Computers & Operations Research 143 (2022) 105794

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Time dependent orienteering problem with time windows and service time dependent profits

M. Khodadadian [a], A. Divsalar [b,*], C. Verbeeck [c], A. Gunawan [d], P. Vansteenwegen [e]

[a] Department of Engineering and Technology, Shomal University, Amol, Iran
[b] Department of Industrial Engineering, Babol Noshirvani University of Technology, Babol, Iran
[c] Department of Strategy, EDHEC Business School, Roubaix, France
[d] School of Computing and Information Systems, Singapore Management University, Singapore
[e] KU Leuven Institute for Mobility – CIB, KU Leuven, Leuven, Belgium

ARTICLE INFO

Keywords:
Time dependent orienteering problem
Service time dependent profits
Variable neighborhood search
Tourist trip planning
Real data set

ABSTRACT

This paper addresses the time dependent orienteering problem with time windows and service time dependent profits (TDOPTW-STP). In the TDOPTW-STP, each vertex is assigned a minimum and a maximum service time and the profit collected at each vertex increases linearly with the service time. The goal is to maximize the total collected profit by determining a subset of vertices to be visited and assigning appropriate service time to each vertex, considering a given time budget and time windows. Moreover, travel times are dependent of the departure times. To solve this problem, a mixed integer linear model is formulated and a metaheuristic algorithm based on variable neighborhood search (VNS) is developed. This algorithm uses three specifically designed neighborhood structures able to deal with the variable service times and profits of vertices. Extensive computational experiments are conducted on test instances adapted from the TDOPTW benchmarks, to validate the performance of our solution approach. Furthermore, a real instance for the city of Shiraz (Iran) is generated. Experimental results demonstrate the suitability of the TDOPTW-STP in practice, and demonstrate that the proposed algorithm is able to obtain high-quality solutions in real-time. Sensitivity analyses clearly show the significant impact of the service time dependent profits on the route plan, especially in the presence of travel time dependency and time windows.

1. Introduction

The orienteering problem (OP) is specified on a network in which vertices represent geographical locations where a profit can be gained and where arcs represent connections between vertices with a certain travel time. The OP is the integration of selecting a subset of vertices to visit, with determining an appropriate path for visiting the selected vertices; thereby the sum of accumulated profits is maximized under a limited time budget. Moreover, it is assumed that each vertex can be visited at most once. The OP has many intriguing applications in defense, tourism and logistics (Vansteenwegen et al., 2011). One of the interesting variants of the OP is the Time Dependent OP with Time Windows (TDOPTW) (Verbeeck et al., 2014). In this problem, each vertex has a time window (opening time and closing time) and a deterministic service time, and the travel time required to traverse a link between two vertices relies on the time of the day the link is passed. As mentioned in Verbeeck et al. (2014), formulating the time dependent problem makes us capable of considering congestion in (multi-modal) routing problems applied in logistic or tourist trip planning. Another relevant variant of the OP in the literature is the OP with Service Time dependent Profits (OPSTP) (Yu et al., 2019), where the collected profit at each vertex is not a fixed single value but depends on the duration of its service or visit time. In this problem, restricted by a time budget, it may not be possible or desirable to gather the maximum profit at each visit. Thus, in order to maximize the total profit, not only a subset of vertices needs to be selected, but also the suitable service time at each selected vertex has to be determined. Many real world applications for the OPSTP has been introduced in the literature. For example, the fishing problem, in which there is a limited allowed time for fishing at each location, having variable amount of fish (Erdoğan and Laporte,

* Corresponding author.
E-mail addresses: khodadadian@shomal.ac.ir (M. Khodadadian), ali.divsalar@nit.ac.ir (A. Divsalar), cedric.verbeeck@edhec.edu (C. Verbeeck), aldygunawan@smu.edu.sg (A. Gunawan), pieter.vansteenwegen@kuleuven.be (P. Vansteenwegen).

https://doi.org/10.1016/j.cor.2022.105794
Received 18 August 2021; Received in revised form 15 March 2022; Accepted 15 March 2022
Available online 18 March 2022

2013). Other examples are the tourist trip planning (Yu et al., 2015), the traffic routing problem (Zhu et al., 2014), and finding the shortest path problem for target searching (Guitouni and Masri, 2014; Pietz and Royset, 2013).

In the current research, a combination of TDOPTW and OPSTP is addressed which we call the TDOPTW-STP. Many interesting applications of TDOPTW-STP can be found in a situation where each vertex has a time window and a service time dependent profit, while the travel time between two vertices relies on the time of the day that link is passed.

In the presented TDOPTW-STP, each vertex has lower and upper bounds for its service time and the collected profit at each vertex is assumed to increase linearly with the service time. The addition of this service-time dependent profit (STP) to the TDOPTW makes the problem more realistic where in real-life, higher profits can be collected during a longer service time, respecting the corresponding lower and upper bounds. Here, we mention some possible applications of the TDOPTW-STP in tourism, logistics, entertainment, and military sectors, where each vertex may have a bounded profit as a linear function of its service time.

In the tourism sector, POIs have opening and closing times, and network travel times may change significantly in peak and off peak hours. On the other hand, duration of visiting POIs by different tourists may be different. For instance, in visiting a museum, some tourists stay longer to collect more information. This increases their visit time, but also their appreciation. In this case, the collected profit depend on the visiting duration at the museum (Vansteenwegen and Gunawan, 2019). In supply chain management, and the VRP with profits for example, or reverse logistics, gaining more profit may require a longer service time at a customer, while travel times depends on traffic congestion (Vansteenwegen and Gunawan, 2019). In the entertainment sector, it is trivial that a longer show at a location, results in a higher profit that can be collected (Erdoğan and Laporte, 2013). It is also probable that show locations have time windows and that travel times between them vary during the day because of weather and traffic conditions. Other applications arise in the military sector and humanitarian logistics, e.g. search and rescue, where a vehicle looks for gathering information and finding survivors, staying longer at target locations should increase the amount of information and number of survivors (Erdoğan and Laporte, 2013). In these situations, depending on conditions of target locations and network properties, target locations may be constrained by time windows, and travel times may fluctuate.

Note that, in all these cases, at each vertex, a minimum service time should be passed to gain the minimum profit and besides, the collected profit has an upper bound as the maximum profit. Moreover, incorporating time windows and time dependency, the variable service time at a vertex in the TDOPTW-STP provides the decision opportunity to consume the time budget more efficiently in the trade-off between travel time and service time. This new combination of properties is a motivation for researchers and practitioners to develop new models and algorithms in resource management.

The main contributions can be summarized as follows:

- The TDOPTW-STP is introduced, motivated and mathematically modelled.
- A metaheuristic solution algorithm based on the variable neighborhood search (VNS) is proposed to solve the problem.
- A set of problem instances with known optimal solutions are generated based on existing benchmark instances of TDOPTW and are used to evaluate the performance of the solution method.
- A real data set based on a real road network is generated and used to show how the TDOPTW with service time dependent profits (STP) can be tackled in a practical case.

Following a literature review in Section 2, the mathematical model of the TDOPTW-STP is described in Section 3. Then, the solution algorithm is explained in Section 4. Afterwards, results and discussion is presented in Section 5. Section 6 concludes this paper and discusses possible future work.

2. Literature review

Tsiligirides (1984) introduced the standard OP that has been proven to be NP-Hard by Golden et al. (1987). Feillet et al. (2005) and Vansteenwegen and Gunawan (2019) give an overview of the category of Traveling Salesman Problems with profits, to which OP belongs. Extensive surveys on the OP and its extensions, solution algorithms and applications are given by Vansteenwegen et al. (2011) and Gunawan et al. (2016). The rest of this literature section is devoted to the two most relevant topics for the current research: the time dependency and variable profits for OP variants.

In the OP, it is assumed that the link travel time is a constant value. However, in many practical situations, it actually depends on the network conditions, such as transportation mode and traffic congestion level. This variant of the OP is called the time dependent OP (TDOP) and is introduced by Fomin and Lingas (2002). Afterward, Li et al. (2010) and Verbeeck et al. (2014) investigated the TDOP more thoroughly. Li et al. (2010) used the dynamic programming idea for an optimal labeling algorithm, which is combined by a mixed integer programming model to solve the TDOP. However, they did not test their algorithm on benchmark instances and thus did not propose performance metrics. Verbeeck et al. (2014) designed a local search based metaheuristic, combined by an ant colony system and tested it on modified OP instances that consist of a complete graph with categorized link travel times. For the time dependent team orienteering problem (TDTOP) several solution algorithms have been proposed in the literature. Li (2011) proposed a mixed integer programming model and an optimal dynamic labeling algorithm but without testing on test instances and presenting any performance metrics. Gavalas et al. (2014) and Gavalas et al. (2015) designed various cluster-based heuristics and tested it on instances created based on the metropolitan area of Athens.

The TDOPTW adds both time dependency in travel time and time window constraints on the moment of service to the basic OP. Abbaspour and Samadzadegan (2011) applied multimodal shortest path finding modules and two adapted genetic algorithms to the TDOPTW. Their algorithm is evaluated over datasets derived from the metropolitan area of Tehran without reporting any performance measure. Verbeeck et al. (2017) proposed an effective and efficient solution method by applying the swap and replace local search procedures and based on the ant colony system with a pre-processing step. This algorithm is verified by realistic instances that originate from a large road network of Benelux (Belgium, The Netherlands and Luxembourg) with available time profiles of link travel times. These instances are the only available benchmarks for TDOPTW in the literature.

Zenker and Ludwig (2009) introduced the TDTOPTW for the first time. However, they have not proposed any solution algorithm for the presented model. They developed a mobile application (ROSE) that presents recommendations, route generation, and navigation to assist pedestrians to access vertices Points of Interest (POIs) by public transportation. Garcia et al. (2009) and Garcia et al. (2013) designed two solution algorithms for the TDTOPTW that are applied to real urban test instances. The first work solved the TDTOPTW as a TOPTW by using the average travel times between all pairs of POIs. Then, a repair procedure checks the derived TOPTW solution based on real travel times and removes some included POIs to provide feasibility conditions. The second work presented an ILS heuristic which finds a near-optimal solution in a few seconds. In this work, a tourist can choose between walking and using public transportation with periodic schedules. The waiting time for public transport depends on the arrival time to the corresponding station. As a result, the travel time between POIs depends on the departure time at a POI and the transportation mode. Liao and Zheng (2018) proposed a hybrid heuristic algorithm based on random simulation to design a personalized day tour in a time dependent stochastic

environment. In this situation, various factors such as the weather, the traffic, the limited capacity of POIs or unforeseen events are stochastic and time dependent. However, the proposed solution by this work is not evaluated on benchmark instances and therefore no performance measures are proposed.

The basic assumption in the classical OP and all above-mentioned variants is that the visit (service) time of vertices is known in advance and the complete profit is obtained when reaching a vertex. The orienteering problem with variable profits (OPVP) is another interesting variant of the OP (Erdoğan and Laporte (2013) in which the obtained profit is not fixed. Yu et al. (2019) have classified OPVP into two streams: (i) the OP with arrival time dependent profits (OPATP), in which profits depend on the arrival time at each vertex, (ii) the OP with service time dependent profits (OPSTP), in which profits depend on the duration of the service time at each vertex. The latter one will be considered in this paper.

Erkut and Zhang (1996) introduced the OPATP where each vertex has a linear decreasing profit as a function of its arrival time, and the goal is to find a single route maximizing the total collected profit. In their paper, small size instances are solved using an exact branch-and-cut method, and large size instances are solved using a greedy heuristic. An OPATP with multiple tour is introduced by Tang et al. (2007) where they developed a tabu search with an adaptive memory to solve instances of the problem. Ekici and Retharekar (2013) introduced the OPATP with multiple vehicles, and a cluster-first route-second heuristic is developed by them. Murat Afsar and Labadie (2013) found lower and upper bounds of the OPATP instances using column generation and evolutionary local search techniques. Peng et al. (2019) addressed the agile earth observation satellite scheduling problem as the TDOPTW with time dependent profits. In this case, the collected profit depends on the start time. Moreover, time dependency and time windows are added to the basic OPATP. They proposed a bidirectional dynamic programming based iterated local search (BDP-ILS). The performance evaluation shows that their solution method performs very well.

In the OPSTP problem, one needs to find the route of the vertices to visit, as well as the proper service time (duration) at each selected vertex. This problem is firstly considered by Erdoğan and Laporte (2013) where they assume that the complete profit at a vertex is gained with several discrete passes or a continuous amount of time to be spent at that vertex. They formulated the discrete model as a linear integer programming problem and the continuous model as a nonlinear integer programming problem. Then, they provided valid inequalities to strengthen the formulation for both concave and convex profit collection functions and developed a unified branch-and-cut algorithm for the two versions. Pietz and Royset (2013) defined the Generalized Orienteering Problem with Resource Dependent Rewards (GOP-RDR) where rewards are collected by visiting each vertex with a concave reward function, and the reward level depends on the amount of limited resources consumed. Arcs in the network are passed while expending the same resources used for reward collection. The path is built up so that the total resource consumption is within predetermined bounds. They gave a branch and bound algorithm for their variant of OPSTP that solves a series of convex partial path relaxations. They developed a heuristic algorithm to start their branch and bound tree with a high-quality solution, speeding up the computation. Regardless of differences arising from the various types of profit function, note that firstly, the OP-STP is a special case of GOP-RDR (as an extension of GOP) in which time budget is considered as the limited resource, secondly, the TDOPTW-STP is not a special case of GOP, due to each of its all 3 crucial properties (TD, TW, and STP), and also is not a special case of GOP-RDR or OP-STP, due to each of its two crucial properties (TD, and TW), and on the other hand, TDOPTW-STP is an extension of TDOPTW and OP-STP.

Guitouni and Masri (2014) studied a search-and-rescue path planning for an aerial search aid in continuous space and time. Their problem is similar to OPVP in the sense that at every vertex the vehicle gathers a proportion of the profit that is conditioned by the time used on that vertex. An optimal solution was found for a numerical instance of ten vertices in 42 min. Yu et al. (2015) have used the OPSTP model for the tourist trip planning, where an increasing profit function depends on the duration of stay at each vertex. They used a piecewise linear approximation for the nonlinear profit function, and used Gurobi to solve the proposed model, and found near-optimal solutions to the nonlinear problem. Gunawan et al. (2018) are the first to present a mathematical programming model for the TOPVP as an extension of the OPVP discrete model, and developed a solution algorithm based on ILS proposed by Chao et al. (1996). Their local search moves consist of two-point exchange, one-point movement, and 2-Opt. For performance evaluation, the proposed algorithm was run on OPVP benchmark instances based on TSP test instances. The authors concluded that the algorithm is able to find optimal solutions in considerably shorter computational time for the small instances and good-quality solutions that have significantly better objective values than those found by CPLEX under reasonable run times. Based on Yu et al. (2019), in the OPSTP problem, the extra decision of assigning appropriate service times to each selected vertex causes the complexity of designing heuristic algorithms for this problem. They conclude that a proper heuristic should consider that: First, a comprehensive representation of the solutions is important; second, since both traveling times and service times use the common source time (the time budget), they may severely interact on each other, which causes another difficulty in determining auspicious solution space regions during the search process. A variant of the OPSTP with a concave service time function is also addressed by Yu et al. (2019). They formulated a mixed integer nonlinear programming model, and developed a matheuristic which decomposes the problem into two subproblems, of routing and scheduling. Then, a tabu search is proposed to find feasible solutions for the routing subproblem, and an exact polynomial time algorithm is designed to optimally solve the scheduling subproblem, which finds the service times of visited vertices. Numerical experiments on both random and adapted TSPLIB instances show the effectiveness of their proposed solution method on instances with up to 200 vertices.

Another relevant problem in the literature is called the patrol routing problem (PRP). In the PRP, given a set of patrol cars and a set of hot spots, the objective is to find patrol cars' routes maximizing the time that these cars spend in hot spots. Each hot spot is assigned a time window and cars receive profits only if they stay at the hot spot within its corresponding time window. Constant travel times between hot spots are known, and all cars start and end their shift at the same times and locations (Dewil et al., 2015). Various variants and applications of the PRP has been studied in Moonen et al. (2007), Takamiya and Watanabe (2011), Lou et al. (2011), Willemse and Joubert (2012), Keskin et al. (2012), Chen (2012), Portugal and Rocha (2013), Chircop et al. (2013), and Dewil et al. (2015). Since the relation between the profit and the service time is linear, the TDOPTW-STP seems very close to the PRP, but there are crucial differences. In the PRP, a certain vertex is left only when it closes or when another vertex with a higher profit rate (collected profit per time) opens at that moment, but in the TDOPTW-STP, there is a maximum service time for each vertex. Thus, for a certain visited vertex, the time of completing the maximum service time has the key role in determining the departure time while this time for a visited vertex is not predetermined on the time profile and depends on its arrival time. Furthermore, in the TDOPTW-STP, there is a nonzero minimum service time for each vertex while in the PRP, as soon as the service time is started, the profit is collected. Moreover, profits in the TDOPTW-STP have a strictly positive offset, unrelated to the profit per time unit. Due to these properties, the PRP instances can easily be solved exactly, by modeling the problem as a minimum cost network flow problem (Dewil et al., 2015). However, this is not applicable to the TDOPTW-STP.

To the best of authors' knowledge, the only work which considers both time dependency and service time dependent profits together in the OPTW, is a recent conference paper by Gündling and Witzel (2020).

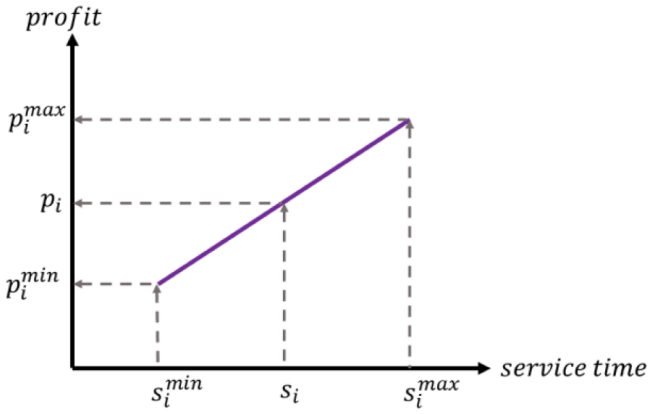

Fig. 3.1. The profit of a vertex as a linear function of service time.

They studied TDTOPTW with adjustable profits and presented the first MILP representation of it. In this work, profit is characterized as a linear function of service time (similar to the work presented in this paper). Without presenting details about the solution approach, they presented a fast iterated local search (ILS) that can be used in practical applications such as a mobile app service for tourists. They claim the practical feasibility of their approach by applying it on the data taken from the city of Berlin. In their ILS, the visit time is dealt with discreetly in 5 min steps. Thus, it seems that it discretely searches the solution space to find the appropriate service time, while our algorithm in this paper does that continuously.

In the TDOPTW-STP, time dependent travel times and service time dependent profits are added simultaneously to the OPTW. In fact, by integrating routing and scheduling, the user (planner, tourist, or driver) tries to consume the time budget for visiting vertices and collecting profit rather than spending the time in the traffic network. Despite its interesting applications, according to the literature review, this new variant of the OPTW is not fully addressed. Moreover, although there are similar problems in the literature, such as PRP and OP-STP, in none of them lower and upper bounds are imposed on service times and profits of each vertex. As briefly mentioned earlier, this property makes the problem even more difficult to solve. In this research, the TDOPTW-STP with minimum and maximum service times is introduced and mathematically modelled. A solution method is implemented, and new instances of the problem are generated and solved using the proposed solution method.

3. Problem description

In this section, the time dependent orienteering problem with time windows and service time dependent profits (TDOPTW-STP) is mathematically defined and formulated as a Mixed Integer Programming (MIP) model. The proposed TDOPTW-STP can be explained on the set $V_c = \{1, \ldots, n\}$ of vertices. In this set, vertex 1 represents the start depot, and vertex n is the end depot. We assume all vertices i and j in V_c are connected by an arc (i,j). Each vertex $i \in V_c$ is assigned a non-negative minimum profit, p_i^{min}, and a maximum profit, p_i^{max}. The effective profit is obtained by visiting the vertex i for a time length of s_i units between its opening time o_i and its closing time $c_i + s_i^{min}$, where s_i^{min} is the minimum required service time to earn the minimum profit of vertex i. The start time of a service at vertex i can only happen within its time window and the end time of service at each vertex i may not exceed $c_i + s_i^{min}$. This implies that if you start a service at time c_i, only the minimal profit can be collected. As presented in Fig. 3.1, the service time for vertex i (s_i) is a continuous variable between s_i^{min} and s_i^{max}, and the profit of vertex i (p_i) is a continuous variable depending on s_i and resides between p_i^{min} and p_i^{max}. A set of linear equations, presented in Eqs. (3-1) to (3-3), is used to formulate the relationship between p_i and s_i. Note that, when the MINLP is presented, p_i is effectively eliminated from the mathematical model of TWOPTD-STP by using these equations. Finally, the corresponding profit and service time for the start and the end depot is set equal to 0.

$$p_i = a_i.s_i + b_i \tag{3-1}$$

$$a_i = \left(p_i^{max} - p_i^{min}\right) / \left(s_i^{max} - s_i^{min}\right) \tag{3-2}$$

$$b_i = p_i^{min} - a_i.s_i^{min} \tag{3-3}$$

Similar to the TDOPTW proposed by Verbeeck et al. (2017), it is assumed that a visit day is divided in k time slots. Furthermore, it is also assumed that lts_t and uts_t indicate the minute when time slot t starts and ends respectively. Afterwards, according to these time slots and their related travel times for each arc (tt_{ij,lts_t}), the linear travel time factors for each arc μ_{ijt} and ϑ_{ijt} can be found using the following equations:

$$\mu_{ijt} = \left(tt_{ij,uts_t} - tt_{ij,lts_t}\right) / \left(uts_t - lts_t\right) \tag{3-4}$$

$$\vartheta_{ijt} = tt_{ij,lts_t} - \mu_{ijt}.lts_t \tag{3-5}$$

Having these coefficients, the travel time from i to j, when departing at time w_{ijt} in time slot t ($lts_t \le w_{ijt} \le uts_t$), is calculated as follows:

$$tt_{ijt} = \mu_{ijt}.w_{ijt} + \vartheta_{ijt} \tag{3-6}$$

In the TDOPTW-STP, the objective is to find a route (solution) which starts and ends in the given depots, and visits a number of vertices to maximize the total collected profit while satisfying a given time budget. Based on this definition, two mathematical models are presented. The first one is a Mixed Integer NonLinear Programming (MINLP) and the second one is its corresponding linear version (MILP) where in both of them the service time is considered as a continuous variable. Decision variables and parameters of the presented MIP models are listed below.
Decision variables

x_{ijt}	1 if arc (i,j) is traversed with a departure time in time slot t by the route, 0 otherwise
w_{ijt}	departure time from i to j in time slot t
s_j	service time of vertex j

Parameters

s_j^{min}	minimum service time of vertex j
s_j^{max}	maximum service time of vertex j
p_j^{min}	minimum profit of vertex j
p_j^{max}	maximum profit of vertex j
a_j	slope coefficient of the linear profit
b_j	intercept coefficient of the linear profit
$tt_{ijw_{ijt}}$	the travel time from vertex i to vertex j, when departing at time w_{ijt} in time slot t
μ_{ijt}	slope coefficient of the linear time dependent travel time
ϑ_{ijt}	intercept coefficient of the linear time dependent travel time
lts_t	start of time slot t
uts_t	end of time slot t
n	number of vertices
k	number of time slots
o_i	opening time of vertex i
c_i	closing time of vertex i
c_n	closing time of end depot

The TDOPTW-STP is modelled as a MINLP model as follow.

$$Max \sum_{i=1}^{n-1} \sum_{j=2}^{n} \sum_{t=1}^{k} \left[\left(a_j.s_j + b_j\right).x_{ijt}\right] \tag{3-7}$$

$$\sum_{j=1}^{n} x_{1j1} = \sum_{i=1}^{n-1} \sum_{t=1}^{k} x_{int} = 1 \tag{3-8}$$

$$\sum_{i=1}^{n-1}\sum_{t=1}^{k} x_{iht} = \sum_{j=2}^{n}\sum_{t=1}^{k} x_{hjt} \leq 1; \forall h = 2, \cdots, n-1 \tag{3-9}$$

$$x_{ijt}.lts_t \leq w_{ijt}; i = 1, \cdots, n-1, j = 2, \cdots, n, \forall t \tag{3-10}$$

$$w_{ijt} \leq x_{ijt}.uts_t; i = 1, \cdots, n-1, j = 2, \cdots, n, \forall t \tag{3-11}$$

$$\sum_{i=1}^{n-1}\sum_{t=1}^{k}[w_{iht} + \mu_{iht}.w_{iht} + \vartheta_{iht}.x_{iht} + s_h.x_{iht}] \leq \sum_{j=2}^{n}\sum_{t=1}^{k} w_{hjt}; \forall h = 2, \cdots, n-1 \tag{3-12}$$

$$\sum_{i=1}^{n-1}\sum_{t=1}^{k}[w_{int} + \mu_{int}.w_{int} + \vartheta_{int}.x_{int}] \leq c_n \tag{3-13}$$

$$\sum_{i=1}^{n-1}\sum_{t=1}^{k}[o_h.x_{iht} + s_h.x_{iht}] \leq \sum_{j=2}^{n}\sum_{t=1}^{k} w_{hjt}; \forall h = 2, \cdots, n-1 \tag{3-14}$$

$$\sum_{j=2}^{n}\sum_{t=1}^{k} w_{hjt} \leq \sum_{i=1}^{n-1}\sum_{t=1}^{k} \left(c_h + s_h^{min}\right).x_{iht}; \forall h = 2, \cdots, n-1 \tag{3-15}$$

$$s_j^{min}.\sum_{i=1}^{n-1}\sum_{t=1}^{k} x_{ijt} \leq s_j \leq s_j^{max}.\sum_{i=1}^{n-1}\sum_{t=1}^{k} x_{ijt}; \forall j = 2, \cdots, n \tag{3-16}$$

$$w_{1i1} = 0; \forall i = 1, \cdots, n \tag{3-17}$$

$$x_{ijt} \in (0, 1); \forall i, j, t \tag{3-18}$$

$$w_{ijt} \in [0, t^{max}]; \forall i, j, t \tag{3-19}$$

The objective function (3-7) maximizes the total collected profit. Constraint (3-8) ensures that vertex 1 and vertex n are the start and end points of the route, respectively. Constraints (3-9) guarantee that each vertex is visited at most once, and ensure the flow balance of the route. Constraints (3-10) and (3-11) specify the departure time in the appropriate time slot required to multiply with its related μ and ϑ in Constraints (3-12) and (3-13). Constraints (3-12) make sure that the departure time of the next vertex in the route is larger than or equal to the sum of the departure time of the previous vertex together with the service time and the travel time. Constraint (3-13) ensures that the arrival time to vertex n is not exceeding its closing time. Note that the time budget (t^{max}) is equal to the difference between the closing time of the end depot and the opening time of the start depot ($c_n - o_1$). So, if the opening time of the start depot is set equal to zero ($o_1 = 0$), the time budget will be equal to the closing time of the end depot ($t^{max} = c_n$). Constraints (3-14) make sure that at each vertex except from the start and the end depot, the departure time does not happen before the opening time plus the service time. Constraints (3-15), ensure that the departure time is not greater the sum of the closing time and the minimum service time of the vertex. Constraints (3-16) ensure that the service time at each visited vertex is between its minimum and maximum service time (s_i^{min} and s_i^{max}). Moreover, without loss of generality, Constraints (3-17) ensure that the start time of the route is at time zero and there is no waiting at the start depot.

In comparison with TDOPTW, the difference is in the objective function and Constraints (3-12), (3-14), (3-15) and (3-16). Note that because of using $s_j.x_{ijt}$ in Eqs. (3-7), (3-12), and (3-14), this problem is modelled as a nonlinear problem.

3.1. Linearization

To linearize and solve the model using the linear solver CPLEX, auxiliary variables d_{ijt} are defined as Eq. (3-20). So that d_{ijt} equals to s_j if $x_{ijt} = 1$, and 0, otherwise.

$$d_{ijt} = s_j.x_{ijt}; \forall i, j, t \tag{3-20}$$

Using auxiliary variables d_{ijt}, a MILP model is developed in which nonlinear Eqs. (3-7), (3-12), and (3-14) are linearized by using Eqs. (3-21) to (3-27). Note that in this MILP, M is a large enough number and its value is set as the upper bound of the continuous and non-negative variable s_j ($Max\left\{s_j^{max}; j = 2, \cdots, n\right\}$).

$$Max \sum_{i=1}^{n-1}\sum_{j=2}^{n}\sum_{t=1}^{k}\left[a_j.d_{ijt} + b_j.x_{ijt}\right] \tag{3-21}$$

$$\sum_{i=1}^{n-1}\sum_{t=1}^{k}[w_{iht} + \mu_{iht}.w_{iht} + \vartheta_{iht}.x_{iht} + d_{iht}] \leq \sum_{j=2}^{n}\sum_{t=1}^{k} w_{hjt}; \forall h = 2, \cdots, n-1 \tag{3-22}$$

$$\sum_{i=1}^{n-1}\sum_{t=1}^{k}[o_h.x_{iht} + d_{iht}] \leq \sum_{j=2}^{n}\sum_{t=1}^{k} w_{hjt}; \forall h = 2, \cdots, n-1 \tag{3-23}$$

$$d_{ijt} - s_j \leq 0; i = 1, \cdots, n-1, j = 2, \cdots, n, \forall t \tag{3-24}$$

$$d_{ijt} \leq M.x_{ijt}; i = 1, \cdots, n-1, j = 2, \cdots, n, \forall t \tag{3-25}$$

$$s_j - d_{ijt} + M.x_{ijt} \leq M; i = 1, \cdots, n-1, j = 2, \cdots, n, \forall t \tag{3-26}$$

$$d_{ijt} \in (0, s_j); \forall j, j, t \tag{3-27}$$

Consequently, in the presented MILP model for the TDOPTW-STP, the objective function is Eq. (3-21), and the constraints are Eqs. (3-8) to (3-11), (3-13), (3-15) to (3-19) and, (3-22) to (3-27).

4. Proposed solution algorithm

Due to the NP-hardness of the TDOPTW-STP, commercial solvers like CPLEX can only solve it for small instances. Furthermore, state of the art methods cannot tackle it because of the variable service time. So, it is necessary to develop an efficient solution method for the TDOPTW-STP. This section of the paper describes the proposed metaheuristic algorithm.

4.1. General structure of the algorithm

Several metaheuristic methods have been presented in the literature to solve various extensions of the OP. Based on the literature, combining Variable Neighborhood Search (VNS) or Variable Neighborhood Descent (VND) with other metaheuristic algorithms were one of the most commonly used approaches due to their favorable results (Divsalar et al., 2013, Paydar and Saidi-Mehrabad, 2013, Dib et al., 2017, Divsalar et al., 2014). The major characteristic that makes a VND-based metaheuristic effective for OP variants is the simplicity and flexibility of this framework. Very few parameters needs to be set and various neighborhood structures can be handled (Divsalar et al., 2013). This latter characteristic of the VND makes it handy to apply both profit increasing and time saving moves simultaneously (Vansteenwegen and Gunawan, 2019). This property has been used to design and apply specific local search moves, which try to find a good order of visits together with the corresponding visit times.

The proposed algorithm is based on the variable neighborhood search (VNS). The general structure of the proposed VNS algorithm is illustrated in Algorithm 4.1. The algorithm begins with an Initialization phase, described in Section 4.2. Then, an improvement phase is implemented iteratively which is described in Section 4.3. *NoImprovement* counts the number of subsequent iterations without improvement and is limited by *MaxNoImprovement* as the stopping criterion of the algorithm. *MaxNoImprovement* is one of the input parameters that is predetermined

when tuning this VNS.

Algorithm 4.1 Pseudo code for the VNS

```
1:  Initialization phase: (Section 4.2)
2:  Preprocessing
3:    S0 ← GenerateInitialSolution;
4:    S* ←VND (S0);
5:  Improvement phase: (Section 4.3)
6:    while NoImprovement < MaxNoImprovement do
7:    S1 ← Shake (S*); (Section 4.3.1)
8:    S2 ← VND (S1); (Section 4.3.2)
9:    S* ←Recentering (S*, S2); (Section 4.3.3)
10: end while
11: return S*
```

To solve a TDOPTW-STP instance, the main decisions are to determine the vertices to visit, and the corresponding departure times and service times. The combination of time dependent travel time and variable service time is the new challenging part of the TDOPTW-STP. This is mainly because in the presence of time dependent travel times, the selection of different service times significantly affects the whole route. Therefore, to deal with this challenge, the proposed VNS contains new contributions compared to the literature. These algorithmic extensions and adaptations are explicitly mentioned in each related section.

4.2. Initialization phase

This phase includes three steps: Preprocessing, GenerateInitialSolution and VND. Similar to the TDOPTW proposed by Verbeeck et al. (2017), a sorted set of nearest neighbors is identified for each vertex i in the preprocessing step. So firstly, for each vertex i, a list of neighbors is defined as a set of vertices j if Eq. (4-1) holds.

$$o_i + s_i^{min} + tt_{ij}^{min} \leq c_j \tag{4-1}$$

where, tt_{ij}^{min} is the minimum time dependent travel time on arc (i,j). This equation guarantees that it is feasible to arrive to vertex j before the closing time when leaving at the earliest possible departure time at vertex i.

Unlike the proposed solution algorithm for the TDOPTW by Verbeeck et al. (2017), preliminary experiments showed that pre-sorting neighbors in these lists does not lead to finding higher quality solutions or reducing the computational time. These results are included in Appendix A. In fact, the combination of time dependency and variable service times makes this pre-sorting useless, and it is very likely that vertices with lower rank are present in the optimal solution.

In the second step, GenerateInitialSolution is performed to produce an initial solution (S_0). This step starts with an empty solution that includes only the start and the end depot. Then, it tries to add, vertices (with their minimum service time) to the solution one by one, based on the nearest neighbor strategy in terms of *NSR* (stands for Node Selection Ratio) as defined in Eq. (4-2).

$$NSR = \begin{cases} dprofit^2/dtime; dtime > 0 \\ (1 - dtime).dprofit^2; dtime \leq 0 \end{cases} \tag{4-2}$$

where, *dprofit* is equal to minimum profit of the neighbor that might be inserted, and *dtime* is the change in the total travel time of the route if the insertion happens. Note that the triangle inequality property might be violated in time dependent OP (Verbeeck et al., 2014) which implies that we might have a negative *dtime* by visiting an extra vertex or a positive *dtime* by removing an included vertex. Therefore, in a time dependent OP the feasibility should be controlled even when one vertex is removed from the solution.

If the travel time increases, *NSR* is set equal to $dprofit^2/dtime$. This means that the more profit and the less travel time, the more promising an insert becomes. Moreover, the square of the profit is applied in *NSR* calculation to give more importance to profit than to travel time. Furthermore, if the travel time decreases or stays the same, *NSR* is set equal to $(1 - dtime).dprofit^2$. This means that as *dtime* becomes more and more negative, $(1 - dtime)$ becomes more and more positive, thus artificially increasing the *NSR*. The purpose is to give vertices that will shorten the route, an advantage upon vertices that will lengthen the route.

Finally, the variable neighborhood descent (VND) is executed on the initial solution to generate the best solution (S*). VND is described in the following subsections.

4.3. Improvement phase

In this phase, three steps are implemented, iteratively. Firstly, the vertices in the current solution are shaken. The algorithm implements the shake as the diversification procedure to better explore the whole solution space. Secondly, a VND with various local search moves is applied. The algorithm uses VND as the intensification procedure. Afterwards, in the recentering step, it is decided from which solution the next iteration should start. In designing the proposed method to achieve an excellent balance between diversification and intensification during the search for the best solution, both aspects of vertex and service time selection are considered.

4.3.1. Shake

VNS escapes from local optima using a shaking phase. In this step, the shake proposed by Vansteenwegen et al. (2009) for TOPTW is adapted for the TDOPTW-STP and presented in Algorithm 4.2. During this phase, one or more included vertices will be removed from the current solution. This Shake has two input parameters; the first remove position (POS_{shake}) and the number of consecutively included vertices to remove (N_{shake}) in the current solution. For the first iteration, POS_{shake} and N_{shake} are initialized to one. Note that in this algorithm, when vertices are removed, because of the time dependency, the solution feasibility is re-checked (line 8). Moreover, every recalculation and updating of time variables such as travel times and *MaxShift* are adopted according to time dependency.

In our algorithm, at the end of each iteration, POS_{shake} is increased by N_{shake}, and N_{shake} is increased by one. As a result, throughout the iterations, different parts of the solution, with different number of vertices, will be maintained and removed. If the number of vertices in the solution is lower than 4 (including the start and the end depot), POS_{shake} and N_{shake} are reset to one. If POS_{shake} or N_{shake} is higher than the number of vertices in the solution, it is reset to one.

Algorithm 4.2 Pseudo code for Shake

```
1:  Remove N_shake included vertices from POS_shake to POS_shake + N_shake-1 in current
    solution;
2:  for included vertex (exactly) before POS_shake do
3:    Recalculate tt_ijdt;
4:  end for
5:  for each included vertex at and after POS_shake do
6:    Recalculate at, wt, sst, dt, tt_ijdt;
7:  end for
8:  if remove is feasible then
9:    for each included vertex after POS_shake do
10:     Update MaxShift;
11:   end for
12: for each included vertex at and before POS_shake do
13:   Calculate MaxShift;
14: end for
15: end if
```

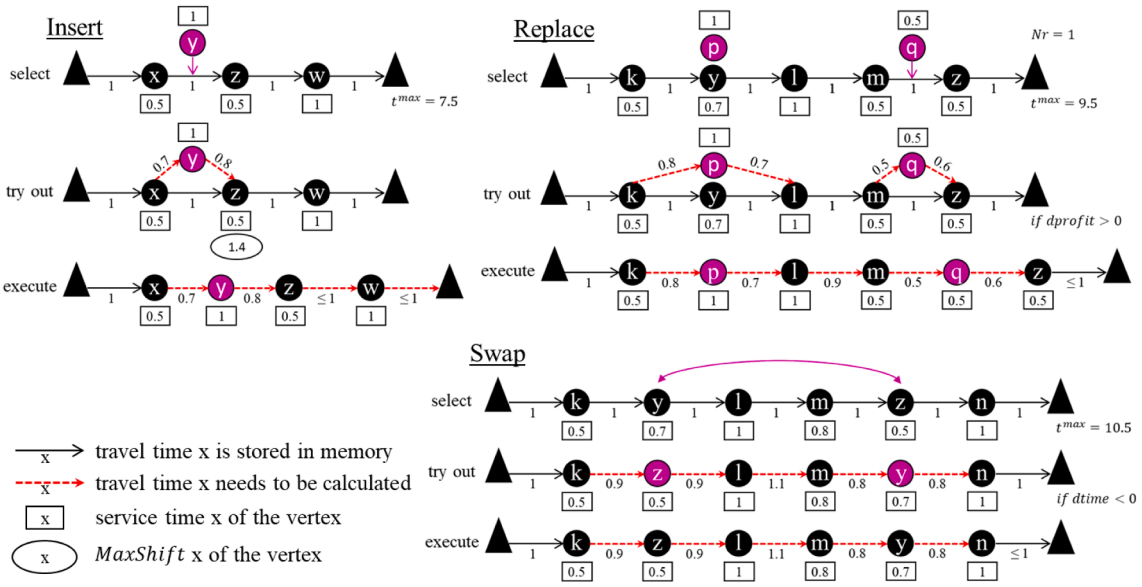

Fig. 4.1. Overview of local search moves.

4.3.2. VND

Three local search moves are used in the VND part of the algorithm. The pseudo code for the VND is shown in Algorithm 4.3. In this local search procedure, the general idea in each iteration is to first insert a higher number of vertices with short service time and then prolong the service time and or replace vertices if it improves the solution. In an overview, the VND starts with Insert-minS which tries to increase the total profit by adding a new vertex with its minimum service time. This move is designed to intensify the solution by spending time on adding an extra vertex rather than increasing service times at visited vertices. The VND continues with a Swap move that attempts to decrease the total travel time by exchanging two included vertices without any changes in their service times. Finding the shortest route between the included vertices is not an explicit part of the OP objective. However, the shorter the route, the more likely it is that extra vertices can be added or the service time of the included ones can be prolonged. Finally, VND performs a Replace move that endeavors to increase the total profit by removing at least one included vertex and adding at least one new vertex with its most appropriate service time. This move itself is composed of multiple neighborhood structures designed to intensify the solution by finding both proper vertices and service times simultaneously. A visual overview of each local search move is provided in Fig. 4.1. In the following, an explanation of each move is presented.

Algorithm 4.3 Pseudo code for VND

Set of 3 neighborhood structures (N_j): Insert-minS, Swap, Replace
$j \leftarrow 1$;
while $j \leq 3$ **do**
  $S_2 \leftarrow$ Apply neighborhood structure N_j on S_1
if S_2 is better than S_1 **then**
  $S_1 \leftarrow S_2$;
  $j \leftarrow 1$;
else
  $j \leftarrow j + 1$;
end if
end while
return S_2

4.4. Insert-minS and Swap

The basic structure of these two moves are taken from Verbeeck et al. (2017) and adapted for the TDOPTW-STP. The pseudo codes for Insert-minS and Swap are shown in Algorithm 4.4 and Algorithm 4.5, respectively. Insert-minS tries to add a new vertex to a position (*pos*) in the current solution in the best improvement manner while Swap attempts to exchange two included vertices using the first improvement manner. Note that in Insert-minS, the minimum service time is used for each inserted vertex, and in Swap, the corresponding service time remains unchanged. For more details, we refer to Verbeeck et al. (2017).

Algorithm 4.4 Pseudo code for Insert-minS

for each non-included neighbor (vertex *i*) with minimum service time **do**
  for each position in the current solution (*pos*) **do**
    if Insert vertex *i* at *pos* is feasible **then**
      Calculate *NSR*;
    end if
  end for
end for
(*sel_i,sel_pos*) ← Determine the best (*i,pos*) with highest *NSR*
Execute Insert for (*sel_i,sel_pos*);
for the included vertex (exactly) before *sel_pos* **do**
  Recalculate tt_{ijdt};
end for
for each included vertex at and after *sel_pos* **do**
  Recalculate *at, wt, sst, dt,* tt_{ijdt};
end for
for each included vertex after *sel_pos* **do**
  Update *MaxShift*;
end for
for each included vertex at and before *sel_pos* **do**
  Calculate *MaxShift*;
end for

Algorithm 4.5 Pseudo code for Swap

```
1:  for each position in the current solution (pos1) do
2:    for each successor position in the current solution (pos2;pos2 > pos1) do
3:      if Swap vertices at pos1 and pos2 is feasible then
4:        Calculate dtime;
5:      if dtime < 0 then
6:        (sel_pos1,sel_pos2) ← Select pos1 and pos2;
7:        Break;
8:      end if
9:    end if
10: end for
11: if dtime < 0 then
12:     Break;
13:   end if
14: end for
15: Execute Swap for (sel_pos1, sel_pos2);
16: for included vertex (exactly) before sel_pos1 do
17:   Recalculate tt_ijdt;
18: end for
19: for each included vertex at and after sel_pos1 do
20:   Recalculate at, wt, sst, dt, tt_ijdt;
21: end for
22: for each included vertex after sel_pos2 do
23:   Update MaxShift;
24: end for
25: for each included vertex at and before sel_pos2 do
26:   Calculate MaxShift;
27: end for
```

After executing a move, it is required for some included vertices to recalculate their arrival time (at), start of service time (sst), waiting time (wt), departure time (dt), and travel time to successor (tt_{ijdt}). In the pseudo codes, this step is called "Recalculate" followed by the time parameter.

Vansteenwegen et al. (2009) defined *MaxShift* as "the maximum time that the service time completion of a given visit can be delayed, without making any visit infeasible". For TDOPTW, Verbeeck et al. (2017) proposed a backward algorithm for calculation of *MaxShift* which we call "Calculate *MaxShift*". Furthermore, they proposed the following formula to "Update *MaxShift*" for a specific vertex i.

$$MaxShift_i^{new} = MaxShift_i^{previous} + (dt_i^{previous} - dt_i^{new}) \quad (4\text{-}3)$$

MaxShift is being used in every local move, wherever an insertion is evaluated for feasibility. Keeping track of *MaxShift* allows to locally evaluate the impact of local search moves. This significantly reduces the required calculation time. In our algorithm for TDOPTW-STP, *MaxShift* of an included vertex is calculated without changing the service time of the succeeding vertices and indicates the time that is available for increasing its service time or inserting a new vertex into route, without turning the solution infeasible.

4.5. Replace

Algorithm 4.6 illustrates the pseudo code for Replace. This local search move first removes Nr vertices from the current solution, and then inserts as many vertices as possible in the solution, one by one. In fact, a combination of Nr removed vertices and a list of candidate vertices for insertion together with their best possible insertion position as well as their corresponding service time is called a feasible combination. The feasible combination with the highest *dprofit* will be executed as the best feasible combination. This process is called "Determine the feasible Replace" in this pseudo code and is presented in detail in Algorithm 4.7. After execution of the move, at, sst, wt, dt, tt_{ijdt} and *MaxShift* for all included vertices are recalculated. For each Nr from 1 to Nr_{max}, the Replace is performed based on the best improvement strategy where Nr_{max} is an input parameter of the algorithm and has to be predetermined.

Algorithm 4.6 Pseudo code for Replace

```
1:  for Nr = 1 to Nr_max do
2:    for pos1 = 2 to SolSize-Nr (in the current solution) do
3:      Determine the feasible Replace for (Nr,pos1);
4:      Calculate dprofit;
5:    end for
6:    sel_pos1 ← Determine the best (pos1) with highest dprofit > 0;
7:    Execute Replace for (Nr,sel_pos1);
8:  end for
9:  for each included vertex do
10:   Recalculate at, wt, sst, dt, tt_ijdt;
11:   Calculate MaxShift;
12: end for
```

In Algorithm 4.7, first, Nr consecutive included vertices are removed from a predetermined place within the current solution, and then, as many non-included vertices as possible are inserted, while the best service time combination is considered. Note that in this algorithm, when Nr vertices are removed, because of the time dependency, the solution feasibility is re-checked (line 8). Moreover, the way that a neighbor vertex is selected for insertion in Replace (Insert-varS; line 16) is different to the Insert-minS move in terms of considering the variable service time. The details of Insert-varS are shown in Algorithm 4.8.

Algorithm 4.7 Pseudo code for "Determine the feasible Replace" for (*Nr,pos1*)

```
1:  Remove Nr included vertices from pos1 to pos1 + Nr-1 in current solution;
2:  for included vertex (exactly) before pos1 do
3:    Recalculate tt_ijdt;
4:  end for
5:  for each included vertex at and after pos1 do
6:    Recalculate at, wt, sst, dt, tt_ijdt;
7:  end for
8:  if remove is feasible then
9:  for each included vertex after pos1 do
10:   Update MaxShift;
11: end for
12: for each included vertex at and before pos1 do
13:   Calculate MaxShift;
14: end for
15: while more improvement is possible do
16:   Insert-varS;
17:   end while
18: end if
```

In the Insert-varS (Algorithm 4.8), the service time for each candidate vertex is selected between its minimum and maximum possible values by *FindBestS* (Algorithm 4.9). For each solution, the insertion position and the selected vertex for insertion, it is first checked if insertion with the minimum service time is feasible, afterwards, an attempt is made to find the maximum feasible service time for the considered vertex.

Algorithm 4.9 shows the details on how service times are set when a vertex is considered for insertion. According to this algorithm, If vertex r is considered for insertion between vertices i and j, the *FindBestS* performs the following steps to find the best possible service time for vertex r.

a. Define at_j^1 as the latest possible arrival time to j, which equals $at_j + wt_j + MaxShift_j$ representing the arrival time, waiting time and *MaxShift* of j in the current solution, respectively. So, dt_r^1 is the latest departure time of r, calculated based on at_j^1 by using a backward calculation process (calculate_departure_time introduced by Verbeeck et al. (2017)). Note that if dt_r^1 is larger than $c_r + s_r^{min}$, $c_r + s_r^{min}$ is assigned to it.
b. It is checked if the insertion of vertex r with $s_r = s_r^{min}$ is feasible. If not, the process stops and the next vertex will be evaluated for insertion. Otherwise, it is checked if the insertion of vertex r with $s_r = s_r^{max}$ is feasible.

c. If the insertion of vertex r with $s_r = s_r^{max}$ is feasible, the s_r^{max} is selected as the service time of r and the process stops. Otherwise, to find the best feasible service time ($s_r^{min} \leq s_r < s_r^{max}$), the following steps are iteratively performed.

I. During this step, based on the insertion of r with service time s_r, first, the new arrival time of vertex j (at_j^2), and then, the new departure time (dt_r^2) are calculated. The change in the arrival time of the vertex j because of this insertion is called $shift_j$ (introduced by Vansteenwegen et al. (2009)). The $shift_j$ will be used in the feasibility check.

II. In this step, s_r is updated to $s_r + dt_r^1 - dt_r^2$. Note that, $dt_r^1 \leq dt_r^2$. If s_r is less than s_r^{min}, it is set equal to s_r^{min}. Then, with this s_r, the feasibility of insertion is checked again. These steps are repeated until a feasible s_r is found.

In addition, in order to clarify the procedure, the corresponding flowchart and a numerical example with arbitrary values can be found in Appendix B.

Algorithm 4.8 Pseudo code for Insert-varS

```
1: for each non-included neighbor vertex (i) do
2:    for each position in the current solution (pos) do
3:       (sel_st) ← Execute FindBestS to Determine the best feasible service time for vertex (i);
4:       Calculate NSR;
5:    end for
6:end for
7: (sel_i,sel_pos,sel_st) ← Determine the best (i,pos,sel_st) with highest NSR
8: Execute Insert for (sel_i,sel_pos,sel_st);
9: for included vertex (exactly) before sel_pos do
10:   Recalculate tt_ijdt;
11: end for
12: for each included vertex at and after sel_pos do
13:   Recalculate at, wt, sst, dt, tt_ijdt;
14: end for
15: for each included vertex after sel_pos do
16:   Update MaxShift;
17: end for
18: for each included vertex at and before sel_pos do
19:   Calculate MaxShift;
20: end for
```

Algorithm 4.9 Pseudo code for *FindBestS* (for Insert r between i and j)

```
1:  at_j^1 ← at_j + wt_j + MaxShift_j
2:  dt_r^1 ← calculate_departure_time (at_j^1)
3:  if (dt_r^1 > c_r + s_r^min) then
4:     dt_r^1 ← c_r + s_r^min
5:  end if
6:  if insert r with s_r = s_r^min is feasible then
7:     s_r ← s_r^max
8:  while insert r with s_r is not feasible then
9:     Calculate shift_j
10:    at_j^2 ← at_j + shift_j
11:    dt_r^2 ← calculate_departure_time (at_j^2)
12:    s_r ← s_r + dt_r^1 − dt_r^2
13:    If (s_r < s_r^min) then
14:    s_r ← s_r^min
15:    end if
16: end while
17: end if
```

4.5.1. Recentering

Each time the shaking and the VND steps are performed, the algorithm compares the current solution with the best solution obtained so far. If the current solution is better than the best found solution, the recentering phase is performed and the best solution is replaced by the current solution, and the *NoImprovement* is initialized to 0. Otherwise, *NoImprovement* is increased by 1.

5. Results and discussion

In this section, the performance of the proposed VNS is analyzed using various numerical experiments. Since no test instances are available for the TDOPTW-STP, in Section 5.1, two different sets of instances are generated based on the TDOPTW benchmark instances. In Section 5.2, first, the set of input parameters of the proposed algorithm are tuned, and then, the results of the VNS are evaluated over TDOPTW instances and the generated TDOPTW-STP instances with known optimal (best) solutions. In Section 5.3, a real-world data set from the city of Shiraz (Iran) is used to show the applicability and suitability of the proposed problem and solution method. All generated instances including Sets 1 and 2, as well as real data instance of Shiraz are available online at https://www.mech.kuleuven.be/en/cib/op/

5.1. Test instances

Verbeeck et al. (2017) created 36 benchmark instances with 20, 50 and 100 vertices for the TDOPTW with an optimal (best known) solution available at https://www.mech.kuleuven.be/en/cib/op/. In this section, two sets of instances are generated. In the first set (Set 1), TDOPTW instances with 20 vertices and 56 time slots are used. For these instances, opening times, closing times, the time dependent travel times, and time budget are retained. Then, to create the variable profit properties, for each vertex, the current service time and profit are used as the minimum service time (s_i^{min}) and the minimum profit (p_i^{min}). The maximum service time (s_i^{max}) and profit (p_i^{max}) are then generated by applying two random coefficients between 1 and 2 multiplied by the corresponding s_i^{min}, and p_i^{min}. Using this process, some small instances (up to 20 vertices and up to 56 time slots) are developed for the TDOPTW-STP and solved to optimality using the implemented mathematical model in CPLEX.

In the second set (Set 2), a procedure is designed to generate TDOPTW-STP instances based on TDOPTW instances so that their optimal (best known) solution becomes the optimal solution for the TDOPTW-STP. The pseudo code for this procedure is presented in Algorithm 5.1. In this procedure, in the first step, for each vertex, the deterministic service time, and profit of the vertex in the TDOPTW instance is set as the maximum of service time (s_i^{max}) and the maximum profit (p_i^{max}), respectively and then all vertices are divided into two parts: The ones included in the optimal (best known) sequence, and the non-included ones. The main idea is to first keep the optimal (best known) solution of the TDOPTW feasible for the TDOPTW-STP and then to force the rest of vertices in an unattractive situation (lower profit, higher minimum service time, and lower linear profit coefficient) where generating a new solution with higher profit would not be possible. It should be noted that, however, in the case of having only a best known solution for the TDOPTW (and not an optimal one) in hand, it is always possible to find a better sequence of vertices also for the TDOPTW-STP.

In the second step, for each included vertex, by selecting two random factors between 0 and 1 (*rand1* and *rand2*), and multiplying them with the corresponding s_i^{max}, and p_i^{max}, the minimum service time (s_i^{min}) and the minimum profit (p_i^{min}) is generated. Furthermore, for each vertex, the slope coefficient of the linear profit (a_i) is calculated by dividing the $p_i^{max} - p_i^{min}$ over $s_i^{max} - s_i^{min}$. Then, the minimum value of the set of the minimum profits over all vertices (*min_pmin*), the maximum value of the set of the maximum service times over all vertices (*max_smax*), and the minimum value of the set of the slope coefficient of the linear profits over all vertices (*min_a)* are calculated. At last, it is necessary to adapt the closing time of vertices for the ones included in the optimal (best known) solution. So, for each included vertex, the closing time of the vertex in the TDOPTW instance plus $s_i^{max} - s_i^{min}$ is set as the closing time of each included vertex. For example, for included vertex r, if $c_r = 20$, $s_r^{max} = 3$, and $s_r^{min} = 1$, closing time of vertex r will be 22 in Set 2. Note that due to constraint (3-29), if c_r would remain 20, the TDOPTW solution might become infeasible for TDOPTW-STP. So far, we make sure

that the optimal solution of the TDOPTW instance remains feasible for the TDOPTW-STP.

In the third step, for each non-included vertex, the minimum value of $\{ p_i^{max}, rand3_i * min\_pmin \}$ in which, *rand3 is* again a random factor between 0 and 1, is assigned to vertex i as its corresponding p_i^{max}. Then, the minimum profit (p_i^{min}) is calculated, as the result of a new random factor between 0 and 1 (*rand4*) multiplied by p_i^{max}. Furthermore, the maximum value of $\{ s_i^{max}, rand5_i * max\_smax \}$ in which, *rand5* is a new random factor greater than 1, is assigned to s_i^{min} that is greater than *max_smax* value. Then, by using another random factor between 0 and 1 (*rand6*), and multiplying it with the *min_a* value, the slope coefficient (a_i) is created in order that it stays lower than *min_a* value. Finally, according to the generated s_i^{min}, p_i^{max}, p_i^{min}, and a_i, the corresponding s_i^{max} is calculated.

Our motivation for this procedure is to keep the p_i^{max} and a_i values of non-included vertices lower than the *min_pmin* and the *min_a* values, and keeping the s_i^{min} value of non-included vertices higher than the *max_smax* value, and therefore to make the non-included vertices not interesting enough to be selected in the TDOPTW-STP optimal solution.

Algorithm 5.1 Pseudo code for generating Set 2

for each vertex i **do**
  $s_i^{max} \leftarrow s_i$;
  $p_i^{max} \leftarrow p_i$;
end for
Determine included vertices and non-included vertices in TDOPTW optimal (best known) solution;
for each included vertex i **do**
  Generate random numbers:
    $0 < rand1_i < 1$;
    $0 < rand2_i < 1$;
  Calculate:
    $s_i^{min} = rand1_i * s_i^{max} \rightarrow 0 < s_i^{min} < s_i^{max}$;
    $p_i^{min} = rand2_i * p_i^{max} \rightarrow 0 < p_i^{min} < p_i^{max}$;
    $a_i = (p_i^{max} - p_i^{min}) / (s_i^{max} - s_i^{min})$;
    $c_i \leftarrow c_i + s_i^{max} - s_i^{min}$;
end for
for all included vertices **do**
  Calculate:
    $max\_smax = \max \{ s_i^{max} \}$;
    $min\_pmin = \min \{ p_i^{min} \}$;
    $min\_a = \min \{ a_i \}$;
end for
for each non-included vertex i **do**
  Generate random numbers:
    $0 < rand3_i < 1$;
    $0 < rand4_i < 1$;
    $rand5_i > 1$;
    $0 < rand6_i < 1$;
  Calculate:
    $p_i^{max} = \min \{ p_i^{max}, rand3_i * min\_pmin \} \rightarrow 0 < p_i^{max} < min\_pmin$;
    $p_i^{min} = rand4_i * p_i^{max} \rightarrow 0 < p_i^{min} < p_i^{max}$;
    $s^{min}{}_i = \max \{ s^{max}{}_i, rand5_i * max\_smax \} \rightarrow max\_smax < s_i^{min}$;
    $a_i = rand6_i * min\_a \rightarrow 0 < a_i < min\_a$;
    $s_i^{max} = s_i^{min} + \lceil (p_i^{max} - p_i^{min}) / a_i \rceil$;
end for

5.2. Numerical experiments

The proposed algorithm is implemented in Visual C++ 2010 and the experiments are performed using a laptop with Intel Core i7 CPU M 640 @ 2.80 GHz processor and 8.00 GB Ram. In the proposed algorithm, there exist 2 input parameters for tuning: *MaxNoImprovement* and Nr_{max}. In order to determine these parameters, a number of primary tests are done. Firstly, 14 instances are randomly selected from Sets 1 and 2 of the TDOPTW-STP, and for each of the parameters, three different values are determined ($MaxNoImprovement = n/3, n/4, n/5$ and $Nr_{max} = 1, 2, 3$). Then, the algorithm is applied on the selected set for all 9 possible combinations of different values of parameters, and the appropriate combination is chosen based on a balance between solution quality and the computation time. As a result, the parameters are set as follow: $MaxNoImprovement = n/5$ and $Nr_{max} = 2$.

Table 5.1
Detailed results of solution algorithms on TDOPTW set.

Instance	ACS		VNS		
	Profit	CPU (Sec)	Profit	Profit gap (%)	CPU (Sec)
20.1.1	159	0.1	159	0.0	0.01
20.1.2	173	0.1	173	0.0	0.01
20.1.3	184	0.1	184	0.0	0.01
20.2.1	188	0.1	188	0.0	0.02
20.2.2	201	0.1	195	3.0	0.01
20.2.3	195	0.1	179	8.2	0.01
20.3.1	277	0.1	253	8.7	0.01
20.3.2	246	0.1	246	0.0	0.03
20.3.3	259	0.1	259	0.0	0.04
20.4.1	274	0.1	262	4.4	0.02
20.4.2	275	0.2	258	6.2	0.02
20.4.3	268	0.1	255	4.9	0.01
50.1.1	288	0.2	279	3.1	0.04
50.1.2	274	0.2	274	0.0	0.05
50.1.3	289	0.3	289	0.0	0.05
50.2.1	298	0.3	298	0.0	0.10
50.2.2	310	0.3	291	6.1	0.13
50.2.3	340	0.4	340	0.0	0.13
50.3.1	339	0.3	326	3.8	0.11
50.3.2	404	0.4	374	7.4	0.09
50.3.3	366	0.5	362	1.1	0.29
50.4.1	476.6	0.5	478	−0.3	0.26
50.4.2	439.8	0.6	429	2.5	0.13
50.4.3	450	0.6	431	4.2	0.19
100.1.1	275	0.4	258	6.2	0.19
100.1.2	278	0.4	276	0.7	0.19
100.1.3	343	0.6	335	2.3	0.43
100.2.1	351.2	0.5	349	0.6	0.55
100.2.2	366.6	0.5	343	6.4	0.28
100.2.3	370	0.6	359	3.0	0.77
100.3.1	436	0.7	417	4.4	0.73
100.3.2	446.6	0.7	437	2.1	1.02
100.3.3	467	0.9	462	1.1	1.34
100.4.1	484	1.0	480	0.8	1.09
100.4.2	494.6	0.9	495	−0.1	1.03
100.4.3	526.8	1.0	526	0.2	1.37
Max		**1.0**		**8.7**	**1.4**
Avg		**0.4**		**2.5**	**0.3**

The first set of experiments is performed on the TDOPTW set of Verbeeck et al. (2017). Our proposed method is compared with the results of the Ant Colony System (ACS) by Verbeeck et al. (2017). The results are displayed in Table 5.1. In this table, the first column gives the instance names. The second and third columns show the results of the ACS. Contrary to our VNS, the ACS contains randomness. Therefore, we run our method only once, and compare the results to the average results of the ACS. Moreover, to have a fare comparison of CPU times, the ACS code is rerun on the same system configuration as our VNS is run. Thus, due to the existing randomness as well as improvement in the ACS, the presented results in this table are slightly different from the ones presented in Verbeeck et al. (2017). The fourth to sixth columns show the results of the proposed VNS.

The profit gap is presented as 100* (ACS profit - VNS profit) / ACS profit, and the CPU shows the computation time in seconds. Note that for running the VNS on this set, we set $s_i^{max} = s_i^{min} = s_i$ and $p_i^{max} = p_i^{min} = p_i$.

It should be noted that, the VNS is designed to solve the TDOPTW problem when the service time is not fixed and the profit depends on the duration of stay at each vertex (TDOPTW-STP), and, our algorithm is tuned for the TDOPTW-STP. However, our aim is to compare the proposed method over the TDOPTW problem to show that the performance of the proposed VNS is acceptable for this problem.

Looking at the results, it can be seen that in total, for 36 instances of TDOPTW, in 10 instances, the gap is zero. The VNS could improve the best known results in 2 instances, and in 24 instances the ACS gives better results. Furthermore, the average and maximum gaps are 2.5 and

Table 5.2
Detailed results of solution algorithms on Set 1.

Instance	CPLEX		VNS		
	Profit	CPU (Sec)	Profit	Profit gap (%)	CPU (Sec)
20.1.1	215.2	9534	211.1	1.91	0.05
20.1.2	221.9	7321	221.9	0.00	0.02
20.1.3	236.3	27,479	236.3	0.00	0.02
20.2.1	239.8	47,961	237.1	1.13	0.02
20.2.2	258.8	25,405	258.3	0.19	0.02
20.2.3	240.1	99,956	240.1	0.00	0.03
Max		**99,956**		**1.91**	**0.05**
Ave		**36,276**		**0.54**	**0.03**

Table 5.3
Detailed results of solution algorithms on Set 2.

Instance	Best known Solution	VNS		
	Profit	Profit	Profit gap (%)	CPU (Sec)
20.1.1	159	159.0	0.00	0.02
20.1.2	173	173.0	0.00	0.01
20.1.3	184	184.0	0.00	0.02
20.2.1	188	188.0	0.00	0.02
20.2.2	201	201.0	0.00	0.02
20.2.3	195	195.0	0.00	0.03
20.3.1	277	277.0	0.00	0.06
20.3.2	246	246.0	0.00	0.03
20.3.3	259	259.0	0.00	0.03
20.4.1	274	274.0	0.00	0.12
20.4.2	275	275.0	0.00	0.04
20.4.3	268	268.0	0.00	0.10
50.1.1	288	288.0	0.00	0.12
50.1.2	274	274.0	0.00	0.24
50.1.3	289	289.0	0.00	0.11
50.2.1	298	298.0	0.00	0.21
50.2.2	310	310.0	0.00	0.09
50.2.3	340	340.0	0.00	0.34
50.3.1	339	332.7	1.86	0.18
50.3.2	404	404.0	0.00	0.21
50.3.3	366	362.0	1.09	0.36
50.4.1	476.6	473.8	0.59	0.73
50.4.2	439.8	434.0	1.32	0.37
50.4.3	450	450.0	0.00	0.38
100.1.1	275	275.0	0.00	0.49
100.1.2	278	276.0	0.72	0.38
100.1.3	343	343.0	0.00	1.04
100.2.1	351.2	351.0	0.06	1.18
100.2.2	366.6	362.8	1.04	1.52
100.2.3	370	370.0	0.00	1.19
100.3.1	436	437.0	−0.23	1.74
100.3.2	446.6	454.0	−1.66	1.58
100.3.3	467	470.0	−0.64	1.57
100.4.1	484	484.0	0.00	2.31
100.4.2	494.6	497.0	−0.49	1.52
100.4.3	526.8	540.0	−2.51	3.30
Max			**1.86**	**3.30**
Ave			**0.03**	**0.60**

Table 5.4
% Gap and CPU time per dataset in Set 2.

n	Max profit gap (%)	Ave profit gap (%)	Max CPU (Sec)	Ave CPU (Sec)
20	0.00	0.00	0.12	0.04
50	1.86	0.40	0.73	0.28
100	1.04	−0.31	3.30	1.48

8.7% and the average and maximum computation times are 0.30 and 1.4 s.

In the next set of experiments, our algorithm is applied on Set 1 of the new test instances (introduced in Section 5.1). Results are presented in Table 5.2. For Set 1, only 6 instances were solved optimally using CPLEX 12.8 and the profit gap (%) is defined as "100*(CPLEX profit - VNS profit) / CPLEX profit".

The results display the correctness of the proposed mathematical model, and the immoderate time required to find optimal solutions for these small instances using a commercial solver. By applying the VNS on these 6 instances of Set 1, the optimal solution is found in 3 instances. In addition, the average gap is only 0.54 % and the average computation time is 0.03 s.

In the third step, our algorithm is applied on Set 2 of the test instances. Results are presented in Tables 5.3 and 5.4. Similar to Table 5.1, for Set 2, the results are compared to best known solutions of Verbeeck et al. (2017). However, the difference is that, here the service time is not fixed. In Table 5.4, the Gap and the CPU time of the results of Set 2 are presented per instance size.

For Set 2, in 24 out of 36 instances, the best known solution is found, and the best known results are improved in 5 instances. The average gap and the maximum gap are only 0.03 % and 1.86 %, respectively. These results prove the high performance quality of the proposed VNS. In addition, the average and the maximum computation times (CPU) are 0.60 and 3.30 s which show that the proposed VNS is fast enough for most application purposes. When comparing the average results of different sizes of instances in Set 2, it can be seen that for all instances with 20 vertices, the best known solution is found, the maximum profit gap (%) belongs to the instances with 50 vertices, and the maximum CPU time belongs to instances with 100 vertices. We may conclude that when the size of instances are getting larger in terms of the number of vertices, the VNS is still able to find high-quality solutions.

5.3. *Real-world instance*

The purpose of this section is to create a real instance for the TDOPTW-STP, to perform a sensitivity analysis to show the potential use of the model in a real application, and to demonstrate the importance of variable service times, leading to higher profits than any kind of prefixed service times (min, average or max). Moreover, it shows that in case of congested (road) networks, our formulation and solution approach is required to obtain high quality solutions. Because of the various tourist attractions, the city of Shiraz with approximately 240 km^2 area and 1.6 million population, is one of the top three tourist destinations in Iran for both domestic and international tourists. In this section, a real data set is created based on the urban road network of Shiraz.

5.3.1. *Data generation*

In a real TDOPTW-STP instance, travel times between POIs are variable and rely on the time of the day. Each POI has a time window and service time dependent profit. The profit is defined as a linear function of the service time and has lower and upper bounds. Which means, that at each POI, the minimum visit time should be spent to gain the minimum profit. After that, the longer the visit time, the higher the profit collected by the tourist, so that the profit increases with a constant slope (profit per unit time) until it reaches its maximum value. The required information to make the TDOPTW-STP instance can be divided into three general categories; specifications of POIs (including location, time window, and maximum and minimum service times), network information (including time dependent travel times between origin–destination pairs during the day) and tourist considerations (including tour time budget, and maximum and minimum profits of POIs). Each of these would be described below.

5.3.1.1. POI data. One hotel (as the start and the end depot) and 38 POIs in the city are selected, and their coordinates (latitude and longitude) are extracted using GoogleMaps. Geographical distribution of all vertices (POIs and the hotel) as well as major streets of the city are shown in Fig. 5.1. In this study, the Royal hotel located in Abolkalam square is considered as both the start and the end depot (vertex number

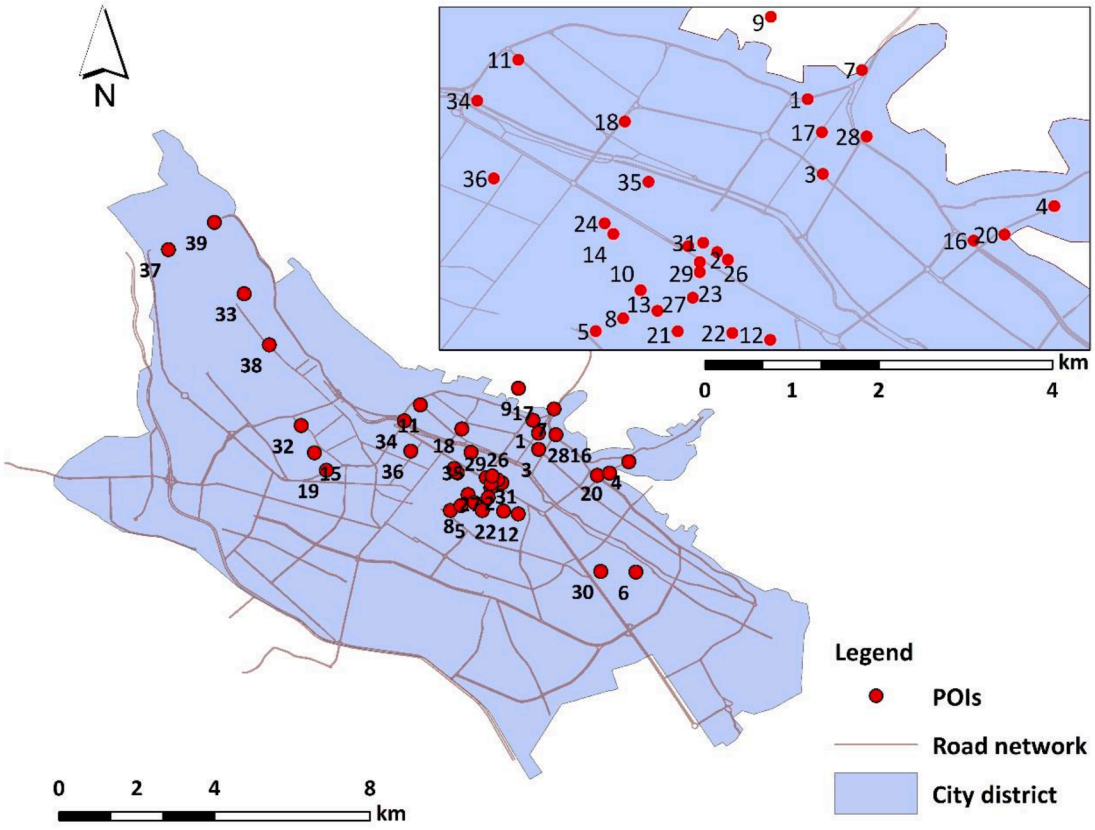

Fig. 5.1. Selected POIs in the city of Shiraz.

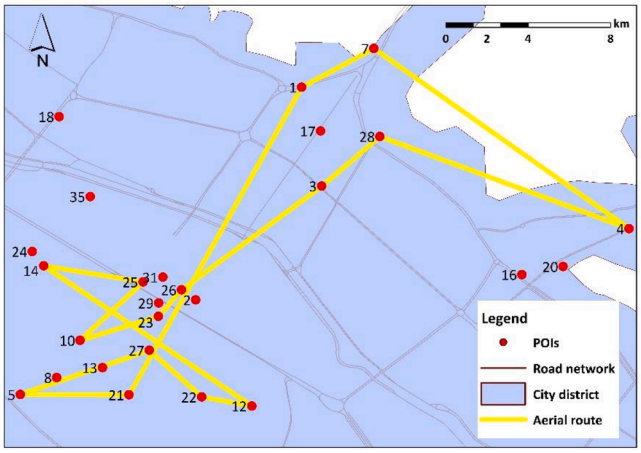

Fig. 5.2. The sequence of visits for Sce 0.

1 in Fig. 5.1). The opening time of the depot is set equal to 6:00 and its profit is considered zero. For other places, the start time is between 6:00 and 15:00, the end time is between 15:00 and 23:00. Information and comments in local tourism websites have been used to collect these real data. According to this information, the maximum service times are between 30 and 90 min. Then, the minimum service time for each vertex is generated by multiplying a random percentage between 60 and 80 % by its maximum value. Properties of POIs in this instance can be found in Appendix C.

5.3.1.2. Network data. Travel times of the road network vary during the day depending on the traffic volume, congestion, and delay, and generally increases during peak periods and decreases during off-peak periods. To develop the real time dependent travel times on the traffic network of Shiraz, the day time between 6 am and 10 pm is divided into 65 time slots of 15 min, and for each time slot, a travel time matrix with dimensions of 39 by 39 is created. The details on how to produce these matrices are explained in Appendix D.

5.3.1.3. Tourist considerations. It should be noted that, in the tourist trip

Table 5.5
Detailed program for Sce 0.

Item	Unit	Start depot	Other included vertices														End depot
vertex	-	hotel	21	5	27	22	12	14	25	10	23	26	3	28	4	7	hotel
at	hr	6.0	6.1	7.2	8.0	8.6	9.5	11.1	12.3	13.5	13.9	15.2	15.8	17.1	17.7	19.1	20.0
wt	hr	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
sst	hr	6.0	6.1	7.2	8.0	8.6	9.5	11.1	12.3	13.5	13.9	15.2	15.8	17.1	17.7	19.1	20.0
s	min	0.0	61.0	40.0	31.3	53.0	85.0	65.0	65.0	21.0	75.0	30.0	70.0	32.0	70.0	55.0	0.0
dt	hr	6.0	7.2	7.9	8.5	9.5	10.9	12.2	13.4	13.8	15.2	15.7	17.0	17.6	18.9	20.0	20.0
tt	min	8.1	4.2	6.8	3.0	2.7	13.4	5.1	5.3	6.2	1.5	7.8	4.1	6.5	10.2	1.9	–

at: arrival time, wt: waiting time, sst: start of service time, s: service time, dt: departure time, tt: travel time.

Table 5.6
Results of the solution algorithm on all scenarios.

Scenario	Problem	Profit	# of visited POIs	Tour time (hr)	CPU (sec)	Imp (%)	Div (%)
Sce 0	TDOPTW-STP	274.025	14	14.000	0.915	–	–
Sce 1-Max	TDOPTW	270.000	12	13.963	0.109	1.47	19.23
Sce 1-Ave	TDOPTW	259.000	14	13.920	0.308	5.48	53.57
Sce 1-Min	TDOPTW	245.000	16	13.919	0.123	10.59	46.67
Sce 2	TDOP-STP	277.276	14	13.566	0.446	−1.19	25.00
Sce 3	OPTW-STP	266.248	13	13.985	0.371	2.84	37.04
Sce 4	TDOPTW-STP	236.534	13	11.452	0.341	13.68	33.33
Sce 5	TDOPTW-STP	275.707	15	14.000	0.450	−0.61	44.83

Table 5.7
Sensitivity analyses of tour efficiency.

Scenario	Problem	Profit	Tour time (hr)	Useful time (hr)	Useful time/time budget
Sce 1-Max	TDOPTW	270	13.963	12.500	89.3%
Sce 0	TDOPTW-STP	274.025	14	12.554	89.7%
Sce 1-Max-3	TDOPTW	226	13.6739	10.500	75.0%
Sce 0–3	TDOPTW-STP	233.516	14	10.798	77.1%

planning, the aim is to provide a personalized plan for a specific tourist according to his/her known preferences. Therefore, tourist considerations including the time budget and the minimum and maximum profits for each POI depend on the personal preferences of that tourist. In this study, we consider an arbitrary tourist, and related tourist data are generated as follows.

According to Eqs. (3-1) to (3-3), for each POI, the profit function is generated by estimating four values of minimum and maximum service time and profit. As explained earlier, minimum and maximum service times are properties of the POIs. However, maximum and minimum profits that can be obtained from visiting a POI depends on the personal preferences of the tourist, and in practice can be directly asked from him/her. An efficient way of finding these profits based on user preferences is itself an ongoing research subject (Tarantino et al., 2019), and is beyond the scope of this paper. In this research, preferences are generated based on an average tourist behavior taken from average visitors' ratings provided by Google Maps. The Google Maps rating is translated to maximum profit of each POI using a similar strategy as explained in Jandaghi et al. (2021), and normalized between 10 and 30. The minimum profit for each vertex is generated by multiplying a random number between 0.65 and 0.75 by its maximum value. The time budget is assumed to be equal to 14 h. So, based on $t^{max} = c_n - o_1$, The closing time of hotel (depot) is set equal to 20:00.

5.3.2. Experiments and analysis

To evaluate the performance of the proposed model and the solution method, a number of scenarios are created. These scenarios are explained in this section. First, the above-explained generated TDOPTW-STP instance is called the base scenario (Sce 0). The proposed solution algorithm is implemented on this base instance (Sce 0) and the result is shown in Fig. 5.2. In this figure, the presented sequence of visits does not necessarily show the selected road routes. This is because in practice the selected road is affected by access restrictions due to the directions of road, geometric design of roads and intersections, and traffic congestion. Other details of the proposed tour are presented in Table 5.5. According to the suggested plan, out of 38 possible POIs, the tourist may visit 14 POIs gaining 274.025 as the profit during 14 h. Note that out of 14 visited POIs, 9 POIs with maximum service time, three POIs with minimum service time and two POIs with a service time between maximum and minimum service times have been suggested.

In order to analyze the sensitivity of the solution algorithm to the input data, five other scenarios are generated based on the base scenario (Sce i, $i \in \{1, \cdots 5\}$). In Sce 1-Max, Sce 1-Ave and Sce 1-Min (TDOPTW), the service time and the profit of POIs are fixed and respectively equal to their maximum, average and minimum values in TDOPTW-STP. In Sce 2, the impact of time windows of all POIs are removed by setting them from 6 am to 11 pm (TDOP-STP). In Sce 3, the travel times between vertices are fixed (OPTW-STP). In this scenario, we set all travel times equal to the travel time at 12:00 (one of the peak periods of traffic). In Sce 4, the time budget is reduced from 14 h to 12 h, and, in Sce 5, the start time of the daily tour (opening time of the start depot) has changed from 6 am to 7.5 am. The proposed VNS is used for all these scenarios and the results are summarized in Table 5.6. In the last two columns, two criteria (Imp and Div) are defined to compare the results of scenarios. Imp shows the percentage improvement in total profit when the TDOPTW-STP is used versus other problem variants. It is calculated as 100* (profit of Sce 0 - profit of Sce x) / profit of Sce x, where Sce x is the considered scenario number. The second index, Div shows the diversity between two different solutions. Div index between two solutions A and B is calculated as the sum of the number of vertices in A not present in B and the number of vertices in B not present in A, and the number of same vertices in both solutions with different service times, divided by the total number of vertices present in A and B.

One main comparison which shows the effectiveness of having variable service times in such tourist application is when we compare the TDOPTW-STP against its fixed service time variant (TDOPTW). In Sce 1, three versions of service times and the corresponding profits are considered. Compared to the base scenario (Sce 0), the profit is decreased by 1.47% in Sce 1-Max, 5.48% In Sce 1-Ave, and 10.59% in Sce1-Min. For this instance, these comparisons clearly illustrate, the added value of considering a TDOPTW-STP, where service times can be optimized, instead of a regular TDOPTW. Moreover, the presented values in Div column shows that the TDOPTW-STP solution is significantly different versus the TDOPTW solutions.

In Sce 2 (TDOP-STP), when time windows are widened, included vertices and their sequence and service times have changed. Due to the flexibility in time windows, higher total profit is gained in lower total tour time. As expected, less binding time windows lead to a higher quality solution. In Sce 3 (OPTW-STP), travel times during the day are assumed as fixed and time independent to show the higher impact of service time dependent profit in the presence of time dependent travel times. Thus, when compared with the Sce 0, the algorithm is not flexible in selecting more appropriate travel times, thereby, less profits are collected. Note that in this scenario, the number of time slots is equal to one, while in other instances, it is equal to 65. As a result of ignoring the time dependency, the solution has around 3% lower quality with around 37% different vertices/service times. In Sce 4, by decreasing the time budget, the total profit is reduced by around 14%, while the solution is around 33% different to the solution of Sce 0.

In Sce 5, the start time of the tour has been changed from 6 am to 7.5

am, and accordingly, the opening time of the hotel has been adapted to 7.5 am. In other words, in this scenario, the tour starts later compared to Sce 0. Despite this change in the start time of tour, the total profit of Sce 5 is slightly more than that of Sce 0. This is because most of the POIs are closed between 6 and 7.5 in the morning, and therefore starting the tour 1.5 h later gives the opportunity to use the time budget more efficiently. Comparing these 8 scenarios, the minimum CPU is equal to 0.109 s for Sce 1-Max (TDOPTW) and the maximum CPU is equal to 0.915 s for Sce 0 (TDOPTW-STP) which is due to the change in the size of the search space. According to this analysis, the quality and the computational time of the proposed method seems appropriate to be used in tourist websites and mobile applications.

Summing up, the conclusion of these experiments is twofold. Firstly, having time dependency and service time dependent profits together is the more appropriate way of modeling the tourist trip planning problem, and secondly, the proposed VNS works correctly and effectively in practice, as can be seen from the obtained solutions in different scenarios. Another relevant and important analyses, is to examine the situations where the TDOPTW-STP is more effective for the tourist trip planning in practice. For a tourist, in a proposed route, travel times and waiting times are counted as useless times, and the unused time budget is considered as lost time. On the other hand, the sum of service times is defined as the useful time. Thus, if it is possible to enlarge the share of useful time in the time budget, the efficiency of the proposed tour will be increased. Therefore, at this point, further investigations have been performed to show the importance of adding the service time dependent profit into the TDOPTW, specifically when routes are more congested. In a more congested traffic network, we expect a lower ratio of useful time over the time budget, and therefore being more flexible in selecting a proper service time would be more beneficial.

In these experiments, all travel times of the Shiraz instance are multiplied by 3 (and checked to satisfy the FIFO condition) to have a more congested network. Then, the new instance is solved in two scenarios: i) when profits are fixed using the corresponding maximum values (Sce 1-Max-3), and ii) when profits are service time dependent (Sce 0–3). Table 5.7 presents the results summary. In this table, the first two rows correspond to the base scenario and the Sce 1-Max which are explained earlier in this section. The next two rows show the results of the same scenarios when travel times are tripled.

In Table 5.7, For each scenario, the profit shows the total profit of the proposed tour; The tour time shows the total time of the tour including the visit times, travel times and waiting times; The useful time is the total time used only for visiting POIs; and the last column shows the ratio of useful time over time budget as an indication of the tour efficiency.

When we compare the Sce 0 to Sce 1-Max, the ratio of useful time/time budget is increased from 89.3% to 89.7% and the profit is increased from 270 to 274.025. Moreover, compare Sce 0–3 to Sce 1-Max-3, this ratio is increased from 75.0% to 77.1% and profit is increased from 226 to 233.516. Note that an increase in the profit does not necessarily mean the growth in the useful time/time budget ratio.

Based on these results, we conclude that the increase in profit when we compare the TDOPTW-STP with the TDOPTW, is more noticeable when the network is congested. These results show that the TDOPTW-STP gives a higher chance for a better time budget management to the tourist.

6. Conclusion

In this paper, the time dependent orienteering problem with time windows and service time dependent profit is introduced. In the TDOPTW-STP, the profit of visiting a vertex depends on the duration of the visit, and this characteristic is added to the time dependent variant of the OPTW, which makes it even more complex to solve. This complex problem has interesting applications in practice, including personalized tourist trip planning. To solve the TDOPTW-STP, a variable neighborhood search is proposed, which uses three local search moves of Insert, Swap, and Replace, adapted to find proper service times together with a good order of visits.

First, the proposed method is compared with the results of the Ant Colony System (ACS) by Verbeeck et al. (2017) for the TDOPTW instances. The average and maximum gaps are 2.5 and 8.7% and the average and maximum computation times are 0.30 and 1.4 s. Next, 6 new instances of TDOPTW-STP are generated and solved using the proposed mathematical model (Set 1). Results are compared with the VNS on these instances. The average gap is only 0.54 % and the average computation time is 0.03 s. In the next step, new TDOPTW-STP instances with known optimal (best) solutions are generated based on the best known solutions of Verbeeck et al. (2017). For this set (Set 2), in 29 out of 36 instances, the best known solution is found or improved. In addition, the average and maximum gaps are only 0.03 % and 1.86 %, respectively. Furthermore, a real data set is created over the city of Shiraz (in Iran) with 39 vertices. Using 6 different scenarios, it is demonstrated that the proposed algorithm is able to obtain high-quality solutions in real-time. This analysis also clearly shows the significant impact of the service time dependent profit property, especially in the presence of time dependency and time windows. Moreover, an extra analysis proved a better time budget management for the tourist when the TDOPTW-STP is used in practice.

For future extensions, a major challenge would be to deal with other types of profit function including a nonlinear one. Another interesting extension would be to address the discrete service time version of the problem, similar to Erdoğan and Laporte (2013), where multiple visits to each vertex are allowed. Further research could focus on the TDOPTW-STP with multiple routes. Several vertices may also have different profit functions which depend on certain factors such as weather conditions or that are closed on certain days. For example, parks have a higher profit when the sun is shining than when it is raining. Furthermore, our problem could be extended by adding mandatory vertices (such as 'must-see POIs' or mandatory customers) or a lunch break. This break has no fixed location or exact timing. In these extensions, some additional constraints should be added and the complexity of the problem will be significantly increased. Lastly, in this paper, we have ignored that the departure time of the start depot could be a decision variable. Therefore, an interesting challenge would be to solve a TDOPTW-STP in which the departure time from the start depot is an additional decision variable. These proposed extensions of the problem will help to model more realistic situations in tourism, logistics, and other possible applications of the problem.

CRediT authorship contribution statement

M. Khodadadian: Conceptualization, Writing – original draft, Software, Investigation, Investigation, Formal analysis. **A. Divsalar:** Conceptualization, Methodology, Validation, Supervision, Writing – review & editing. **C. Verbeeck:** Conceptualization, Validation, Writing – review & editing. **A. Gunawan:** Validation, Writing – review & editing. **P. Vansteenwegen:** Validation, Writing – review & editing.

Acknowledgement

We would like to thank Dr. Reza Golshan Khavas for providing the code for collecting online travel time information of the city of Shiraz.

Appendix A. Preliminary experiment for sorting neighbors in the Initialization phase of the solution algorithm

See Tables A.1 and A.2.

Table A.1
Sorting states.

State	Sorting criterion
1	without sorting
2	$p_j^{min}/\left(s_j^{min}+tt_{ij}^{min}\right)$
3	$p_j^{Ave}/\left(s_j^{Ave}+tt_{ij}\right)$
4	$p_j^{max}/\left(s_j^{max}+tt_{ij}^{min}\right)$

Table A.2
Detailed results of solution algorithm on Set 1 and Set 2 for each sorting state.

Set	Instance	state 1		state 2		state 3		state 4	
		Profit	CPU (Sec)	Profit	CPU (Sec)	Profit	CPU (Sec)	Profit	CPU (Sec)
1	20.1.1	211.06	0.04	211.06	0.05	211.06	0.04	211.06	0.04
1	20.1.2	221.91	0.02	221.91	0.02	221.91	0.02	221.91	0.02
1	20.1.3	236.28	0.02	236.28	0.02	236.28	0.02	236.28	0.02
1	20.2.1	237.12	0.02	237.12	0.02	237.12	0.02	237.12	0.02
1	20.2.2	258.26	0.02	258.26	0.02	258.26	0.02	258.26	0.02
1	20.2.3	240.06	0.03	240.06	0.03	240.06	0.03	240.06	0.03
2	20.1.1	159.00	0.02	159.00	0.02	159.00	0.02	159.00	0.02
2	20.1.2	173.00	0.01	173.00	0.01	173.00	0.02	173.00	0.02
2	20.1.3	184.00	0.02	184.00	0.02	184.00	0.02	184.00	0.02
2	20.2.1	188.00	0.02	188.00	0.02	188.00	0.02	188.00	0.02
2	20.2.2	201.00	0.02	201.00	0.02	201.00	0.02	201.00	0.02
2	20.2.3	195.00	0.04	195.00	0.04	195.00	0.03	195.00	0.04
2	20.3.1	277.00	0.07	277.00	0.06	277.00	0.07	277.00	0.06
2	20.3.2	246.00	0.03	246.00	0.03	246.00	0.03	246.00	0.03
2	20.3.3	259.00	0.04	259.00	0.03	259.00	0.03	259.00	0.04
2	20.4.1	274.00	0.13	274.00	0.13	274.00	0.13	274.00	0.13
2	20.4.2	275.00	0.05	275.00	0.04	275.00	0.04	275.00	0.05
2	20.4.3	268.00	0.11	268.00	0.11	268.00	0.11	268.00	0.11
2	50.1.1	288.00	0.12	288.00	0.12	288.00	0.12	288.00	0.12
2	50.1.2	274.00	0.11	274.00	0.11	274.00	0.11	274.00	0.10
2	50.1.3	289.00	0.10	289.00	0.10	289.00	0.11	289.00	0.25
2	50.2.1	298.00	0.21	298.00	0.21	298.00	0.21	298.00	0.29
2	50.2.2	310.00	0.10	310.00	0.12	310.00	0.12	310.00	0.10
2	50.2.3	340.00	0.35	340.00	0.33	340.00	0.34	340.00	0.34
2	50.3.1	332.74	0.17	332.74	0.17	332.74	0.17	332.74	0.17
2	50.3.2	404.00	0.22	404.00	0.21	404.00	0.21	404.00	0.22
2	50.3.3	362.00	0.33	362.00	0.33	362.00	0.33	362.00	0.34
2	50.4.1	473.83	0.74	473.83	0.74	473.83	0.74	473.83	0.77
2	50.4.2	434.00	0.32	434.00	0.33	434.00	0.33	434.00	0.32
2	50.4.3	450.00	0.36	450.00	0.37	450.00	0.36	450.00	0.39
2	100.1.1	275.00	0.48	275.00	0.48	275.00	0.48	275.00	0.50
2	100.1.2	276.00	0.36	276.00	0.36	276.00	0.37	276.00	0.37
2	100.1.3	343.00	0.99	343.00	0.97	343.00	0.96	343.00	1.02
2	100.2.1	351.00	1.12	351.00	1.14	351.00	1.13	351.00	1.13
2	100.2.2	362.81	1.10	362.81	1.11	362.81	1.11	362.81	1.15
2	100.2.3	370.00	1.08	370.00	1.08	370.00	1.11	370.00	1.09
2	100.3.1	437.00	1.70	437.00	1.71	437.00	1.71	437.00	1.76
2	100.3.2	454.00	1.54	454.00	1.53	454.00	1.54	454.00	1.57
2	100.3.3	470.00	1.57	470.00	1.58	470.00	1.59	470.00	1.60
2	100.4.1	484.00	2.33	484.00	2.35	484.00	2.35	484.00	2.37
2	100.4.2	497.00	1.53	497.00	1.53	497.00	1.53	497.00	1.55
2	100.4.3	540.00	3.33	540.00	3.41	540.00	3.37	540.00	3.37
Min		**159.00**	**0.01**	**159.00**	**0.01**	**159.00**	**0.02**	**159.00**	**0.02**
Ave		**314.74**	**0.50**	**314.74**	**0.50**	**314.74**	**0.50**	**314.74**	**0.51**
Max		**540.00**	**3.33**	**540.00**	**3.41**	**540.00**	**3.37**	**540.00**	**3.37**

Appendix B. The flowchart and a numerical example of *FindBestS* (for insertion of vertex r between vertex i and vertex j)

See Fig. B.1 and Fig. B.2.

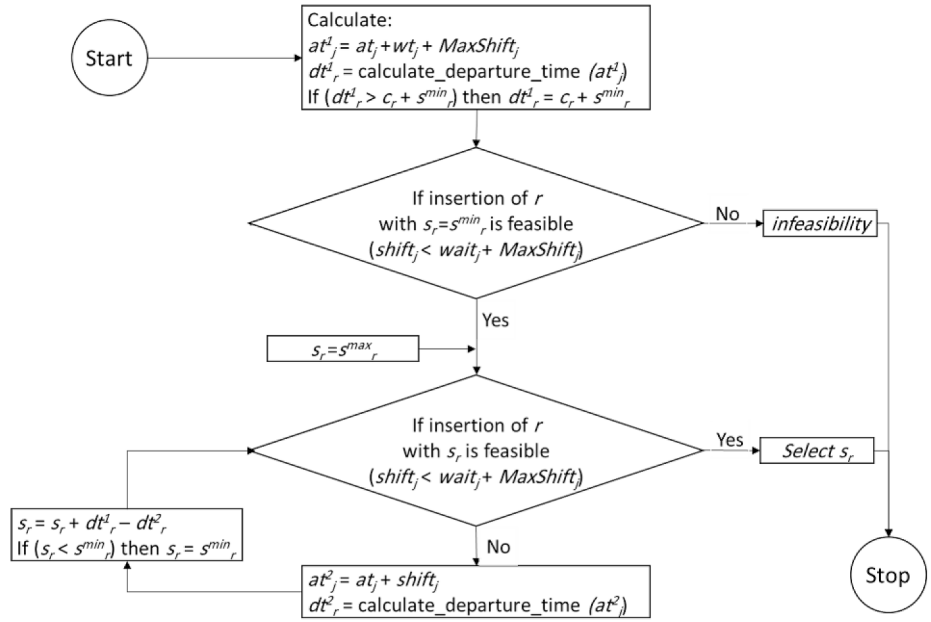

Fig. B.1. Flowchart of *FindBestS*.

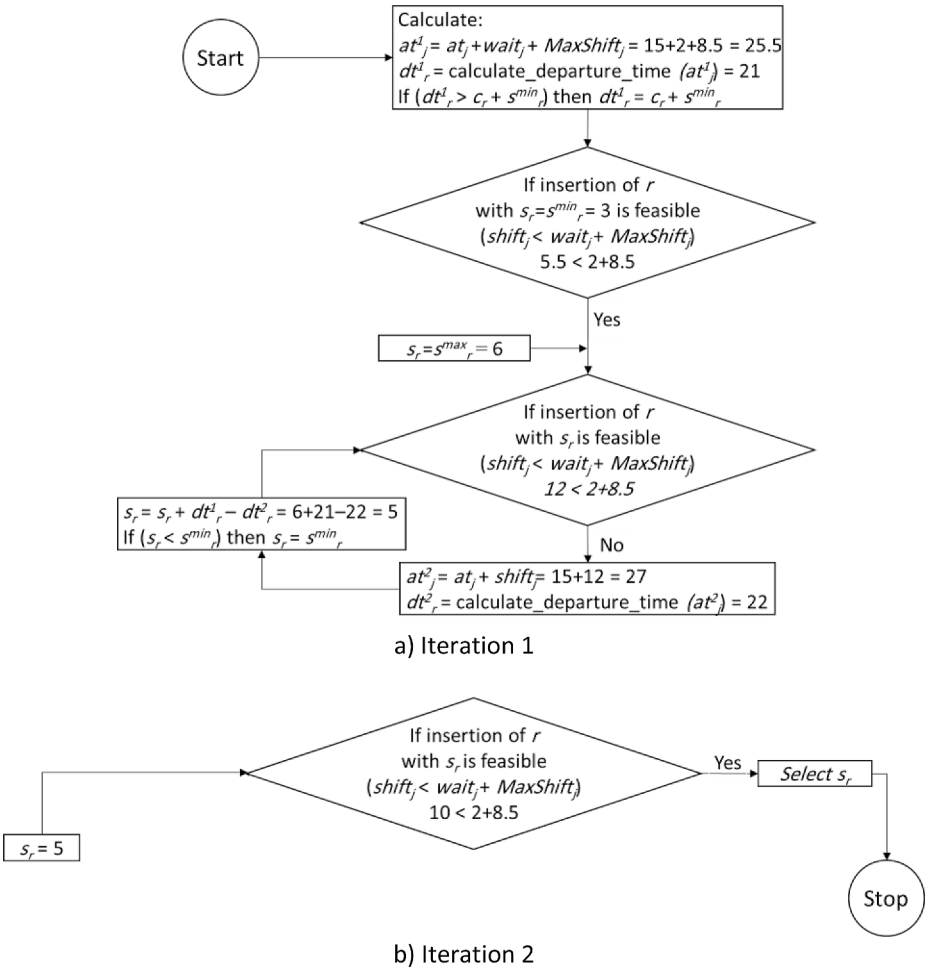

Fig. B.2. A numerical example of *FindBestS*.

Appendix C. The TDOPTW-STP Shiraz instance

See Table C.1.

Table C.1
Properties of vertices in TDOPTD-STP instance (Sce 0).

Vertex	Latitude (N)	Longitude (E)	Opening time (second)	Closing time (second)	Min service time (second)	Max service time (second)	Min profit	Max profit
1 (hotel)	29.632842	52.556191	21,600	79,200	0	0	0	0
2	29.616274	52.547944	32,400	72,000	1800	3000	9	13
3	29.625141	52.557752	28,800	78,300	2820	4200	21	30
4	29.621823	52.581676	28,800	76,140	2880	4200	19	28
5	29.608909	52.534294	25,200	79,200	1680	2400	10	15
6	29.592549	52.583558	25,200	79,200	1800	2400	8	11
7	29.635857	52.561808	28,800	79,200	2580	3300	15	21
8	29.61023	52.537126	30,600	64,800	2280	3600	9	13
9	29.641362	52.552385	21,600	82,800	1080	1800	8	12
10	29.613129	52.53894	36,000	61,200	1260	2100	10	14
11	29.636911	52.526297	28,800	70,200	3540	4800	16	23
12	29.608012	52.552318	34,200	68,400	4080	5100	19	27
13	29.611002	52.540683	30,600	72,000	2700	3600	13	19
14	29.618932	52.536115	32,400	79,200	3060	3900	16	22
15	29.624204	52.498186	34,200	63,000	4020	5400	15	20
16	29.618251	52.573336	28,800	64,800	3000	4200	13	19
17	29.629435	52.557661	27,000	73,800	3660	5100	15	20
18	29.630551	52.537318	23,400	82,800	3600	4800	12	17
19	29.619564	52.501315	23,400	82,800	2700	4200	10	15
20	29.618877	52.57654	25,200	82,800	2460	3600	8	11
21	29.608884	52.542752	21,600	82,800	3540	5400	18	25
22	29.608717	52.548418	28,800	64,800	3180	4200	16	22
23	29.614993	52.545037	32,400	66,600	3300	4500	19	26
24	29.620046	52.535222	43,200	59,400	1920	2700	12	17
25	29.617676	52.543851	32,400	73,800	2820	3900	16	24
26	29.617064	52.546841	30,600	72,000	1140	1800	12	16
27	29.612353	52.544329	28,800	72,000	1680	2700	8	11
28	29.629004	52.562279	30,600	63,000	1920	3000	12	16
29	29.616037	52.545057	30,600	63,000	3780	4800	13	19
30	29.59277	52.574244	32,400	57,600	2340	3900	12	16
31	29.618046	52.545394	32,400	54,000	2280	3600	13	18
32	29.631428	52.494688	43,200	54,000	3660	5400	10	14
33	29.666442	52.479491	39,600	54,000	3300	5400	10	14
34	29.632692	52.522044	43,200	54,000	3300	5400	8	11
35	29.624327	52.53974	54,000	81,000	1920	3000	9	13
36	29.62467	52.523773	54,000	81,000	1980	3000	8	12
37	29.678142	52.459419	54,000	81,000	2040	3000	8	11
38	29.652894	52.486172	36,000	75,600	2640	3600	10	14
39	29.685339	52.471618	34,200	82,800	3960	5400	12	16

Appendix D. Details of finding travel times for the real instance of Shiraz

To consider changes in the travel time of the road network during the day, the day time between 6 am and 10 pm is divided into 65 time slots of 15 min. Therefore, to collect the travel time of the network during the day, a travel time matrix with dimensions of 39 by 39 is created 65 times (at the start of each time slot) during the day.

To collect this information, the online information of the “Neshan” website is used. “Neshan” is a navigation application similar to “Waze” and belongs to an Iranian company called “Rajman Information Structures”. It uses the information technology (IT) and by analyzing the temporal-spatial position of vehicle probes in the urban road network, produces high quality and accurate data including road travel times. In fact, a user request includes the location of origin and destination, and the response includes the proposed route and the corresponding travel time at that moment.

For a large volume of requests at one moment, it is impossible to do this step manually. Therefore, a Python program has been used. The program first asks the input information including the number of vertices, the geographical coordinates of the vertices, start and end time of data gathering, and the length of the time slots. Then, at the start of each time slot, a matrix containing 39*39 requests is sent and the received responses are recorded in a text file. This process is repeated 65 times, every 15 min, during the day. The received time dependent travel times per time slot are stored for each virtual arc. A virtual arc is a dummy arc that holds a concatenation of arcs connecting two POIs. During this process, the linear piecewise travel times are guaranteed to follow the FIFO rule if $\left|\mu_{ijt}\right| \leq 1$ (Fleischmann et al., 2004).

References

Abbaspour, R.A., Samadzadegan, F., 2011. Time-dependent personal tour planning and scheduling in metropolises. Expert Syst. Appl. 38 (10), 12439–12452.

Chao, I.M., Golden, B.L., Wasil, E.A., 1996. A fast and effective heuristic for the orienteering problem. Eur. J. Oper. Res. 88 (3), 475–489.

Chen, X., 2012. Fast patrol route planning in dynamic environments. IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. 42 (4), 894–904.

Chircop, P.A., Surendonk, T.J., van den Briel, M.H.L., Walsh, T., 2013. A column generation approach for the scheduling of patrol boats to provide complete patrol coverage. In: Proceedings of the 20th International Congress on Modelling and Simulation, pp. 1–6.
Dewil, R., Vansteenwegen, P., Cattrysse, D., Van Oudheusden, D., 2015. A minimum cost network flow model for the maximum covering and patrol routing problem. Eur. J. Oper. Res. 247 (1), 27–36.
Dib, O., Manier, M.-A.-A., Moalic, L., Caminada, A., 2017. Combining VNS with Genetic Algorithm to solve the one-to-one routing issue in road networks. Comput. Oper. Res. 78, 420–430.
Divsalar, A., Vansteenwegen, P., Cattrysse, D., 2013. A variable neighborhood search method for the orienteering problem with hotel selection. Int. J. Prod. Econ. 145 (1), 150–160.
Divsalar, A., Vansteenwegen, P., Sörensen, K., Cattrysse, D., 2014. A memetic algorithm for the orienteering problem with hotel selection. Eur. J. Oper. Res. 237 (1), 29–49.
Ekici, A., Retharekar, A., 2013. Multiple agents maximum collection problem with time dependent rewards. Comput. Ind. Eng. 64 (4), 1009–1018.
Erdoğan, G., Laporte, G., 2013. The orienteering problem with variable profits. Networks 61 (2), 104–116.
Erkut, E., Zhang, J., 1996. The maximum collection problem with time-dependent rewards. Naval Res. Logistics (NRL) 43 (5), 749–763.
Feillet, D., Dejax, P., Gendreau, M., 2005. Traveling salesman problems with profits. Transport. Sci. 39 (2), 188–205.
Fleischmann, B., Gietz, M., Gnutzmann, S., 2004. Time-varying travel times in vehicle routing. Transport. Sci. 38 (2), 160–173.
Fomin, F.V., Lingas, A., 2002. Approximation algorithms for time-dependent orienteering. Inf. Process. Lett. 83 (2), 57–62.
Garcia, A., Linaza, M.T., Arbelaitz, O., Vansteenwegen, P., 2009. Intelligent Routing System for a Personalised Electronic Tourist Guide. Springer, Wien New York, pp. 185–197.
Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., Linaza, M.T., 2013. Integrating public transportation in personalised electronic tourist guides. Comput. Oper. Res. 40 (3), 758–774.
Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., 2014. Efficient cluster-based heuristics for the team orienteering problem with time windows. Asia-Pacific J. Oper. Res. 36 (01), 1950001.
Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., Vathis, N., 2015. Heuristics for the time dependent team orienteering problem: application to tourist route planning. Comput. Oper. Res. 62, 36–50.
Golden, B.L., Levy, L., Vohra, R., 1987. The orienteering problem. *Naval Res. Logistics (NRL)* 34 (3), 307–318.
Guitouni, A., Masri, H., 2014. An orienteering model for the search and rescue problem. CMS 11 (4), 459–473.
Gunawan, A., Lau, H.C., Vansteenwegen, P., 2016. Orienteering Problem: a survey of recent variants, solution approaches and applications. Eur. J. Oper. Res. 255 (2), 315–332.
Gunawan, A., Ng, K.M., Kendall, G., Lai, J., 2018. An iterated local search algorithm for the team orienteering problem with variable profits. Eng. Optim. 50 (7), 1148–1163.
Gündling, F., Witzel, T., 2020Time-Dependent Tourist Tour Planning with Adjustable Profits. *In 20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2020)*, *Schloss Dagstuhl-Leibniz-Zentrum für Informatik*.
Keskin, B.B., Li, S.R., Steil, D., Spiller, S., 2012. Analysis of an integrated maximum covering and patrol routing problem. Transport. Res. Part E: Logist. Transport. Rev. 48 (1), 215–232.
Jandaghi, H., Divsalar, A., Emami, S., 2021. The categorized orienteering problem with count-dependent profits. Appl. Soft Comput. 113.
Li, J., 2011. Model and algorithm for time-dependent team orienteering problem. In: International Conference on Computer Education, Simulation and Modeling. Springer, Berlin, Heidelberg, pp. 1–7.
Li, J., Wu, Q., Li, X., Zhu, D., 2010. Study on the time-dependent orienteering problem. In: 2010 International Conference on E-Product E-Service and E-Entertainment. IEEE, pp. 1–4.
Liao, Z., Zheng, W., 2018. Using a heuristic algorithm to design a personalized day tour route in a time-dependent stochastic environment. Tourism Manage. 68, 284–300.
Lou, Y., Yin, Y., Lawphongpanich, S., 2011. Freeway service patrol deployment planning for incident management and congestion mitigation. Transport. Res. Part C: Emerg. Technol. 19 (2), 283–295.
Moonen, M., Cattrysse, D., Van Oudheusden, D., 2007. Organising patrol deployment against violent crimes. Oper. Res. Int. Journal 7 (3), 401–417.
Murat Afsar, H., Labadie, N., 2013. Team orienteering problem with decreasing profits. Electr. Notes Discr. Mathematics 41, 285–293.
Paydar, M.M., Saidi-Mehrabad, M., 2013. A hybrid genetic-variable neighborhood search algorithm for the cell formation problem based on grouping efficacy. Comput. Oper. Res. 40 (4), 980–990.
Peng, G., Dewil, R., Verbeeck, C., Gunawan, A., Xing, L., Vansteenwegen, P., 2019. Agile earth observation satellite scheduling: an orienteering problem with time-dependent profits and travel times. Comput. Oper. Res. 111, 84–98.
Pietz, J., Royset, J.O., 2013. Generalized orienteering problem with resource dependent rewards. Naval Res. Logistics (NRL) 60 (4), 294–312.
Portugal, D., Rocha, R.P., 2013. Distributed multi-robot patrol: a scalable and fault-tolerant framework. Rob. Auton. Syst. 61 (12), 1572–1587.
Takamiya, M., Watanabe, T., 2011. Planning high responsive police patrol routes with frequency constraints. *In Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication, ICUIMC 2011*, (pp. 1–8).
Tang, H., Miller-Hooks, E., Tomastik, R., 2007. Scheduling technicians for planned maintenance of geographically distributed equipment. Transport. Res. Part E: Logistics Transport. Rev. 43 (5), 591–609.
Tarantino, E., De Falco, I., Scafuri, U., 2019. A mobile personalized tourist guide and its user evaluation. Inf. Technol. Tourism 21 (3), 413–455.
Tsiligirides, T., 1984. Heuristic methods applied to orienteering. J. Oper. Res. Soc. 35 (9), 797–809.
Vansteenwegen, P., Gunawan, A., 2019. Orienteering problems. In: Speranza, M.G., Oliveira, J.F. (Eds.), EURO Advanced Tutorials on Operational Research. Springer International Publishing.
Vansteenwegen, P., Souffriau, W., Oudheusden, D.V., 2011. The orienteering problem: a survey. Eur. J. Oper. Res. 209 (1), 1–10.
Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., Van Oudheusden, D., 2009. Iterated local search for the team orienteering problem with time windows. Comput. Oper. Res. 36 (12), 3281–3290.
Verbeeck, C., Sörensen, K., Aghezzaf, E.H., Vansteenwegen, P., 2014. A fast solution method for the time-dependent orienteering problem. Eur. J. Oper. Res. 236 (2), 419–432.
Verbeeck, C., Vansteenwegen, P., Aghezzaf, E.H., 2017. The time-dependent orienteering problem with time windows: a fast ant colony system. Ann. Oper. Res. 254 (1), 481–505.
Willemse, E.J., Joubert, J.W., 2012. Applying min-max k postmen problems to the routing of security guards. J. Operat. Res. Soc. 63 (2), 245–260.
Yu, J., Aslam, J., Karaman, S., Rus, D., 2015. Anytime planning of optimal schedules for a mobile sensing robot. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 5279–5286.
Yu, Q., Fang, K., Zhu, N., Ma, S., 2019. A matheuristic approach to the orienteering problem with service time dependent profits. Eur. J. Oper. Res. 273 (2), 488–503.
Zenker, B., Ludwig, B., 2009. Rose: assisting pedestrians to find preferred events and comfortable public transport connections. In: Proceedings of the 6th International Conference on Mobile Technology, Application and Systems, pp. 1–5.
Zhu, N., Liu, Y., Ma, S., He, Z., 2014. Mobile traffic sensor routing in dynamic transportation systems. IEEE Trans. Intell. Transp. Syst. 15 (5), 2273–2285.