non-inserted nodes is considered for insertion.

4.3.2 Node Selection Strategy & Removal/Insertion Operators

Hammami et al. (2020) propose and argue the effectiveness of four node selection strategies and five insertion/removal operators for their TOP HALNS algorithm. Subsequently, these options are considered in this HALNS implementation as well, and are expected to work properly after adjusting for time windows and profit collection rates where necessary. The four considered node selection strategies are as follows:

1. NS1: This strategy extends ζ_{ip} to incorporate adding node i into a solution such that total travel time is minimally increased. To that extent we introduce $D(P^{+i})$, which denotes the total travel time of inserting node i into solution P such that increase in the total travel time is minimal (Hammami et al., 2020). Then, the first node to be selected is node $i^* \in \hat{V} \setminus V_P : i^* = \text{argmax}_{i \in \hat{V} \setminus V_P} r_i \beta_i (l_i - e_i)/D(P^{+i})$, taking time windows and profit collection rates into account.

2. NS2: This strategy prioritises nodes having the highest obtainable profit according to time windows and profit collection rates, $r_i \beta_i (l_i - e_i)$. Selection is made using a roulette wheel procedure in order to avoid cycling, giving higher probabilities to nodes with higher profits.

3. NS3: This strategy randomly selects nodes in order to diversify the search.

4. NS4: This strategy selects nodes which have been removed from previous solutions the last, in order to give them a chance at getting reinserted at better positions.

The five removal operators are as follows:

1. R1: This operator randomly removes nodes from the current solution in order to diversify the search.

2. R2: This operator removes the nodes having the smallest obtainable profit according to time windows and profit collection reates, $r_i \beta_i (l_i - e_i)$. Selection is made using a roulette wheel procedure in order to avoid cycling, giving higher probabilities to nodes with smaller profits.

3. R3: This operator, similarly to NS1, defines $D(P^{-i})$ as the largest decrease in travel time if node i is removed from solution P. Then, nodes are removed using roulette wheel selection based on

the largest saving in travel time, giving higher probabilities to nodes having a higher value for $D(P) - D(P^{-i})$.

4. R4: This operator randomly selects and removes all nodes in a tour $p \in P$ in order to diversify the search.
5. R5: This operator removes a sequence of subsequent nodes of random length α^p in a randomly selected tour $p \in P$, in order to create room for potentially more profitable sequence of non-inserted nodes.

Lastly, the five considered insertion operators are as follows:

1. I1: This operator iteratively inserts nodes at the earliest feasible position in a tour.
2. I2: This operator iteratively inserts nodes at the latest feasible position in a tour.
3. I3: This operator iteratively inserts nodes in a randomly selected feasible position.
4. I4: This operator iteratively inserts nodes at a feasible position minimising total travel time.
5. I5: This operator iteratively inserts nodes at a feasible position minimising the sum of travel time between the considered node and its predecessor and between the considered node and its successor within a route.

To clarify, an insertion is feasible if it respects time windows and does not exceed the time limit. Respecting time windows entails that it is possible for the tour to arrive at the inserted node before its latest available time, and that it is able to arrive before the latest available time of each of the subsequent nodes in the tour, disregarding service times. This has serious implications for the algorithm, since it is not unreasonable to assume that a large proportion of the nodes selected for insertion cannot be feasibly inserted. Subsequently, the Δ-parameter should always be larger than the Γ-parameter to ensure that generated routes do not continuously decrease in size.

4.3.3 Local Search

The LS procedure consists of two swap operators, a complete 2-opt procedure, and a move operator (Gunawan et al., 2015b). The first swap operator exchanges two nodes within a single tour, starting with the tour having the lowest remaining available service time $\hat{T}$. The second swap procedure extends the first one to swap two nodes within two tours, starting with those tours having the lowest and second lowest values for $\hat{T}$. The 2-opt procedure starts by selecting the tour having the lowest value for $\hat{T}$. The move operator reallocates a node from one tour to another. All aforementioned operators consider all possible combinations of differing nodes, and apply changes only if it results in an increase in $\hat{T}$ and does not violate any constraints. For each of the tours $p \in P$, these procedures are iteratively executed until no further improvement is found. Lastly, the I5 operator is used to try to insert non-inserted nodes into the current solution according to NS1 until an infeasible insertion is encountered, as proposed by Hammami et al. (2020).

4.3.4 Service-Time Scheduling Problem

When a route is modified at any point in the HALNS framework, the service times at each of its nodes has to be determined in order to obtain the corresponding objective value. Let us define V_p the set of nodes visited by tour p, and E_p the corresponding set of edges between consecutive nodes. Decision variables are identically defined as in model (1)-(15). For each tour $p \in P$, the SSP is formulated as per equations (21)-(27).

$$\max \sum\nolimits_{i \in V_p \setminus \{1,v\}} r_i(1 - e^{\beta_i z_{ip}}) \tag{21}$$

$$\text{s.t.} \sum\nolimits_{(i,j) \in E_p} t_{ij} + \sum\nolimits_{i \in V_p \setminus \{1,v\}} (z_{ip} + w_{ip}) \leq T_{max} \tag{22}$$

$$T_{ip} + z_{ip} + t_{ij} - T_{jp} \leq 0 \qquad \forall (i,j) \in E_p \tag{23}$$

$$T_{jp} - (T_{ip} + z_{ip} + t_{ij}) \leq w_{jp} \qquad \forall (i,j) \in E_p \tag{24}$$

$$e_i \leq T_{ip} \leq l_i \qquad \forall i \in V_p \tag{25}$$

$$0 \leq z_{ip} \leq l_i - T_{ip} \qquad \forall i \in V_p \setminus \{1,v\} \tag{26}$$

$$w_{ip} \geq 0 \qquad \forall i \in V_p \tag{27}$$

The objective function (21) maximises the total collected profit according to the service time at all visited nodes of the considered tour. Constraints (23) ensure that the consecutively visited nodes have their starting times updated correctly. Similarly, constraints (24) ensure that the waiting time at a node is updated correctly, if it arrives before the earliest available time. Constraints (25) enforce the time windows at each of the nodes. Constraints (26) impose bounds on the service times. Lastly, constraint (27) ensures non-negativity of the waiting time. Given that the objective function is equivalent to to that of model (1)-(15), the same linearisation scheme by Erdoğan and Laporte (2013) can be applied. Subsequently, the implemented model consists of objective (18) subject to (16), (17), (19), and (22)-(27) for a given tour $p \in P$ and corresponding set of nodes V_p and set of edges E_p. The version of the HALNS algorithm where the SSP is solved exactly with the general purpose solver `CPLEX` using the aforementioned formulation will be denoted by HALNS-E.

The SSP will have to be solved numerous times during the HALNS algorithm. Subsequently, an MCS heuristic will be implemented, with the aim to decrease overall solving time while not substantially diminishing solution quality. Q. Yu et al. (2021) propose an MCS heuristic to solve the SSP for the TOP-TTP, which is adjusted to ignore arrival-time dependent profits and incorporate time windows. The outline of the MCS heuristic is shown in Algorithm 2.

The MCS method starts from an initial service time solution, and iteratively generates improved solutions according to a search direction. The search direction is based on the product of two factors, where the first one depends on objective values of neighbour solutions with respect to the current solution, and the second one depends on the distance between a neighbour solution and the current solution. If a modification results in the violation of the time limit, or time windows for that matter, a repair procedure is executed, reducing service times until the solution is feasible again.

Algorithm 2: Modified Coordinate Search

```
   Input: Sequence of nodes $U = \{u_1, \dots, u_m\}$
1  $z^0 = \{z^0_1, \dots, z^0_m\} \leftarrow$ ServiceTimeInitialisation
2  $k = 0$, $z^* = z^k$
3  while $\epsilon > \epsilon_{min}$ do
4      Generate $D^k = \{z^k + e_i : i = 1, \dots, m\}$
5      Calculate $\vec{h}$ as per (29)
6      Compute $s'$ as per (28)
7      for node $u_i \in U$ do
8          if $a_{u_i} < e_{u_i}$ according to $s'$ then
9              $w_{u_i} = e_{u_i} - a_{u_i}$
10             $T_{u_i} = e_{u_i}$
11         else if $a_{u_i} \leq l_{u_i}$ according to $s'$ then
12             $T_{u_i} = a_{u_i}$
13         else
14             $\Delta T = a_{u_i} - l_{u_i}$
15             for $j \leftarrow i - 1$ to 1 do
16                 $\Delta z'_j = \min\{z'_j, \Delta T\}$
17                 $z' = \{z'_1, \dots, z'_j - \Delta z'_j, \dots, z'_m\}$
18                 UpdateTimes($j$,$i$)
19                 $\Delta T = a_{u_i} - l_{u_i}$
20                 if $\Delta T \leq 0$ then
21                     $T_{u_i} = a_{u_i}$
22                     break;
23         $\hat{z}'_i = \min\{z'_i, l_{u_i} - T_{u_i}\}$
24         $s' = \{z'_1, \dots, \hat{z}'_j, \dots, z'_m\}$
25     if $v(z') > v(z^k)$ then
26         $z^* = z'$, $k = k + 1$, $z^k = s'$
27     else
28         $\epsilon = \epsilon/2$, $k = k + 1$
29 return $z^*$
```

We denote the ordered set of nodes of a route p by $U = u_1, \dots, u_m$, wherein the depots are excluded. Then, the method `ServiceTimeInitialisation` tries to initialise the service times of each node $u_i \in U, i = 1, \dots, m$ such that $z_i = (T_{max} - \sum_{(i,j) \in E_p} t_{ij})/m$. Due to the time windows, this allocation is not feasible in most cases, and as such the same method as in lines (7)-(24) of Algorithm 2 will be used to ensure feasibility. Let us first consider generating a trail solution s'. A set of candidate solutions is constructed such that $D^k = \{z^k + e_i : i = 1, \dots, m\}$. Then the trail solution is generated according to (28), where $\vec{d}$ corresponds to $\vec{h}/\|\vec{h}\|$.

$$s' = s^k + \epsilon \vec{d} \tag{28}$$

The vector $\vec{h}$ is generated according to (29), consisting of the two factors w_l and $\vec{\pi_l}$ defined for each candidate solution $z_l \in D^k$. The first factor w_l is based on a weighted difference in objective value for each of the candidate solutions $z_l \in D^k$ with respect to the current MCS solution z^k. The second factor $\vec{\pi_i}$ is defined by a weighted difference in solution distance for each of the candidate solutions $z_l \in D^k$ with respect to the current MCS solution z^k. Q. Yu et al. (2021) remark that the trail solution is guaranteed to find an improved solution, although some constraints may be violated.

$$\vec{h} = \sum\nolimits_{l=1}^{m} w_l \vec{\pi_l} \tag{29}$$

$$w_l = \frac{v(z_l) - v(z^k)}{\sum_{q=1}^{m} |v(z_q) - v(z^k)|} \qquad \forall z_l \in D^k \tag{30}$$

$$\vec{\pi_l} = \frac{z_l - z^k}{\|z_l - z^k\|} \qquad \forall z_l \in D^k \tag{31}$$

Subsequently, a repair method is used in order to ensure feasibility, taking place in lines (7)-(24) of Algorithm 2. For each node $u_i \in U$, the arrival time a_{u_i} is determined using the service start time of, service time of, and travel time from its predecessor node. If its arrival time is before its earliest available time, waiting time w_{u_i} and service start time T_{u_i} are updated accordingly. If its arrival time is within the time window, only the service start time has to be updated, assuming that waiting times are initialised to zero. Lastly, if its arrival is later than its latest available time window, the solution has to be repaired. ΔT denotes the needed reduction in service time in order to make the solution feasible again. Then, the repair method iteratively visits nodes preceding the current one in a backward fashion, maximally reducing the service time at those nodes. After the service time at a node is adjusted, the `UpdateTimes` method updates the waiting and service start times of nodes affected by the change. This is necessary to calculate ΔT again afterwards, as a modification in service times at preceding nodes may not translate entirely to the arrival time at the considered node due to the time windows. When overall service time has been reduced such that the arrival at node u_i is feasible, the repair method is terminated. Due to how the routes are constructed to be feasible in the HALNS framework when no service times are considered, the aforementioned procedure always terminates. Given that the arrival at node u_i is feasible, its service time is then updated based on the minimum of the trail solution service time z_i' and available service time $l_{u_i} - T_{u_i}$. The version of the HALNS algorithm where the SSP is solved heuristically with the MCS will be denoted by HALNS-MCS. Note that the SSP is solved exactly with `CPLEX` once the algorithm terminates in order to obtain the correct objective value of the routes.

Q. Yu et al. (2021) make use of a greedy repair method in their MCS implementation, maximally reducing service times at preceding nodes yielding the smallest difference in objective value. However, due to the presence of time windows, this approach has deliberately been put aside. To illustrate, assume that the greedy approach dictates that the service time at the node 1 needs to be reduced by ΔT in order to arrive feasibly at node i. The result of executing this decision may be that waiting time needs to be incurred at a next node, due to its arrival being too early with respect to the time window. Then, reducing the service time by ΔT at node 1 does not reduce the arrival time at node i by ΔT, and subsequently no termination is guaranteed. This greedy approach may result in an unnecessary amount of reduction in service times across all nodes in the route. The proposed repair method aims to avoid just that. It may be possible to incorporate a greedy repair method in the MCS for the TOPSTPTW, but it will require additional thought on how a service time reduction at preceding nodes affects the arrival time at the desired node. The question then still is if the potentially improved solution quality justifies the additionally needed computations and likely increased solving times.

4.3.5 Set Packing Problem

As mentioned in the general outline of the HALNS framework in Section 4.3, the SPP may be used to optimally select $|P|$ routes among all generated routes, maximising the collectively collected profit (Hammami et al., 2020). Let G denote the set of all distinct tours generated during the ALNS iterations, which entails that $V_g \neq V_h$ for each $g, h \in G$. Each tour $g \in G$ has an associated profit ϕ_g equal to the sum of all profits collected at the visited nodes. Let us define the parameter a_{ig}, which is equal to one if node $i \in V \setminus \{1\}$ is contained in tour g, zero otherwise. The decision variable θ_g is equal to one if tour g is chosen, zero otherwise. Then, the SPP is formulated as per equations (32)-(35), and may be solved using the general purpose solver CPLEX.

$$\max \sum\nolimits_{g \in G} \phi_g \theta_g \tag{32}$$

$$\text{s.t.} \sum\nolimits_{g \in G} a_{ig} \theta_g \leq 1 \qquad \forall i \in V \setminus \{1,\} \tag{33}$$

$$\sum\nolimits_{g \in G} \theta_g = |P| \tag{34}$$

$$\theta_g \in \{0, 1\} \qquad \forall g \in G \tag{35}$$

The objective function (32) maximises obtained profits over selected tours. Constraints (33) ensure that across all selected tours each of the nodes is visited at most once. Constraint (34) ensures that the exactly $|P|$ tours are selected. Lastly, constraints (35) enforce binarity of the decision variables.

This chapter has discussed the necessary information to solve the TOPSTPTW either exactly or heuristically. Section 4.1 introduces a mathematical formulation, while Section 4.2 puts forward a linearisation of its nonlinear objective function. The linearised formulation may be solved using the general purpose solver `CPLEX`. Section 4.3 introduces the HALNS framework, while its subsections describe its various components. Section 4.3.1 discusses how an initial solution for the TOPSTPTW is derived, while Section 4.3.2 introduces the various node selection strategies and removal/insertion operators which are used to modify said solution. Section 4.3.3 presents a LS procedure, which aims to reduce total travel time and increase the objective value of a solution. Section 4.3.4 discusses an exact mathematical formulation and an MCS heuristic for the SSP. Lastly, Section 4.3.5 describes a mathematical formulation for the SPP, which is used to find a set of non-overlapping tours maximising the objective value. With this information, all solution methods can be implemented in the programming language of choice. However, benchmark instances will have to be generated first, since there are none available at time of writing this.

5 Data

The only known benchmark instances for the TOPSTP come from Gunawan et al. (2018), which are generated according to the procedures of Chao et al. (1996b) and Erdoğan and Laporte (2013). As such, the TSP instances kroA100, kroB100, kroC100, kroA200 and kroB200 from TSPLIB will be transformed to TOPSTP instances according to the formulae of Erdoğan and Laporte (2013) and Gunawan et al. (2018). Subsequently, time windows are constructed for each of the nodes following Kantor and Rosenwein (1992) in order to obtain TOPSTPTW instances. Complications arising from dependencies in the data are not expected for the generated instances. Firstly, without time windows the instances are valid TOPSTP instances. Secondly, the constructed time windows may influence the nodes to be visited and the amount of time spent at the nodes. However, given the structure of the problem, constructing a route is always feasible.

The locations of the nodes in the TSP instances are randomly generated in a plane, and consist of either 100 or 200 nodes, as indicated by the names of the instances. The TOPSTP instances will contain the first $k \in \{25, 50, 75, 100\}$ nodes of the TSP instance in case of it having 100 nodes, while this is equal to the first $k \in \{125, 150, 175, 200\}$ nodes in case of it having 200 nodes. This is due to the fact that the larger instances are extensions of the smaller instances, and as such duplicate computational results are avoided (Gunawan et al., 2018). The first node of each instance is set to be the depot, and $|P| \in \{2, 3, 4\}$ tours are considered.

As discussed in Section 2, the problem may be represented on a graph $G = (V, E)$. Each node $i \in V$ has an x-coordinate X_i and y-coordinate Y_i. Travel cost t_{ij} for each edge $(i, j) \in E$ are based on the Euclidean distance between the corresponding nodes $i, j \in V, i \neq j$. In this case, distance is assumed to equal travel time, and as such no specific unit of measurement will be provided for it.

r_i	$=$	$10 + (X_i + Y_i) \bmod 90$
β_i	$=$	$(10 + (((X_i + Y_i) \bmod 20)/20) \cdot 90)/20000$
T_{max}	$=$	$\lfloor 2.5 \cdot (X_{max} - X_{min} + Y_{max} - Y_{min})/\lvert P\rvert \rfloor$
e_i	$\in$	$[0, T_{max}/2]$
l_i	$\in$	$[e_i, \min(e_i + 2T_{max}/3, T_{max})]$

Table 1: Summary of TSP-to-TOPSTPTW Transformation Formulae

In determining the values of the relevant parameters, we'll first introduce $X_{max}, X_{min}, Y_{max}, Y_{min}$ as the maximum and minimum x- and y-coordinates of all nodes in an instance. Then, the formulae for transforming TSP instances to TOPSTPTW instances are summarised in Table 1 (Erdoğan and Laporte, 2013; Gunawan et al., 2018; Kantor and Rosenwein, 1992). For each node $i \in V$, r_i is contained on the interval $[10, 100)$, and for β_i this is $[0.0005, 0.005)$. The time window parameters e_i and l_i are determined using a uniformly distributed random variable on the intervals shown in Table 1 (Kantor and Rosenwein, 1992). Using the aforementioned instance characteristics, a total of 80 distinct instances are generated and available for this research. The value of T_{max} depends on the characteristics of the instance, but is contained on the interval [3358,7401] for all instances.

Additionally, small-scale TOPSTPTW instances are created from TSP instances kroA100 and kroB100, with $|P| \in \{1, 2, 3\}$ and containing the first $k \in \{5, 10, 15\}$ nodes. In case $|P| = 1$, $T_{max} \in \{5000, 7500, 10000, 12500\}$, else $T_{max} \in \{5000, 7000, 9000\}$ (Gunawan et al., 2018). These 60 instances are small enough for a general purpose solver such that a general purpose solver can solve them to optimality, and as such provide the opportunity to verify obtained results of the proposed methods with optimal solutions. All generated benchmark instances are available on GitHub.

6 Results

All computations have been performed on a workstation equipped with an Intel i5-10600K @4.1-4.8GHz and 32GB of RAM memory, running the Windows 10 operating system, and all code implemented in `Java` along with the use of the `CPLEX 12.10.0` library. For ease of notation, let us refer to an instance where $|P| = p$, $|V| = v$, and $T_{max} = t$ generated from TSP instance X by X-p-v-t. As an example, the instance generated from the TSP instance kroB100 with $|P| = 2$, $|V| = 10$, and $T_{max} = 9000$ will be referred to as kroB100-2-10-9000

6.1 Parameter Settings

Setting the HALNS parameters required some thought and tweaking, and the chosen values are shown in Table 2. The value of α remains unchanged compared to that of Hammami et al. (2020), while the β parameter was not even present in their work. The reasoning behind always having a higher value for β than α comes from the fact that node insertion only occurs when it is feasible. Given the additional difficulty imposed by time windows, this choice ensures that route size does not continually decrease over the iterations due to the inability to insert selected nodes. Note that V^+ and V^- denote the set of used and unused nodes, respectively. This terminology is extended in the case of α^p, where V_p^+ denotes the set of used nodes in a given route p.

Parameter	Description	Value
Q	Maximum number of run segments	20
N_{max}	Maximum number of iterations per run segment	20
$N_{best_{max}}$	Maximum number of iterations without improvement	100
α	Random number of nodes to remove from current solution	$\alpha \in [1, \lfloor 0.25\|V^+\| \rfloor]$
β	Random number of nodes to add to current solution	$\beta \in [\alpha, \alpha + \lfloor 0.25\|V^-\| \rfloor]$
S_0	SA initial temperature	95
S_{min}	SA minimum temperature	0.1
c	SA cooling rate	0.9
ρ_1	Operator score increment case 1	20
ρ_2	Operator score increment case 2	5
ρ_3	Operator score increment case 3	1
λ	Reaction factor adaptive weighting	0.85
α^p	Random size of sequence to remove	$\alpha^p \in [\lfloor 0.4\|V_p^+\| \rfloor, \lfloor 0.6\|V_p^+\| \rfloor]$
ϵ	Step size	5
ϵ_{min}	Minimum step size	1
$\|\|e_i\|\|$	Neighbour distance	0.1

Table 2: HALNS and MCS Parameter Values

The HALNS loop parameters Q, N_{max}, and $N_{best_{max}}$ appear relatively low, yet a choice had to be made between solution quality and solving times. The chosen values appear to achieve satisfactory results within reasonable computational time. The increase in solution quality obtained by increasing the values of these loop parameters did not appear to justify the increase in solving times. Additionally, due to the choice of these loop parameters, the SA parameters needed to be adjusted as well compared to Hammami et al. (2020), in order for the SPP to be executed at all. Regarding the MCS parameters, the value of ϵ is halved compared to Q. Yu et al. (2021), and $\|e_i\|$ multiplied by a factor of 10. These adjusted values improved computational times in the case of the TOPSTPTW, while having minimal effect on the solution quality.

6.2 Performance Evaluation

In order to evaluate the performance of the proposed methods against optimal solutions, the small-scale instances with up to 15 nodes and up to 3 routes are solved using all methods, as shown in Table 3 and 4. Table 5 summarises some performance metrics over all considered instances. `CPLEX` is able to obtain optimal solutions for all instances within 1,800 seconds. In some individual cases, `CPLEX` is quicker in solving the instance, but on average HALNS-E is the faster method. This is evident when looking at the average solving time over all instances, which is 87.88 seconds for `CPLEX` and 20.94 seconds for HALNS-E.

				`CPLEX`		HALNS-E			HALNS-MCS		
Instance	$\lvert P\rvert$	$\lvert V\rvert$	T_{max}	Obj. Val.	Time (s)	Obj. Val.	Time (s)	Gap %	Obj. Val.	Time (s)	Gap %
kroA100	1	5	5,000	53.57	0.05	53.57	1.43	0.00	53.57	0.02	0.00
	1	5	7,500	140.88	0.26	140.88	3.12	0.00	98.74	0.05	29.91
	1	5	10,000	136.24	0.14	136.24	5.06	0.00	135.06	0.07	0.87
	1	5	12,500	223.10	0.42	194.98	18.94	12.60	81.27	0.08	63.57
	1	10	5,000	162.95	0.55	143.99	14.72	11.64	145.22	0.08	10.88
	1	10	7,500	244.37	3.95	225.70	22.55	7.64	216.87	0.10	11.25
	1	10	10,000	269.39	6.21	258.09	25.34	4.19	217.50	18.84	19.26
	1	10	12,500	326.28	19.75	299.22	18.85	8.29	256.13	0.36	21.50
	1	15	5,000	189.68	5.77	183.13	6.97	3.45	117.11	0.08	38.26
	1	15	7,500	238.11	7.62	238.11	14.25	0.00	136.46	0.04	42.69
	1	15	10,000	319.53	89.19	274.85	16.30	13.98	220.57	12.67	30.97
	1	15	12,500	385.72	166.80	356.76	32.03	7.51	297.14	594.27	22.97
	2	5	5,000	93.24	0.13	93.24	5.20	0.00	88.51	0.04	5.07
	2	5	7,000	233.84	1.22	233.84	10.11	0.00	167.80	0.09	28.24
	2	5	9,000	240.60	0.69	240.60	10.03	0.00	208.35	0.08	13.40
	2	10	5,000	307.37	2.39	300.38	11.87	2.27	175.24	0.10	42.99
	2	10	7,000	387.21	17.89	383.31	26.20	1.01	232.68	5.94	39.91
	2	10	9,000	393.95	36.20	389.94	34.03	1.02	312.80	80.93	20.61
	2	15	5,000	317.94	29.35	317.94	35.86	0.00	245.72	0.30	22.72
	2	15	7,000	323.51	1.96	323.51	32.16	0.00	239.83	0.06	25.86
	2	15	9,000	584.62	1,400.94	578.61	42.77	1.03	429.12	10.05	26.60
	3	5	5,000	197.06	1.44	197.06	8.16	0.00	164.55	0.10	16.49
	3	5	7,000	210.11	2.04	210.11	9.25	0.00	200.14	0.12	4.75
	3	5	9,000	236.41	2.54	236.41	38.56	0.00	193.27	0.14	18.25
	3	10	5,000	339.74	29.86	339.74	17.00	0.00	312.99	0.11	7.87
	3	10	7,000	382.12	21.18	382.12	16.23	0.00	317.24	0.20	16.98
	3	10	9,000	442.42	90.01	442.42	43.39	0.00	368.50	0.68	16.71
	3	15	5,000	455.57	77.62	455.57	16.98	0.00	299.65	0.66	34.23
	3	15	7,000	501.62	132.52	501.62	35.09	0.00	479.03	0.47	4.50
	3	15	9,000	610.69	545.47	577.44	61.96	5.44	415.33	0.48	32.00
Mean				298.26	89.80	295.45	21.14	2.67	227.55	24.24	22.31

Note: Best HALNS-MCS result over 10 runs reported. Overall time limit of 1,800 seconds imposed.

Table 3: Performance Evaluation Results for kroA100

This is largely due to CPLEX struggling with some specific instances, such as kroA100-2-15-9000, showing significantly increased solving times. For those instances however, HALNS-E is able to find a near-optimal solution in a fraction of the computational time, and in addition shows a maximum computational time of 107.48 seconds. Overall, HALNS-E is able to find the optimal solution for 53.3% of the instances, and shows an average optimality gap of 6.43% for the remaining instances. The maximum optimality gap obtained by HALNS-E for these instances is 15.54% for instance kroB100-1-5-10000, which can be acceptable, yet still being large.

				CPLEX		HALNS-E			HALNS-MCS		
Instance	$\|P\|$	$\|V\|$	T_{max}	Obj. Val.	Time (s)	Obj. Val.	Time (s)	Gap %	Obj. Val.	Time (s)	Gap %
kroB100	1	5	5,000	89.96	0.20	89.96	3.40	0.00	76.89	0.04	14.52
	1	5	7,500	211.26	0.42	211.26	4.97	0.00	160.29	0.05	24.12
	1	5	10,000	163.52	0.25	163.52	7.09	0.00	115.15	0.15	29.58
	1	5	12,500	173.30	0.43	157.65	7.78	9.03	155.93	0.06	10.02
	1	10	5,000	176.11	0.49	149.76	14.33	14.96	136.24	0.06	22.64
	1	10	7,500	222.34	2.98	215.57	12.08	3.04	190.53	0.07	14.31
	1	10	10,000	352.14	3.52	297.40	15.76	15.54	241.44	0.46	31.44
	1	10	12,500	404.30	12.38	404.30	36.21	0.00	317.96	200.44	21.35
	1	15	5,000	226.71	3.13	226.71	8.96	0.00	162.58	0.06	28.29
	1	15	7,500	270.93	6.49	247.38	24.00	8.69	217.38	0.05	19.77
	1	15	10,000	381.47	145.26	360.11	24.51	5.60	263.09	1.62	31.03
	1	15	12,500	418.54	230.27	416.49	40.14	0.49	250.23	1,919.17	40.21
	2	5	5,000	128.64	0.09	128.64	3.41	0.00	96.45	0.05	25.02
	2	5	7,000	236.80	0.77	236.80	10.36	0.00	236.80	0.10	0.00
	2	5	9,000	242.95	0.76	242.95	17.68	0.00	173.93	0.10	28.41
	2	10	5,000	325.07	4.19	321.44	13.43	1.12	249.18	0.28	23.35
	2	10	7,000	434.80	20.40	370.14	11.78	14.87	334.61	8.03	23.04
	2	10	9,000	412.98	15.22	358.71	14.94	13.14	362.15	625.20	12.31
	2	15	5,000	250.57	33.51	250.57	12.00	0.00	196.12	0.37	21.73
	2	15	7,000	246.94	1.59	246.94	8.94	0.00	223.13	0.08	9.64
	2	15	9,000	612.08	774.50	595.53	34.38	2.70	480.10	2.69	21.57
	3	5	5,000	151.38	0.78	151.38	6.77	0.00	150.32	0.13	0.70
	3	5	7,000	263.71	3.09	258.27	21.94	2.06	243.61	149.34	7.63
	3	5	9,000	248.52	1.73	248.52	22.94	0.00	234.81	1.80	5.52
	3	10	5,000	393.46	27.80	393.46	16.57	0.00	370.90	2.53	5.74
	3	10	7,000	498.47	41.47	464.84	30.62	6.75	468.41	52.62	6.03
	3	10	9,000	532.37	128.19	530.38	107.48	0.37	532.37	454.70	0.00
	3	15	5,000	325.96	38.69	325.96	20.27	0.00	316.72	0.56	2.84
	3	15	7,000	599.91	366.42	599.91	31.63	0.00	538.01	0.21	10.32
	3	15	9,000	581.44	713.62	572.58	37.42	1.52	520.39	1,809.95	10.51
Mean				319.22	85.95	313.53	20.73	3.33	267.79	174.36	16.56

Note: Best HALNS-MCS result over 10 runs reported. Overall time limit of 1,800 seconds imposed.

Table 4: Performance Evaluation Results for kroB100

Noteworthy is that HALNS-MCS has some unexpected results for these instances. Due to the SSP being solved heuristically in this method, it was expected that the solution quality would degrade to some extent. This is apparent in the fact that only 5% of the instances can be solved to optimality, with an average optimality gap of 20.54% for the remaining instances. Moreover, the maximum optimality gap is reported as 63.57% for instance kroA100-1-5-12500, which is unfortunately significantly larger than that of HALNS-E. What is unexpected is that HALNS-MCS on average takes more computational time than the other methods to solve the instances, namely 99.30 seconds. This discrepancy is mainly due to a minority of the instances taking more than 400 seconds to solve, or even reaching the imposed overall time limit of 1,800 seconds in the case of the instances kroB100-1-15-12500 and kroB100-3-15-9000. As will be shown in Section 6.3.2, such outliers do not appear in the larger instances. Subsequently, it seems that the MCS method is struggling when it has a small amount of available nodes to choose from.

Given that averages may give a distorted view due to the presence of outliers, Table 5 shows some additional metrics over all of the small-scale instances. The median solving time for HALNS-E is 16.43 seconds, which is fairly close to the average solving time of 20.94 seconds. `CPLEX` and HALNS-MCS show very different results. The median solving time of HALNS-MCS is 0.18 seconds, while that of `CPLEX` is 4.98. The respective average solving times are 99.30 and 87.88 seconds. Subsequently, when looking at these numbers, several observations can be made. HALNS-E is the most reliable method of the three, as the solving times do not fluctuate severely across the instances. HALNS-MCS does not appear to be performing that well according to the average solving time, but its median solving time is significantly lower than the other methods. Additionally, it attains the lowest solving time across all three methods for 81.67% of the instances. `CPLEX`' median solving time is lower than that of HALNS-E, but the significantly higher average solving time indicates a notably higher degree of fluctuations in solving times.

	CPLEX	HALNS-E	HALNS-MCS
Average Solving Time	87.88	20.94	99.30
Median Solving Time	4.98	16.43	0.18
% Lowest Solving Time	6.67	11.67	81.67
% Optimal Solution	100.00	53.33	5.00

Table 5: Summarised Performance Evaluation Results for kroA100 and kroB100

Figure 1 and 2 visualise the optimal solution obtained for the instance kroA-2-10-5000. In each of the figures the horizontal axis shows the time stamps of events occuring in the route, with the accompanying text describing the event. The time windows for each of the visited nodes can be seen in the upper left corner of each of the figures. The figures depict that the service start times and leaving times of a solution respect the time windows, and Figure 2 shows us that waiting time is incurred when arriving too early at a node.

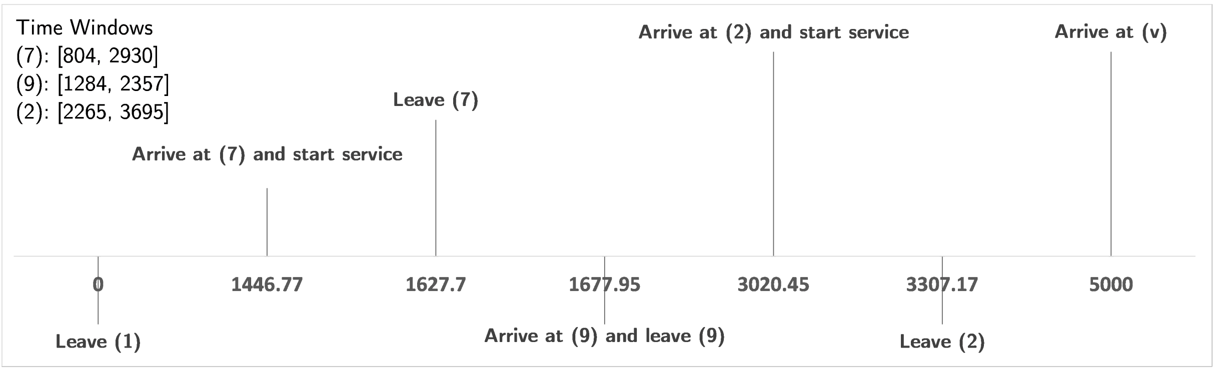

Figure 1: Route 1 of Optimal Solution for kroA-2-10-5000

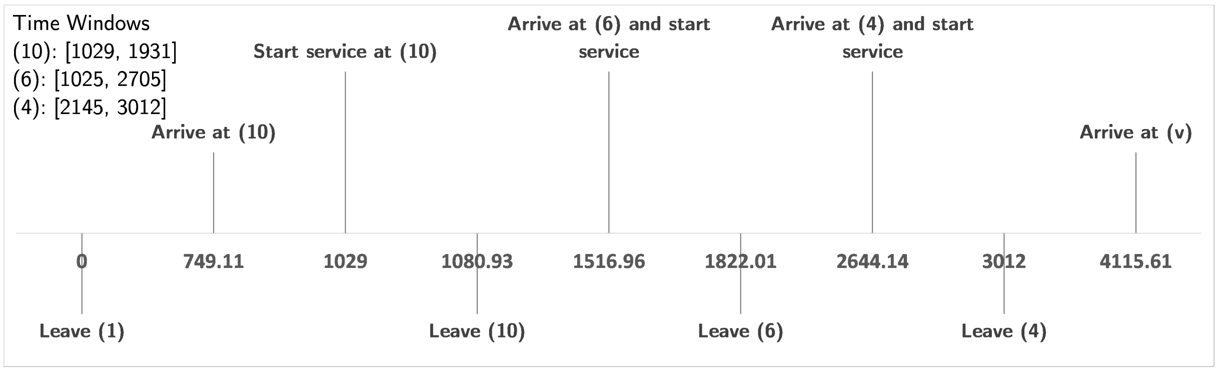

Figure 2: Route 2 of Optimal Solution for kroA-2-10-5000

6.3 Computational Results

The nature of the HALNS framework is inherently probabilistic, among others due to its use of random numbers in order to determine which selection strategy or operator is used. Due to this characteristic, solutions may vary per execution of the algorithm, not excluding the assumption that a different seed may be used for the random number generation per run. These facts led to the decision to execute each of the HALNS methods ten times in order to get their average performance, while additionally being able to extract the best possible solution among these separate runs. Subsequently, Table 10 and Table 11 in Appendix A show the average computational results for the considered instances.

6.3.1 Exact Approach

As a baseline, all instances are solved with `CPLEX` under a time limit of 1,800 seconds, such that an upper bound on the objective value and a best feasible solution is obtained. These results are shown in Table 6, and showcase the computational difficulty of solving the TOPSTPTW in an exact manner. Still, for some of the smaller instances, namely kroA100-4-25-3553, kroB100-4-25-3358 and kroC100-4-25-3510, an optimal solution has been found. But the optimality gap of the best feasible solution with respect to the obtained upper bound is 781.15% and 2,630.99% on average for instance sets with up 100 and 200 nodes, respectively. Looking at maximum optimality gaps, this becomes 4,015.93% and 8,458.66% for the respective sets of instances. The HALNS methods are nevertheless compared to the obtained upper bounds, but that should be done with a careful eye.

The upper bound on the objective value may be rather weak for increased instance sizes, as may be deduced from the aforementioned optimality gaps for the best found feasible solution. Hence, the optimality gap with respect to the upper bound obtained by `CPLEX` can be seen as itself an upper bound on the optimality gap. A stronger upper bound for the TOPSTPTW may be obtained by increasing the imposed time limit of `CPLEX`, or possibly more sophisticated methods may be developed specifically for the TOPSTPTW. That is however not the main focus of this research. Besides the optimality gap, the objective value of the HALNS methods will be compared to the best feasible solution found by `CPLEX` in order to show how much of an improved solution these methods can obtain in less than 1,800 seconds.

Instance	$\|P\|$	$\|V\|$	T_{max}	UB	Best Obj.	Instance	$\|P\|$	$\|V\|$	T_{max}	UB	Best Obj.
kroA100	2	25	7,106	725.60	690.45	kroA200	2	125	7,345	3,221.13	298.07
	2	50	7,308	1,414.97	214.94		2	150	7,380	3,221.13	298.07
	2	75	7,351	1,933.05	248.67		2	175	7,380	4,197.46	69.92
	2	100	7,351	2,450.11	85.73		2	200	7,380	4,715.54	107.58
	3	25	4,737	686.55	617.60		3	125	4,896	3,177.23	0.00
	3	50	4,872	1,341.53	206.30		3	150	4,920	3,808.46	245.85
	3	75	4,900	1,948.51	358.09		3	175	4,920	4,039.41	47.20
	3	100	4,900	2,652.81	0.00		3	200	4,920	4,662.67	291.43
	4	25	3,553	379.29	379.29		4	125	3,672	3,001.11	116.84
	4	50	3,654	1,393.09	399.67		4	150	3,690	3,603.73	309.26
	4	75	3,675	1,738.52	280.38		4	175	3,690	4,173.45	228.61
	4	100	3,675	2,303.64	58.84		4	200	3,690	4,593.34	259.42
kroB100	2	25	6,717	724.61	497.65	kroB200	2	125	7,391	3,149.58	276.98
	2	50	7,051	1,551.88	261.21		2	150	7,391	3,794.03	227.48
	2	75	7,275	2,058.20	0.00		2	175	7,391	4,406.26	94.00
	2	100	7,391	2,710.89	341.40		2	200	7,401	5,087.07	219.86
	3	25	4,478	753.42	407.07		3	125	4,927	3,182.38	250.28
	3	50	4,700	1,295.45	260.40		3	150	4,927	3,680.48	93.79
	3	75	4,850	1,871.65	0.00		3	175	4,927	4,290.78	326.77
	3	100	4,927	2,735.77	125.60		3	200	4,934	4,906.98	327.39
	4	25	3,358	343.65	343.65		4	125	3,695	3,031.29	93.48
	4	50	3,525	1,138.21	429.98		4	150	3,695	3,341.38	91.55
	4	75	3,637	1,731.33	333.10		4	175	3,695	3,822.71	208.43
	4	100	3,695	2,794.81	141.06		4	200	3,700	4,687.13	154.41
kroC100	2	25	7,020	847.38	554.26						
	2	50	7,186	1,345.26	440.07						
	2	75	7,240	2,087.82	137.88						
	2	100	7,345	2,798.28	67.99						
	3	25	4,680	674.22	499.33						
	3	50	4,790	1,336.02	0.00						
	3	75	4,826	2,021.24	334.77						
	3	100	4,896	2,504.59	356.60						
	4	25	3,510	341.11	341.11						
	4	50	3,593	1,136.71	304.19						
	4	75	3,620	1,869.49	244.57						
	4	100	3,672	2,365.71	159.77						

Note: UB and Best Obj. denote the best upper bound and the objective value of the best solution found by `CPLEX` at 1,800s, respectively.

Table 6: Computational Results `CPLEX` for kroA100, kroB100, kroC100, kroA200 and kroB200

6.3.2 Heuristic Approach

Table 7 shows the detailed computational results for the instances with up to 100 nodes. HALNS-E shows strong results, always finding an improved solution compared to `CPLEX`, save for one instance. For that one instance kroA100-2-25-7106, the reported optimality gap is just 5.39%. Nevertheless, the average percentual improvement over the best feasible solution found by `CPLEX` is 391.25% with an average computational time of 80.16 seconds. More interestingly, HALNS-E is able to obtain the optimal solution for four instances. `CPLEX` was also able to find the optimal solution for three of these instance, but HALNS-E was quicker in each case. HALNS-E took 38.67, 20.04, and 35.94 seconds for the respective instances kroA100-4-25-3553, kroB100-4-25-3358 and kroC100-4-25-3510, while `CPLEX` took 960.83, 50.24, and 75.86 seconds. Moreover, the maximum percentual improvement is equal to 2,073.60%, excluding those instances for which the objective value of the best feasible solution is zero.

				HALNS-E				HALNS-MCS			
Instance	$\|P\|$	$\|V\|$	T_{max}	Obj. Val.	Time (s)	Gap %	Improve %	Obj. Val.	Time (s)	Gap %	Improve %
kroA100	2	25	7,106	686.48	36.84	5.39	0.00	423.22	9.92	41.67	0.00
	2	50	7,308	884.42	87.69	37.50	311.48	484.39	14.28	65.77	125.36
	2	75	7,351	1,273.80	151.78	34.10	412.25	574.74	13.03	70.27	131.13
	2	100	7,351	1,136.91	146.77	53.60	1,226.23	452.96	10.33	81.51	428.38
	3	25	4,737	686.55	63.44	0.00	11.17	441.26	0.27	35.73	0.00
	3	50	4,872	861.64	63.96	35.77	317.67	473.56	1.58	64.70	129.55
	3	75	4,900	986.23	83.26	49.39	175.42	525.93	0.63	73.01	46.87
	3	100	4,900	1,338.06	122.87	49.56	>1,000.00	552.17	2.19	79.19	>1,000.00
	4	25	3,553	379.29	38.67	0.00	0.00	293.27	0.12	22.69	0.00
	4	50	3,654	677.16	41.21	51.39	69.43	427.25	1.09	69.33	6.90
	4	75	3,675	921.38	71.88	47.00	228.62	468.57	0.17	73.05	67.12
	4	100	3,675	1,279.03	92.69	44.48	2,073.60	554.76	0.29	75.92	842.77
kroB100	2	25	6,717	499.91	42.27	31.01	0.45	306.09	1.65	57.76	0.00
	2	50	7,051	838.97	98.12	45.94	221.19	540.70	2.19	65.16	107.00
	2	75	7,275	1,267.23	150.41	38.43	>1,000.00	521.81	14.98	74.65	>1,000.00
	2	100	7,391	1,370.01	127.27	49.46	301.29	549.69	2.35	79.72	61.01
	3	25	4,478	537.26	28.92	28.69	31.98	395.20	0.17	47.55	0.00
	3	50	4,700	758.30	96.81	41.46	191.21	542.49	0.97	58.12	108.33
	3	75	4,850	1,189.39	102.01	36.45	>1,000.00	558.25	0.71	70.17	>1,000.00
	3	100	4,927	1,488.61	69.77	45.59	1,085.17	781.89	1.21	71.42	522.51
	4	25	3,358	343.65	20.04	0.00	0.00	343.65	0.12	0.00	0.00
	4	50	3,525	798.84	56.90	29.82	85.79	442.46	0.27	61.13	2.90
	4	75	3,637	986.93	61.10	43.00	196.28	598.12	0.44	65.45	79.56
	4	100	3,695	1,368.08	108.49	51.05	869.89	746.10	0.36	73.30	428.94
kroC100	2	25	7,020	658.86	63.87	22.25	18.87	375.33	0.77	55.71	0.00
	2	50	7,186	853.97	98.14	36.52	94.05	546.07	1.31	59.41	24.09
	2	75	7,240	1,231.88	122.75	41.00	793.45	641.47	13.36	69.28	365.24
	2	100	7,345	1,440.33	156.23	48.53	2,018.55	549.39	2.25	80.37	708.08
	3	25	4,680	536.16	39.26	20.48	7.38	427.00	0.16	36.67	0.00
	3	50	4,790	896.91	54.90	32.87	>1,000.00	632.21	3.11	52.68	>1,000.00
	3	75	4,826	1,206.73	70.65	40.30	260.47	546.17	1.41	72.98	63.15
	3	100	4,896	1,290.64	120.24	48.47	261.93	658.20	2.54	73.72	84.58
	4	25	3,510	341.11	35.94	0.00	0.00	341.11	0.14	0.00	0.00
	4	50	3,593	729.51	55.52	35.82	139.82	510.58	0.18	55.08	67.85
	4	75	3,620	960.43	62.27	48.63	292.70	603.05	0.78	67.74	146.57
	4	100	3,672	1,240.01	42.79	47.58	676.11	641.75	0.30	72.87	304.67
Mean				957.63	80.16	39.73	391.25	516.34	2.93	63.93	152.62

Note: Gaps are given with respect to the upper bound found by `CPLEX` after 1,800 seconds; Improvement is measured with respect to the best feasible solution obtained by `CPLEX` after 1,800 seconds; Best HALNS results over 10 runs reported.

Table 7: Computational Results HALNS for kroA100, kroB100, and kroC100

An average optimality gap of 39.73% is obtained over all instances, with a maximum value of 53.60%. Remember that the value of the reported optimality gaps are likely to be overstated due to the potentially weak upper bound. Moreover, a solution is obtained within 3 minutes for all instances, with a maximum computational time of 156.23 seconds.

Turning our attention to HALNS-MCS, the direction of the results is what was expected. That is, computational times have improved and solution quality has decreased compared to HALNS-E. A solution is obtained for all instances within one minute, with an average and maximum computational time of 2.93 and 14.98 seconds, respectively. HALNS-MCS was able to obtain the optimal solution for two of the three instances that HALNS-E was able to solve optimally. On the other hand, HALNS-MCS was not able to find an improved solution for a total of seven instances. The average and maximum percentual improvement over all instances is 152.62% and 842.77%, respectively, which constitutes a substantial deterioration in solution quality compared to HALNS-E. This trend continues when looking at the reported optimality gaps for HALNS-MCS, which on average and at maximum are 63.93% and 81.51%, respectively.

				HALNS-E				HALNS-MCS			
Instance	$\lvert P\rvert$	$\lvert V\rvert$	T_{max}	Obj. Val.	Time (s)	Gap %	Improve %	Obj. Val.	Time (s)	Gap %	Improve %
kroA200	2	125	7,345	1,538.47	214.11	52.24	416.15	552.67	1.68	82.84	85.42
	2	150	7,380	1,561.13	206.52	55.95	>1,000.00	602.62	0.96	83.00	>1,000.00
	2	175	7,380	1,604.68	214.85	61.77	2,195.18	451.37	0.49	89.25	545.60
	2	200	7,380	1,723.80	183.42	63.44	1,502.32	662.06	22.57	85.96	515.41
	3	125	4,896	1,582.27	113.42	50.20	>1,000.00	728.40	0.57	77.07	>1,000.00
	3	150	4,920	1,648.27	125.85	56.72	570.45	670.56	0.70	82.39	172.75
	3	175	4,920	1,723.38	138.43	57.34	3,551.49	628.95	0.55	84.43	1,232.60
	3	200	4,920	1,979.96	151.97	57.54	579.39	601.97	0.60	87.09	106.56
	4	125	3,672	1,330.73	54.91	55.66	1,038.97	585.44	0.22	80.49	401.08
	4	150	3,690	1,456.67	109.46	59.58	371.01	690.46	1.54	80.84	123.26
	4	175	3,690	1,655.74	73.31	60.33	624.28	703.66	1.62	83.14	207.81
	4	200	3,690	1,700.25	114.99	62.98	555.42	728.96	1.04	84.13	181.00
kroB200	2	125	7,391	1,716.39	192.29	45.50	519.69	738.92	2.53	76.54	166.78
	2	150	7,391	1,810.46	203.49	52.28	695.87	610.85	2.22	83.90	168.53
	2	175	7,391	1,950.13	234.72	55.74	1,974.66	771.32	10.69	82.49	720.58
	2	200	7,401	1,953.40	265.51	61.60	788.49	627.82	2.02	87.66	185.56
	3	125	4,927	1,639.82	88.07	48.47	555.19	824.50	1.78	74.09	229.43
	3	150	4,927	1,788.89	125.58	51.40	1,807.34	698.81	1.31	81.01	645.08
	3	175	4,927	1,759.50	110.39	58.99	438.45	705.60	2.19	83.56	115.93
	3	200	4,934	1,922.39	137.95	60.82	487.19	650.29	1.33	86.75	98.63
	4	125	3,695	1,379.85	92.16	54.48	1,376.11	706.78	0.50	76.68	656.10
	4	150	3,695	1,579.96	96.10	52.72	1,625.83	737.07	1.94	77.94	705.12
	4	175	3,695	1,776.32	88.94	53.53	752.26	740.38	0.96	80.63	255.23
	4	200	3,700	2,066.97	112.87	55.90	1,238.63	791.24	1.06	83.12	412.43
Mean				1,702.06	143.72	56.05	1,075.65	675.45	2.54	82.29	360.49

Note: Gaps are given with respect to the upper bound found by `CPLEX` after 1,800 seconds; Improvement is measured with respect to the best feasible solution obtained by `CPLEX` after 1,800 seconds; Best HALNS results over 10 runs reported.

Table 8: Computational Results HALNS for kroA200 and kroB200

Table 8 shows the detailed computational results for the instances with up to 200 nodes. For these instances, none of the HALNS methods is able to find an optimal solution, yet they both are able to find an improved solution for all instances compared to `CPLEX` within considerably less time. Note that `CPLEX` could not find any optimal solution within 1,800 seconds for these larger instances. HALNS-E takes on average 143.72 seconds to find a solution, with a maximum computational time of 265.51 seconds. HALNS-MCS on the other hand does not show significantly increased solving times on average compared to the instances with up to 100 nodes, with an average and maximum solving time of 2.54 and 22.57 seconds.

With respect to solution quality, the results become more difficult to interpret. This is due to the fact that almost none of the objective values of the best feasible solutions found by `CPLEX` exceed 300, and all obtained upper bounds exceed 3,000. Subsequently, virtually any solution found by both HALNS methods will be an improved solution, which is positive in its own right. However, the optimality gaps for these instances become increasingly ambiguous, due to the increased likelihood of them becoming increasingly weak for larger instance sizes.

As before, HALNS-E outperforms HALNS-MCS by a large margin in terms of solution quality, although it comes at the cost of significantly larger computational times. The average and maximum optimality gap for HALNS-E is equal to 56.05% and 63.44%, respectively, while these values are 82.29% and 89.25%

for HALNS-MCS. These results are apparent when realising that virtually all objective values obtained by HALNS-E are a factor of at least two larger than those obtained by HALNS-MCS. Similarly, the average and maximum improvement over `CPLEX` is 1,075.65% and 3,551.49% for HALNS-E, respectively, while these values are 360.49% and 1,232.60% for HALNS-MCS.

6.4 Algorithm Analysis

In order to the evaluate the performance of the algorithm and its components, results have been obtained for HALNS-E with certain components left out or disabled. Firstly, the component of adaptive weighting is disabled such that each of the node selection strategies and removal/insertion operators have an equal chance to be selected during the entire solving process. Secondly, the local search component is left out to see whether its addition has a significant effect on the algorithm's outcome. Hammami et al. (2020) used a similar strategy to evaluate their HALNS algorithm. For the analysis of this implementation of the HALNS algorithm, adaptive weighting and local search where thought to show the greatest impact if left out or disabled. Additionally, like Hammami et al. (2020), differing values of the reaction factor λ or leaving out the SPP could be considered. However, these options have been disregarded due to time considerations. The average results for HALNS-E and these derivative implementations for the benchmark instances with up to 100 nodes are shown in Table 9.

	Avg. Objective Value	Avg. Solving Time (s)	Avg. Gap %	Avg. Improve %
HALNS-E	957.63	80.16	39.73	391.25
HALNS-E w/o Adaptive Weighting	938.73	75.62	39.49	390.08
HALNS-E w/o Local Search	956.70	76.79	38.64	381.61

Note: Averages taken of best HALNS results over 10 runs.

Table 9: Average Results HALNS-E and derivative implementations for kroA100, kroB100, and kroC100

The average solving time of HALNS-E is greater than the derivative implementations by approximately four seconds. Given that extra computations have to be performed in the complete algorithm, this is in line with expectations. The effect of adaptive weighting on the average objective value is larger than that of the local search component, showing a decrease of 18.90 and 0.93 compared to HALNS-E, respectively. The average percentual improvement compared to the best feasible solution found by `CPLEX` is favourable for HALNS-E, being 1.17 higher when adaptive weighting is disabled and 9.64 higher when the local search component is left out. The average optimality gaps show a slightly different picture on the other hand, with HALNS-E having the highest value among the considered implementations. Given that the computed optimality gaps are upper bounds, the results are favourable for the complete HALNS-E algorithm, with it having the highest average objective value and percentual improvement compared to the derivative implementations.

7 Conclusion & Discussion

The purpose of this research is to study the Team Orienteering Problem with Service-Time-Dependent Profits and Time Windows. To that extent, a Mixed Integer Nonlinear Programming formulation is developed, where a concave increasing objective function is maximised. As a baseline, the proposed formulation is linearised and solved using the general purpose solver `CPLEX` under a time limit of 1,800 seconds. The main focus of this research is a Hybrid Adaptive Large Neighbourhood Search heuristic to solve the TOPSTPTW, which consists of routing and scheduling subproblems. Routes are constructed and modified within the HALNS framework, while the scheduling of service times takes places each time a route is modified. The Service-Time Scheduling Problem is the most demanding part of the algorithm, and as such an exact and heuristic approach is proposed. The exact approach HALNS-E follows a linearised MINLP formulation for the SSP, which is solved using `CPLEX`, while the heuristic approach HALNS-MCS follows a Modified Coordinate Search method.

HALNS-E significantly outperforms `CPLEX` for the considered instances with more than 25 nodes, always finding an improved solution within four minutes. Whether these solutions are in actuality close to optimal is difficult to determine, due to the likelihood of the obtained upper bounds being rather weak. Still, the percentual improvements over the best feasible solution found by `CPLEX` are often quite substantial. HALNS-MCS is a quick method to solve the TOPSTPTW, with no computational time exceeding 30 seconds for instances with 25 or more nodes. The solution quality of this method however does leave room for improvement, showing significantly lower objective values compared to HALNS-E. For smaller instances with up to 15 nodes the results are more ambiguous to interpret, where occasionally `CPLEX` is quicker or obtains better solutions than HALNS-E. More interestingly, the behaviour of HALNS-MCS becomes unpredictable for these instances, at times showing unusually large solving times of over 400 seconds.

HALNS-E is a reliable method to solve the TOPSTPTW, showing steady computational times and strong solutions. In general, the TOPSTPTW quickly becomes computationally difficult for increased instance sizes. As such, a general purpose solver such as `CPLEX` is not favoured for solving real-world problems. If solving times are more important than solution quality, HALNS-MCS is the preferred method. However, it should be taken into account that the solution quality will be far from optimal in this case, and instances with less than 25 nodes will possibly take an unexpected amount of time to solve.

The SSP assumes that the order of the nodes in a route is fixed, for future research on this topic an idea could be to omit this assumption from the MINLP formulation. This entails that the SSP can also change the order of the nodes in a given route if that would result in a higher objective value. This approach is computationally more intense, but could potentially result in quicker convergence of the algorithm. In this case, an accompanying MCS heuristic will not be possible, as the order of the nodes is important in determining a solution. Furthermore, the MCS heuristic as implemented in this research may be further tweaked. First of all, the repair method could be improved upon, through investigating the relationship between decreasing service times and arrival times, and potentially through the implementation of a greedy repair method. Secondly, it may be interesting to construct candidate solutions in such a fashion that it is not possible to generate infeasible trail solutions. Additionally, a simple yet intuitive idea when performing multiple runs of the algorithm would be to reuse the best found solution of a previous run as an initial solution for a new run. This results in a stronger initial solution, which may possibly improve the performance of subsequent runs. Lastly, future research on the TOPSTPTW would benefit from stronger upper bounds in order to better determine optimality gaps.

References

Afsar, H., & Labadie, N. (2013). Team Orienteering Problem with Decreasing Profits. *Electronic Notes in Discrete Mathematics*, *41*, 285–293. https://doi.org/10.1016/j.endm.2013.05.104

Archetti, C., Hertz, A., & Speranza, M. (2007). Methaheuristics for the Team Orienteering Problem. *Journal of Heuristics*, *13*, 49–76. https://doi.org/10.1007/s10732-006-9004-0

Bianchessi, N., Mansini, R., & Speranza, M. (2018). A Branch-and-Cut Algorithm for the Team Orienteering Problem. *International Transactions in Operational Research*, *25*(2), 627–635. https://doi.org/10.1111/itor.12422

Bouly, H., Dang, D.-C., & Moukrim, A. (2010). A Memetic Algorithm for the Team Orienteering Problem. *4OR*, *8*, 49–70. https://doi.org/10.1007/s10288-008-0094-4

Boussier, S., Feillet, D., & Gendreau, M. (2007). An Exact Algorithm for Team Orienteering Problems. *4OR*, *5*, 211–230. https://doi.org/10.1007/s10288-006-0009-1

Butt, S., & Cavalier, T. (1994). A Heuristic for the Multiple Tour Maximum Collection Problem. *Computers & Operations Research*, *21*(1), 101–111. https://doi.org/10.1016/0305-0548(94)90065-5

Butt, S., & Ryan, D. (1999). An Optimal Solution Procedure for the Multiple Tour Maximum Collection Problem using Column Generation. *Computers & Operations Research*, *26*(4), 427–441. https://doi.org/10.1016/S0305-0548(98)00071-9

Chao, I.-M., Golden, B., & Wasil, E. (1996a). A Fast and Effective Heuristic for the Orienteering Problem. *European Journal of Operational Research*, *88*(3), 475–489. https://doi.org/10.1016/0377-2217(95)00035-6

Chao, I.-M., Golden, B., & Wasil, E. (1996b). The Team Orienteering Problem. *European Journal of Operational Research*, *88*(3), 464–474. https://doi.org/10.1016/0377-2217(94)00289-4

Dang, D.-C., El-Hajj, R., & Moukrim, A. (2013). A Branch-and-Cut Algorithm for Solving the Team Orienteering Problem. In C. Gomes & M. Sellmann (Eds.), *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2013)*. Springer. https://doi.org/10.1007/978-3-642-38171-3_23

Dang, D.-C., Guibadj, R., & Moukrim, A. (2013a). A PSO-Based Memetic Algorithm. In C. Gomes & M. Sellmann (Eds.), *Applications of Evolutionary Computation (EvoApplications 2011)* (pp. 471–480). Springer. https://doi.org/10.1007/978-3-642-38171-3_23

Dang, D.-C., Guibadj, R., & Moukrim, A. (2013b). An Effective PSO-Inspired Algorithm for the Team Orienteering Problem. *European Journal of Operations Research*, *229*(2), 332–344. https://doi.org/10.1016/j.ejor.2013.02.049

Ekici, A., & Retharekar, A. (2013). Multiple Agents Maximum Collection Problem with Time Dependent Rewards. *Computers & Industrial Engineering*, *64*(4), 1009–1018. https://doi.org/10.1016/j.cie.2013.01.010

El-Hajj, R., Dang, D.-C., & Moukrim, A. (2016). Solving the Team Orienteering Problem with Cutting Planes. *Computers & Operations Research*, *74*, 21–30. https://doi.org/10.1016/j.cor.2016.04.008

Erdoğan, G., & Laporte, G. (2013). The Orienteering Problem with Variable Profits. *Networks*, *61*(2), 104–116. https://doi.org/10.1002/net.21496

Erkut, E., & Zhang, J. (1996). The Maximum Collection Problem with Time-Dependent Rewards. *Naval Research Logistics*, *43*(5), 749–763. https://doi.org/10.1002/(SICI)1520-6750(199608)43:5⟨749::AID-NAV10⟩3.0.CO;2-J

Fischetti, M., González, J., & Toth, P. (1998). Solving the Orienteering Problem through Branch-and-Cut. *INFORMS Journal on Computing*, *10*(2), 133–148. https://doi.org/10.1287/ijoc.10.2.133

Golden, B., Wang, Q., & Liu, L. (1988). A Multifaceted Heuristic for the Orienteering Problem. *Naval Research*, *35*(3), 359–366. https://doi.org/10.1002/1520-6750(198806)35:3⟨359::AID-NAV3220350305⟩3.0.CO;2-H

Golden, B., Levy, L., & Vohra, R. (1987). The Orienteering Problem. *Naval Reseach Logistics*, *34*(3), 307–318. https://doi.org/10.1002/1520-6750(198706)34:3⟨307::AID-NAV3220340302⟩3.0.CO;2-D

Guitouni, A., & Masri, H. (2014). An Orienteering Model for the Search and Rescue Problem. *Computational Management Science*, *11*, 459–473. https://doi.org/10.1007/s10287-013-0179-1

Gunawan, A., Lau, H., & Lu, K. (2015a). SAILS: Hybrid Algorithm for the Team Orienteering Problem with Time Windows. *Proceedings of the 7th Multidisciplinary International Scheduling Conference (MISTA 2015), Prague, Czech Replublic, August 25-28*, 276–295. Available at: https://ink.library.smu.edu.sg/sis_research/3398

Gunawan, A., Lau, H., & Lu, K. (2015b). Well-Tuned ILS for Extended Team Orienteering Problem with Time Windows. *LARC Technical Report Series*. Singapore Mangement University.

Gunawan, A., Ng, K., Kendall, G., & Lai, J. (2018). An Iterated Local Search Algorithm for the Team Orienteering Problem with Variable Profits. *Engineering Optimization*, *50*(7), 1148–1163. https://doi.org/10.1080/0305215X.2017.1417398

Hammami, F., Rekik, M., & Coelho, L. (2020). A Hybrid Adaptive Large Neighbourhood Search Heuristic for the Team Orienteering Problem. *Computers & Operations Research*, *123*, 1–18. https://doi.org/10.1016/j.cor.2020.105034

Kantor, M., & Rosenwein, M. (1992). The Orienteering Problem with Time Windows. *Journal of Operational Research Society*, *43*(6), 629–635. https://doi.org/10.1057/jors.1992.88

Ke, L., Archetti, C., & Feng, Z. (2008). Ants can Solve the Team Orienteering Problem. *Computers & Industrial Engineering*, *54*(3), 648–665. https://doi.org/10.1016/j.cie.2007.10.001

Ke, L., Zhai, L., Li, J., & Chan, F. (2016). Pareto Mimic Algorithm: An Approach to the Team Orienteering Problem. *Omega*, *61*, 155–166. https://doi.org/10.1016/j.omega.2015.08.003

Keller, J. M., Gray, M. R., & Givens, J. A. (1985). A fuzzy k-nearest neighbor algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-15*(4), 580–585. https://doi.org/10.1109/TSMC.1985.6313426

Keshtkaran, M., Ziarati, K., Bettinelli, A., & Vigo, D. (2016). Enhanced Exact Solution Methods for the Team Orienteering Problem. *International Journal of Production Research*, *54*(2), 591–601. https://doi.org/10.1080/00207543.2015.1058982

Labadie, N., Mansini, R., Melechovsky, J., & Calvo, R. (2012). The Team Orienteering Problem with Time Windows: An LP-based Granular Variable Neighborhood Search. *European Journal of Operational Research*, *220*(1), 15–27. https://doi.org/10.1016/j.ejor.2012.01.030

Laporte, G., & Martello, S. (1990). The Selective Travelling Salesman Problem. *Discrete Applied Mathematics*, *26*(2-3), 193–207. https://doi.org/10.1016/0166-218X(90)90100-Q

Lin, S.-W. (2013). Solving the Team Orienteering Problem using Effective Multi-Start Simulated Annealing. *Applied Soft Computing*, *13*(2), 1064–1073. https://doi.org/10.1016/j.asoc.2012.09.022

Miller, C., Tucker, A., & Zemlin, R. (1960). Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM*, *7*(4), 326–329. https://doi.org/10.1145/321043.321046

Pessoa, A., Sadykov, R., Uchoa, E., & Vanderbeck, F. (2019). A Generic Exact Solver for Vehicle Routing and Related Problems. In A. Lodi & V. Nagarajan (Eds.), *Integer Programming and Combinatorial Optimization (IPCO 2019)*. Springer. https://doi.org/10.1007/978-3-030-17953-3_27

Pillai, R. (1992). *The Traveling Salesman Subset-Tour Problem with One Additional Constrain (TSSP+1)* (Doctoral dissertation). The University of Tennessee, Knoxville, TN.

Poggi, M., Viana, H., & Uchoa, E. (2010). The Team Orienteering Problem: Formulations and Branch-Cut and Price. In T. Erlebach & M. Lübbecke (Eds.), *10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10)* (pp. 142–155). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. https://doi.org/10.4230/OASIcs.ATMOS.2010.142

Shaw, P. (1998). Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In M. Maher & J. Puget (Eds.), *Principles and Practice of Constraint Programming (CP98)*. Springer. https://doi.org/10.1007/3-540-49481-2_30

Solomon, M. (1987). Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, *35*(2), 254–265. https://doi.org/10.1287/opre.35.2.254

Souffriau, W., Vansteenwegen, P., van den Berghe, G., & van Oudheusden, D. (2010). A Path Relinking Approach for the Team Orienteering Problem. *Computers & Operations Research*, *37*(11), 1853–1859. https://doi.org/10.1016/j.cor.2009.05.002

Tang, H., & Miller-Hooks, E. (2005). A TABU Search Heuristic for the Team Orienteering Problem. *Computers & Operations Research*, *32*(6), 1379–1407. https://doi.org/10.1016/j.cor.2003.11.008

Tang, H., Miller-Hooks, E., & Tomastik, R. (2007). Scheduling Technicians for Planned Maintenance of Geographically Distributed Equipment. *Transportation Research Part E: Logistics and Transportation Review*, *43*(5), 591–609. https://doi.org/10.1016/j.tre.2006.03.004

Tsiligirides, T. (1984). Heuristic Methods Applied to Orienteering. *Journal of the Operational Research Society*, *35*, 797–809. https://doi.org/10.1057/jors.1984.162

Vansteenwegen, P., Souffriau, W., van den Berghe, G., & van Oudheusden, D. (2009). Methaheuristics for Tourist Trip Planning. In K. Sörensen, M. Sevaux, W. Habenicht, & M. Geiger (Eds.), *Metaheuristics in the Service Industry* (pp. 471–480). Springer. https://doi.org/10.1007/978-3-642-00939-6_2

Voorneveld, M. (2003). Characterization of Pareto Dominance. *Operations Research Letters*, *31*(1), 7–11. https://doi.org/10.1016/S0167-6377(02)00189-X

Wren, A., & Holliday, A. (1972). Computer Scheduling of Vehicles from One or More Depots to a Number of Delivery Points. *Journal of the Operational Research Society*, *23*, 333–344. https://doi.org/10.1057/jors.1972.53

Yu, J., Aslam, J., Karaman, S., & Rus, D. (2015). Anytime Planning of Optimal Schedules for a Mobile Sensing Robot. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5279–5286. https://doi.org/10.1109/IROS.2015.7354122

Yu, Q., Adulyasak, Y., Rousseau, L.-M., Zhu, N., & Ma, S. (2021). Team Orienteering with Time Varying Profit. *INFORMS Journal on Computing*. https://doi.org/10.1287/ijoc.2020.1026

Yu, Q., Fang, F., Zhu, N., & Ma, S. (2019). A Matheuristic Approach to the Orienteering Problem with Service Time Dependent Profits. *European Journal of Operations Research*, *273*(2), 488–503. https://doi.org/10.1016/j.ejor.2018.08.007

Appendix A Average Results

				HALNS-E								HALNS-MCS							
Instance	$\|P\|$	$\|V\|$	T_{max}	Obj. Val.		Time (s)		Gap %		Improve %		Obj. Val.		Time (s)		Gap %		Improve %	
kroA100	2	25	7,106	655.25	(37.47)	49.06	(18.08)	9.69	(5.16)	0.00	(0.00)	408.69	(13.17)	8.04	(4.06)	43.68	(1.82)	0.00	(0.00)
	2	50	7,308	812.26	(69.77)	73.16	(23.79)	42.60	(4.93)	277.91	(32.46)	434.80	(21.48)	3.06	(4.53)	69.27	(1.52)	102.29	(9.99)
	2	75	7,351	1,110.41	(81.04)	118.75	(36.21)	42.56	(4.19)	346.54	(32.59)	475.53	(65.16)	30.11	(13.14)	75.40	(3.37)	91.23	(26.20)
	2	100	7,351	1,021.28	(69.34)	111.94	(36.75)	58.32	(2.82)	1,091.34	(80.88)	390.03	(48.82)	4.67	(2.74)	84.08	(1.99)	354.98	(56.95)
	3	25	4,737	632.87	(26.25)	44.29	(15.80)	7.82	(3.82)	2.68	(4.07)	423.54	(10.27)	0.24	(0.10)	38.31	(1.50)	0.00	(0.00)
	3	50	4,872	847.56	(15.06)	62.05	(16.08)	36.82	(1.12)	310.84	(7.30)	434.92	(37.63)	0.57	(0.37)	67.58	(2.80)	110.82	(18.24)
	3	75	4,900	903.72	(61.16)	74.25	(17.86)	53.62	(3.14)	152.38	(17.08)	462.93	(47.62)	0.82	(0.73)	76.24	(2.44)	29.50	(12.73)
	3	100	4,900	1,244.95	(43.15)	91.99	(22.35)	53.07	(1.63)	>1,000.00	N.A.	534.98	(17.20)	2.36	(1.27)	79.83	(0.65)	>1,000.00	N.A.
	4	25	3,553	379.29	(0.00)	22.35	(7.15)	0.00	(0.00)	0.00	(0.00)	289.99	(4.92)	0.13	(0.02)	23.55	(1.30)	0.00	(0.00)
	4	50	3,654	659.45	(9.58)	46.03	(22.77)	52.66	(0.69)	65.00	(2.40)	384.61	(21.05)	0.23	(0.30)	72.39	(1.51)	0.94	(2.24)
	4	75	3,675	880.93	(28.21)	66.39	(25.14)	49.33	(1.62)	214.19	(10.06)	437.48	(24.39)	0.19	(0.04)	74.84	(1.40)	56.03	(8.70)
	4	100	3,675	1,196.54	(65.86)	78.14	(12.87)	48.06	(2.86)	1,933.42	(111.93)	520.25	(21.46)	0.31	(0.07)	77.42	(0.93)	784.12	(36.48)
kroB100	2	25	6,717	456.17	(15.90	46.36	(17.77)	37.05	(2.19)	0.05	(0.14)	287.29	(19.58)	1.48	(0.24)	60.35	(2.70)	0.00	(0.00)
	2	50	7,051	796.56	(40.88)	82.68	(25.87)	48.67	(2.63)	204.96	(15.65)	499.45	(31.86)	3.62	(3.46)	67.82	(2.05)	91.21	(12.20)
	2	75	7,275	1,163.44	(56.85)	136.79	(32.61)	43.47	(2.76)	>1,000.00	N.A.	463.15	(47.17)	130.58	(218.92)	77.50	(2.29)	>1,000.00	N.A.
	2	100	7,391	1,267.04	(66.13)	126.20	(27.62)	53.26	(2.44)	271.13	(19.37)	502.38	(25.94)	1.67	(0.79)	81.47	(0.96)	47.15	(7.60)
	3	25	4,478	524.69	(9.75)	27.44	(12.38)	30.36	(1.29)	28.90	(2.40)	370.77	(10.17)	0.22	(0.03)	50.79	(1.35)	0.00	(0.00)
	3	50	4,700	723.73	(26.09)	54.81	(18.17)	44.13	(2.01)	177.93	(10.02)	494.33	(29.74)	0.55	(0.24)	61.84	(2.30)	89.83	(11.42)
	3	75	4,850	1,143.61	(48.75)	79.42	(21.52)	38.90	(2.60)	>1,000.00	N.A.	517.68	(25.75)	0.79	(0.44)	72.34	(1.38)	>1,000.00	N.A.
	3	100	4,927	1,410.03	(52.78)	100.08	(16.34)	48.46	(1.93)	1,022.61	(42.02)	676.99	(57.80)	3.59	(8.11)	75.25	(2.11)	439.00	(46.02)
	4	25	3,358	343.63	(0.00)	24.23	(6.64)	0.00	(0.00)	0.00	(0.00)	343.63	(0.00)	0.11	(0.01)	0.00	(0.00)	0.00	(0.00)
	4	50	3,525	750.69	(34.24)	55.87	(10.64)	34.05	(3.01)	74.59	(7.96)	406.57	(36.85)	0.18	(0.10)	64.28	(3.24)	0.81	(1.14)
	4	75	3,637	965.55	(9.44)	67.57	(16.97)	44.23	(0.55)	189.87	(2.83)	574.34	(27.10)	0.46	(0.18)	66.83	(1.57)	72.42	(8.14)
	4	100	3,695	1,302.42	(48.10)	74.03	(22.90)	53.40	(1.72)	823.33	(34.10)	657.53	(46.29)	0.51	(0.23)	76.47	(1.66)	366.15	(32.82)
kroC100	2	25	7,020	640.70	(20.61)	50.55	(10.18)	24.39	(2.43)	15.60	(3.72)	363.38	(8.88)	1.48	(0.51)	57.12	(1.05)	0.00	(0.00)
	2	50	7,186	815.62	(38.17)	62.33	(20.77)	39.37	(2.84)	85.34	(8.67)	487.71	(37.96)	1.95	(3.40)	63.75	(2.82)	11.29	(7.78)
	2	75	7,240	1,131.19	(61.52)	110.99	(35.86)	45.82	(2.95)	720.42	(44.62)	602.63	(25.15)	153.32	(232.41)	71.14	(1.20)	337.07	(18.24)
	2	100	7,345	1,345.62	(75.21)	149.79	(21.11)	51.91	(2.69)	1,879.24	(110.62)	484.08	(31.60)	1.06	(0.46)	82.70	(1.13)	612.02	(46.48)
	3	25	4,680	528.59	(6.04)	32.07	(9.77)	21.60	(0.90)	5.86	(1.21)	369.86	(21.17)	0.23	(0.13)	45.14	(3.14)	0.00	(0.00)
	3	50	4,790	862.91	(34.38)	71.51	(28.99)	35.41	(2.57)	>1,000.00	N.A.	526.68	(63.35)	1.19	(1.30)	60.58	(4.74)	>1,000.00	N.A.
	3	75	4,826	1,139.07	(59.83)	76.19	(15.72)	43.65	(2.96)	240.25	(17.87)	515.33	(24.36)	2.97	(2.30)	74.50	(1.21)	53.93	(7.28)
	3	100	4,896	1,244.76	(31.91)	83.59	(21.13)	50.30	(1.27)	249.06	(8.95)	581.55	(58.93)	2.98	(1.98)	76.78	(2.35)	63.08	(16.53)
	4	25	3,510	341.11	(0.00)	22.83	(7.82)	0.00	(0.00)	0.00	(0.00)	341.11	(0.00)	0.12	(0.01)	0.00	(0.00)	0.00	(0.00)
	4	50	3,593	713.09	(10.39)	46.81	(12.03)	37.27	(0.91)	134.42	(3.41)	509.10	(1.39)	0.19	(0.05)	55.21	(0.12)	67.36	(0.46)
	4	75	3,620	898.20	(38.67)	63.57	(21.88)	51.96	(2.07)	267.25	(15.81)	511.73	(56.00)	0.63	(0.22)	72.63	(3.00)	109.23	(22.90)
	4	100	3,672	1,189.86	(33.23)	61.38	(16.61)	49.70	(1.40)	644.72	(20.80)	580.39	(51.11)	0.27	(0.08)	75.47	(2.16)	263.26	(31.99)

Note: Gaps are given with respect to the upper bound found by `CPLEX` after 1,800 seconds; Improvement is measured with respect to the best feasible solution obtained by `CPLEX` after 1,800 seconds; Standard deviation in parentheses.

Table 10: Average Computational Results HALNS for kroA100, kroB100, and kroC100 over 10 runs

				HALNS-E								HALNS-MCS							
Instance	$\|P\|$	$\|V\|$	T_{max}	Obj. Val.		Time (s)		Gap %		Improve %		Obj. Val.		Time (s)		Gap %		Improve %	
kroA200	2	125	7,345	1,410.27	(64.25)	163.87	(39.43)	56.22	(1.99)	373.14	(21.56)	504.99	(21.30)	0.59	(0.39)	84.32	(0.66)	69.42	(7.14)
	2	150	7,380	1,357.48	(111.94)	169.20	(33.79)	61.70	(3.16)	>1,000.00	N.A.	520.04	(34.07)	0.81	(0.25)	85.33	(0.96)	>1,000.00	N.A.
	2	175	7,380	1,440.21	(87.12)	157.11	(30.76)	65.69	(2.08)	1,959.94	(124.61)	443.43	(2.79)	0.64	(0.45)	89.44	(0.07)	534.23	(3.99)
	2	200	7,380	1,602.02	(67.97)	214.74	(26.75)	66.03	(1.44)	1,389.13	(63.18)	543.12	(86.19)	4.94	(7.39)	88.48	(1.83)	404.85	(80.12)
	3	125	4,896	1,391.06	(90.59)	89.15	(20.25)	56.22	(2.85)	>1,000.00	N.A.	581.32	(87.50)	0.86	(0.43)	81.70	(2.75)	>1,000.00	N.A.
	3	150	4,920	1,499.03	(101.08)	109.45	(36.39)	60.64	(2.65)	509.74	(41.11)	604.88	(35.55)	0.75	(0.16)	84.12	(0.85)	146.04	(13.24)
	3	175	4,920	1,511.54	(119.54)	102.17	(31.34)	62.58	(2.96)	3,102.62	(253.28)	580.32	(30.68)	0.65	(0.27)	85.63	(0.76)	1,129.58	(65.00)
	3	200	4,920	1,774.67	(141.72)	104.05	(33.58)	61.94	(3.04)	508.95	(48.63)	468.15	(71.85)	0.40	(0.27)	89.96	(1.54)	60.64	(24.66)
	4	125	3,672	1,267.66	(68.78)	64.98	(12.58)	57.76	(2.29)	984.99	(58.87)	541.26	(17.51)	0.21	(0.04)	81.96	(0.58)	363.26	(14.99)
	4	150	3,690	1,391.51	(35.62)	83.02	(19.92)	61.39	(0.99)	349.94	(11.52)	633.36	(55.78)	0.55	(0.37)	82.43	(1.55)	104.80	(18.04)
	4	175	3,690	1,525.89	(124.74)	84.23	(17.75)	63.44	(2.99)	567.48	(54.56)	659.10	(26.59)	1.55	(0.97)	84.21	(0.64)	188.31	(11.63)
	4	200	3,690	1,625.16	(47.15)	94.47	(22.69)	64.62	(1.03)	526.47	(18.18)	684.18	(24.88)	0.59	(0.23)	85.11	(0.54)	163.74	(9.59)
kroB200	2	125	7,391	1,399.62	(152.15)	133.81	(47.44)	55.56	(4.83)	405.32	(54.93)	637.87	(57.73)	18.82	(47.25)	79.75	(1.83)	130.30	(20.84)
	2	150	7,391	1,686.75	(74.03)	174.33	(23.85)	55.54	(1.95)	641.49	(32.54)	539.50	(25.33)	7.09	(5.34)	85.78	(0.67)	137.17	(11.14)
	2	175	7,391	1,761.27	(103.59)	184.50	(56.74)	60.03	(2.35)	1,773.75	(110.20)	680.94	(31.86)	2.41	(2.92)	84.55	(0.72)	624.43	(33.90)
	2	200	7,401	1,800.32	(119.11)	204.74	(52.50)	64.61	(2.34)	718.86	(54.18)	555.92	(51.58)	3.31	(2.66)	89.07	(1.01)	152.86	(23.46)
	3	125	4,927	1,517.21	(152.15)	101.23	(23.28)	52.32	(2.57)	506.20	(32.64)	737.14	(52.99)	1.26	(0.66)	76.84	(1.67)	194.52	(21.17)
	3	150	4,927	1,606.85	(97.48)	93.88	(17.99)	56.34	(2.65)	1,613.25	(103.94)	603.94	(42.51)	0.62	(0.29)	83.59	(1.15)	543.93	(45.32)
	3	175	4,927	1,615.80	(95.16)	108.92	(29.92)	62.34	(2.22)	394.48	(29.12)	634.05	(54.15)	2.27	(1.13)	85.22	(1.26)	94.03	(16.57)
	3	200	4,934	1,846.71	(91.51)	144.29	(25.68)	62.37	(1.86)	464.07	(27.95)	557.64	(51.58)	1.00	(2.66)	88.64	(1.01)	70.33	(23.46)
	4	125	3,695	1,300.14	(79.59)	87.65	(30.95)	57.11	(2.63)	1,290.85	(85.15)	634.96	(37.76)	0.79	(0.40)	79.05	(1.25)	579.26	(40.40)
	4	150	3,695	1,494.36	(52.51)	86.33	(18.57)	55.28	(1.57)	1,532.33	(57.36)	702.87	(17.62)	2.46	(1.61)	78.96	(0.53)	667.76	(19.25)
	4	175	3,695	1,656.68	(94.60)	90.22	(22.52)	56.66	(2.47)	694.85	(45.39)	660.41	(61.07)	0.87	(0.39)	82.72	(1.60)	216.86	(29.30)
	4	200	3,700	1,848.31	(110.41)	104.58	(18.93)	60.57	(2.36)	1,097.02	(71.51)	695.34	(74.25)	1.00	(0.48)	85.16	(1.58)	350.32	(48.08)

Note: Gaps are given with respect to the upper bound found by `CPLEX` after 1,800 seconds; Improvement is measured with respect to the best feasible solution obtained by `CPLEX` after 1,800 seconds; Standard deviation in parentheses.

Table 11: Average Computational Results HALNS-E for kroA200 and kroB200 over 10 runs