Computers & Operations Research 37 (2010) 351 – 367

Contents lists available at ScienceDirect

Computers & Operations Research

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

Heuristics for the multi-period orienteering problem with multiple time windows

Fabien Tricoire[a,*], Martin Romauch[a], Karl F. Doerner[a,b], Richard F. Hartl[a]

[a]Department of Business Administration, University of Vienna, Bruenner Strasse 72, 1210 Vienna, Austria
[b]Salzburg Research Forschungsgesellschaft, Jakob-Haringer Strasse 5/III, 5020 Salzburg, Austria

ARTICLE INFO

Available online 6 June 2009

Keywords:
Orienteering
Multi-period
Multiple time windows
Route duration constraints
Variable neighborhood search

ABSTRACT

We present the multi-period orienteering problem with multiple time windows (MuPOPTW), a new routing problem combining objective and constraints of the orienteering problem (OP) and team orienteering problem (TOP), constraints from standard vehicle routing problems, and original constraints from a real-world application. The problem itself comes from a real industrial case. Specific route duration constraints result in a route feasibility subproblem. We propose an exact algorithm for this subproblem, and we embed it in a variable neighborhood search method to solve the whole routing problem. We then provide experimental results for this method. We compare them to a commercial solver. We also adapt our method to standard benchmark OP and TOP instances, and provide comparative tables with state-of-the-art algorithms.

1. Introduction

In this paper we present a solution procedure for the individual route planning of field workers and sales representatives (e.g. sales representatives of food retailers or of the pharmaceutic industry). The problem stems from a software distribution company which sells decision support systems for salesman and marketing departments. The developed algorithm will be integrated in an application for scheduling and planning future working days of sales representatives in an off-line fashion. Field workers and sales representatives can use this tool off-line to plan their trips for a certain planning horizon individually.

Field workers and sales representatives have the duty to visit their customers on a regular basis in specific time intervals. The different requirements for the planning of the customer visits and the resulting planning of the traveling salesman tour can be different for different branches. In the following we consider a variant of sales representatives which has the flexibility to schedule his/her customer visit by him-/herself on a weekly basis. This is a decentralized approach where the salesman can organize the schedule of visits—in comparison to that a centralistic approach is presented in Polacek et al. [17]. An usual planning process of the sales representative is that he/she decides at the end of the week which customers or potential customers should be visited for the next week. The mandatory customers are selected. Some of them have specific time windows when they can be visited. There are even some customers who have a different time window for every day. E.g. one customer can be visited on Monday from 9.00 to 12.00 but on Tuesday he/she can only be visited from 11.00 to 12.00 or from 13.00 to 15.00. Note that up to two different time windows per day are allowed.

As already mentioned sales representatives have to visit their customers with long-term relations on a regular basis in order to present new products. In our problem formulation these are the mandatory customers. Moreover it is also very important for the success of a company to acquire new customers. To consider this situation potential customers located nearby should be possibly integrated into the customer tours. When a salesman for tires visits regularly the garages which sells the companies' tires—he/she should possibly also include garages which are not customers so far. These are the optional customers in our problem. The optional customer selection decision has to be integrated in the solution procedure. For several branches purchasable databases exist, providing the potential customers who can be integrated in such routes. These potential customers do not have to be visited and some of them are only included in the schedule when good combinations result with the mandatory customer.

The contribution of the paper is threefold:

- We introduce and define a new problem for scheduling the customer visits of sales representatives. This problem is a generalization of the orienteering problem (OP), team orienteering problem (TOP), orienteering problem with time windows (OPTW) and team orienteering problem with time windows (TOPTW), and is

* Corresponding author. Tel.: +43 1 4277 38114; fax: +43 1 4277 38094.
E-mail addresses: Fabien.Tricoire@univie.ac.at (F. Tricoire), Martin.Romauch@univie.ac.at (M. Romauch), Karl.Doerner@univie.ac.at (K.F. Doerner), Richard.Hartl@univie.ac.at (R.F. Hartl).

0305-0548/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2009.05.012

denoted as the multi-period orienteering problem with multiple time windows (MuPOPTW).

- We develop an exact algorithm for the route feasibility check when having multiple time windows. Multiple time windows can occur in any routing problems with time window requirements.
- We develop an efficient variable neighborhood search algorithm for this specific problem and apply it also to standard instances.

In the TOP a set of customers needs to be selected and a set of tours organized in order to maximize the profit. Feillet et al. (cf. [8]) gather these problem types for a single tour under the name TSP with profits. The used objective functions may be the maximization of the collected total profit (OP), the minimization of the total traveling costs (prize-collecting TSP) by reaching a predefined minimum profit, or the optimization of a combination of both (profitable tour problem). Archetti et al. (cf. [1]) name the extension of the TSP with profits to multiple tours as vehicle routing problems (VRP) with profits. The extension of the OP to multiple tours, which is a special case of VRP with profits, was introduced under the name TOP by Chao et al. (cf. [5]). The OPTW and TOPTW are generalizations of OP and TOP, with time window constraints. OPTW was introduced by Righini and Salani and solved to optimality with a dynamic programming algorithm [19,20]. Vansteenwegen introduced the TOPTW, and provided a first heuristic approach for the OPTW and TOPTW [28].

The most suitable problem definition for the problem of the sales representatives is the TOPTW. In fact the representatives problem combines optional customers from all orienteering problems and mandatory customers from more traditional routing problems. A multi-period horizon is also an important characteristic here, along with multiple time windows. This means that each customer may be visited on different days, on different time slots, and also that each customer may have several time windows for each given day. In practice, a customer has zero, one or two time windows per day. In our problem formulation the time windows are not allowed to overlap.

The profits of the optional customers are estimated. It is the expected probability that the potential customer can become a customer, multiplied by the expected turnover linked to this customer.

We designed a metaheuristic solution approach to solve the considered problem. Recent publications show that variable neighborhood search (VNS) algorithms fare very well when compared to other solution techniques, especially for routing problems (see e.g. [18,16,11]).

The VNS is an improvement metaheuristic introduced in [15,14]. The basic idea is a systematic change of the neighborhood used within a local search. Here, several neighborhood structures are applied instead of a single one, as it is the case in a standard local search implementation. The algorithm is applied to standard instances for the TOP and also to real-world instances from a company.

The remainder of the paper is organized as follows. A detailed description of the problem is given in Section 2, as well as a mathematical model. The description of the algorithm for checking route feasibility is reported in Section 4. The metaheuristic solution procedure based on ideas of VNS is reported in Section 3. Detailed numerical results are reported in Section 5.

2. Model formulation

We consider a TOP with multiple time windows. Potential and mandatory customers have to be visited in a planning horizon. The planning horizon consists of several periods—in each period one route is planned. In our problem we introduce additional time aspects. We have a maximum route duration of the daily routes which cannot be exceeded. Furthermore, a maximum working time in the planning horizon is allowed. Consider a planning horizon of three days—the maximum route duration may not exceed 10 h whereby the maximum working time within this period may not exceed 24 h.

There exists a wide time window at the depot which must contain the starting and ending time of the route. Within this time window the route of the salesman should start and can end. The customers can have zero, one or two time windows. In the two time window case one out of these has to be selected. Note that the time windows on one day are non-overlapping. The time windows are hard, and must not be violated.

The profits of the optional customers are derived from an estimation of expected turnover and the probability that an order can be placed with this potential customer. The customers have different service times. The visit (in the classical routing literature also called service) has to end before the end of the time window. This is not standard to the classical routing notation, where service only has to start before the end of a time window. The practitioner preferred this notation; a transformation to the notation of the standard literature is straightforward.

There is no capacity restriction on the vehicle. The distance matrix consists of the travel time. The objective is to maximize the profit; all the constraints have to be satisfied, there are no soft constraints.

Let $V=U\dot{\cup}W\dot{\cup}O$ (disjoint sets) be the set of nodes which consists of mandatory (U) and optional (W) customers, and the depots (O). I.e. the set $U\subset V$ contains the mandatory customers that have to be visited and the set $W\subset V$ contains the potential customers, that are candidates for bringing new profit. For each day $t\in\mathcal{T}=\{1,\dots,|\mathcal{T}|\}$ (e.g. $\mathcal{T}=\{1,2,3\}$) we define:

- A starting depot $o_s^t\in O$ and an ending depot $o_e^t\in O$. For the sake of genericity all these points are different, although in practice they might be associated to identical locations. This allows to handle a wide range of practical cases; for instance, in the case where a driver makes a several-day trip across the country and stops every night in previously fixed hotels, the ending point for day t will refer to the same location as the starting point for day $t+1$. Since all the points are different, we have $|O|=2.|\mathcal{T}|$.
- For each customer $i\in U\cup W$, the time windows $\tau_i^{tk}=[a_i^{tk},b_i^{tk}]$, where k is the element of the index set $\Theta_i^t=\{1,\dots,K_i^t\}$ (e.g. $\Theta_i^t=\{1,2\}$ if there are two possible time windows for customer i on day t). For each depot $o\in O$, $\Theta_o^t=\{1\}$ and $\tau_o^{t1}=[0,\mathcal{B}]$.

The service time for customer i is denoted s_i and d_{ij} is the travel time from location i to location j. For each day t we plan a route that starts at a given node o_s^t and ends at o_e^t, the duration of which is bounded by $\aleph^t$. Additionally, the sum of route durations should not exceed $\tilde{\aleph}$. The profit of the optional customer $i\in W$ is denoted by q_i (e.g. the expected value of the profit). Now we define the main decision variables and explain their meaning; for the routing we have

$$x_{ij}^t=\begin{cases}1 & \text{if customer } j \text{ is directly visited after customer } i \text{ on day } t\\ 0 & \text{else}\end{cases}$$

and for the customer selection we define

$$y_i^{tk}=\begin{cases}1 & \text{if customer } i \text{ is selected on day } t \text{ with time window } k\\ 0 & \text{else}\end{cases}$$

The departure time from customer i on day t is stored in the continuous variables α_i^t. For the sake of simplicity we also define the number of visits of customer i:

$$y_i:=\sum_{t\in\mathcal{T}}\sum_k y_i^{tk}$$

Since multiple visits are not considered, constraints should ensure that this number never exceeds 1. Now we detail the handling of multiple time windows: let $\{\tau_i^{tk}|k\in\Theta_i^t\}$ (where $\Theta_i^t=\{1,\dots,K_i^t\}$ is an index set) be the set of possible time windows for customer i on day t. Then the k th time window is denoted by $\tau_i^{tk}=[a_i^{tk},b_i^{tk}]$.

Using these notations, we write a mathematical model for the MuPOPTW. The objective is to maximize the expected profit:

$$\max \sum_{i \in W} q_i y_i \tag{1}$$

s.t.

$$\sum_{j \neq i} x_{ij}^t = \sum_{j \neq i} x_{ji}^t \quad \underset{i \in U \cup W}{\forall} \ \underset{t \in \mathscr{T}}{\forall} \tag{2}$$

$$\sum_{i \in V \setminus \{o_s^t\}} x_{o_s^t i}^t = \sum_{i \in V \setminus \{o_e^t\}} x_{i o_e^t}^t = 1 \quad \underset{t \in \mathscr{T}}{\forall} \tag{3}$$

$$y_i := \sum_k \sum_t y_i^{tk} \quad \underset{i \in V}{\forall} \tag{4}$$

$$y_i = 1 \quad \underset{i \in U}{\forall} \tag{5}$$

$$y_i \leq 1 \quad \underset{i \in W}{\forall} \tag{6}$$

$$\sum_k y_i^{tk} = \sum_{j \neq i} x_{ij}^t \quad \underset{i \in V}{\forall} \quad \underset{t \in \mathscr{T}}{\forall} \tag{7}$$

$$\alpha_i^t + d_{ij} + s_j + \mathscr{B}.x_{ij}^t \leq \alpha_j^t + \mathscr{B} \quad \underset{t \in \mathscr{T}}{\forall} \ \underset{i \in V}{\forall} \ \underset{j \in V}{\forall} \tag{8}$$

$$(a_i^{tk} + s_i) y_i^{tk} \leq \alpha_i^t \leq b_i^{tk} + \mathscr{B}(1 - y_i^{tk}) \quad \underset{i \in V}{\forall} \ \underset{t \in \mathscr{T}}{\forall} \ \underset{k \in \Theta_i^t}{\forall} \tag{9}$$

$$\alpha_{o_e^t}^t - \alpha_{o_s^t}^t \leq \aleph^t \quad \underset{t \in \mathscr{T}}{\forall} \tag{10}$$

$$\sum_{t \in \mathscr{T}} (\alpha_{o_e^t}^t - \alpha_{o_s^t}^t) \leq \tilde{\aleph} \tag{11}$$

Constraints (2) ensure connectivity in the solution; the depots represent a particular case, which is handled by constraints (3). Mandatory customers should be visited exactly once (5), whereas optional customers can be visited once or not at all (6). We also need an auxiliary constraint to ensure that only customers that are visited on a given day can be serviced on this same day (7). Constraints (8) forbid to leave from a node before arriving to this node and performing the required service. Time windows are handled in a classical way by constraints (9), with the noticeable addition of an index, since we are dealing with multiple time windows. Route duration limit is also taken care of (10), as well as total duration limit for the whole horizon (11).

As shown in Section 5, solving the model by using a commercial mixed integer programming solver is not promising. We therefore developed heuristic methods in order to deal with the MuPOPTW; within these heuristics an exact route feasibility check is used. Since this check is quite difficult, we defer its description and the developed solution procedure in Section 4; a formal proof is provided in Appendix A.

3. Heuristic algorithms for the MuPOPTW

We developed two heuristic algorithms for the MuPOPTW: a constructive heuristic, and a VNS. The constructive heuristic is deterministic, and provides a starting solution for the VNS, which is a stochastic local search algorithm. Both these methods rely on the route feasibility check algorithm detailed in the next section, and make intensive use of it. Therefore, all feasibility aspects relative to multiple time windows are delegated to this route feasibility check algorithm. Concerning the multi-period aspect, each day of the horizon is associated to a different tour in the solution. This means for instance that moving a customer from one tour to another also implies that this customer is moved to another day. By consequence, care has to be taken regarding the availability of customers on given days, and a customer cannot be assigned to a tour (day) for which it has no available time window. As described later, these checks are performed when relevant, which means either during shaking or during local search, depending on the context.

3.1. Constructive heuristic

We explain here how we build the starting solution, which will be the first incumbent solution of the VNS.

We used a modified version of the *best insertion* heuristic by Solomon [23]. This heuristic proceeds in two steps:

1. For each unplanned demand, compute the best feasible insertion in the partial solution, following criterion c_1.
2. Perform the best of all these best feasible insertions, following criterion c_2.

The feasibility of the insertions is checked via Algorithm 4 (to be explained in Section 4). Naturally, the choice of criteria c_1 and c_2 has an impact. In our case, c_1 is “minimize extra distance” and c_2 is “minimize extra time”. Those are also criteria used in the original best insertion heuristic by Solomon. Minimizing extra distance means that the best insertion is the one that involves the lowest increase in distance. Minimizing extra time means that the best insertion is the one that involves the lowest increase in time required to perform the route in which the insertion is performed. The underlying idea is to use the waiting time in an efficient manner, and to perform cheap insertions that make use of the waiting time, instead of increasing the duration of routes.

However, we do not apply directly this heuristic to the whole set of customers. In a first stage, we only apply it to the set of mandatory customers, to avoid infeasible solutions later. Once all mandatory customers are planned, the list of optional customers is sorted by decreasing order of expected profit. Then the optional customers are considered sequentially for insertion, and if possible, “best-inserted”. This means that given a customer we look for its best possible insertion in every route considering criterion c_1, and then insert it in the best route considering criterion c_2.

Our algorithm also differs in one point with the original one by Solomon: we consider all possible routes from the beginning, where in Solomon's algorithm only one route is considered and new routes are created when it is no longer possible to insert customers into the already existing routes. Since our objective is to maximize profit, it is more than likely that all possible routes (i.e. days) will need to be used, in order to visit as many customers as possible. Therefore the starting solution is made of empty routes, one for each day, going from starting point to ending point for this day.

3.2. Variable neighborhood search

The basic idea of VNS is a systematic change of neighborhoods within a local search procedure. More precisely, the VNS explores larger and larger neighborhoods of the incumbent solution. The search jumps from its current point in the solution space to a new one if an improvement has been made or some acceptance criterion is met. The steps of the basic VNS are shown in Algorithm 1, where N_κ ($\kappa = 1, \ldots, \kappa_{\max}$) is the set of neighborhoods. The general stopping condition can be a limit on the CPU time, on the number of iterations, or on the number of iterations without improvements. We refer to Mladenović and Hansen [15] and Hansen and Mladenović [9] for a more thorough description of VNS. Next, we describe the different components of the VNS that we implemented for the MuPOPTW.

Algorithm 1. Steps of the VNS (see Hansen and Mladenović [9]).

```
Initialization: Select the set of neighborhood structures $N_\kappa$ ($\kappa = 1,\dots,\kappa_{\max}$) to be used in the search; find an initial solution $x$ as incumbent solution; choose a stopping condition;
Repeat the following steps until the stopping condition is met:
  1. Set $\kappa \leftarrow 1$;
  2. Repeat the following steps until $\kappa = \kappa_{\max}$:
    (a) Shaking: Generate a random point $x'$ from the $\kappa$th neighborhood of incumbent solution $x$ ($x' \in N_\kappa(x)$);
    (b) Iterative improvement: Apply some iterative improvement method with $x'$ as initial solution; denote by $x''$ the so-obtained local optimum;
    (c) Move or not: If $x''$ is better than the incumbent or some acceptance criterion is met, accept $x''$ as new incumbent ($x \leftarrow x''$) and continue the search with $N_1$ ($\kappa \leftarrow 1$); otherwise, set $\kappa \leftarrow \kappa + 1$;
```

The initial solution is provided by the adapted best insertion algorithm described previously. We now describe the three components for this VNS.

3.3. Shaking

The set of neighborhoods used for shaking is the central design element of the VNS. Each neighborhood should strike a proper balance between perturbing the incumbent solution and retaining the good parts of the incumbent solution. We use the usually accepted idea that optimizing the routes with regards to traditional cost/duration objectives will help to make free room in the routes, thus allowing to insert more optional customers. For this reason our first neighborhoods work on this aspect. One effective neighborhood for vehicle routing is based on the *cross-exchange* operator [24]. In a cross-exchange, two segments of different routes are interchanged, by preserving the orientation of both the segment(s) and the route(s). Such an operator was successfully used in a number of applications, including, e.g. the VNS for the VRP with time windows by Bräysy [2].

In our VNS, the cross-exchange is used to define a set of neighborhoods that allows the exploration of solutions increasingly distant from the incumbent, in order to overcome local optimality and strive for global optimality. The metric to measure the increasing size of a neighborhood is given by the maximum number p of customers in the route segments to be interchanged. Let n_R denote the number of customers in a route R, then in neighborhood κ the maximum length of a segment is given by $\min\{n_R, \kappa\}$. The minimum segment length is always 1 for the first route and 0 for the second one. As a consequence, it is possible to move nodes from one route to another, but "identity" moves never happen, since at least one node is systematically moved. When exchanging customers randomly, it may happen that a customer is moved to a day (tour) for which it has no valid time window, which could not possibly lead to a feasible solution. Therefore, customers are only moved to another tour if they can be visited on the day associated to this tour.

In each neighborhood all the possible segment lengths are equally likely to be chosen. Once a cross-exchange is chosen, the new routes are constructed, possibly with some infeasibility due do time constraints (time windows and route duration), and passed to the iterative improvement phase. Keeping the minimum length of exchanged segments always equal to 1/0 implies that the neighborhoods are nested. This choice biases the search towards smaller segment lengths, focusing more on the portion of the solution space rather close to the incumbent solution.

Table 1
Shaking neighborhoods for the VNS.

κ	Neighborhood type	Parameters		
1–8	Cross-exchange	$p = \kappa$		
		κ	p (remove)	q (insert)
9–12	Optional exchange 1	9	0	1
		10	1	1
		11	2	1
		12	0	2
13–17	Optional exchange 2	$p = \kappa - 12$		

In order to allow changes in the choice of planned and unplanned customers, we use a second kind of neighborhood, which we call *optional exchange* 1. It consists in selecting a route and a position in this route, removing a certain number p of optional customers found at this position, and inserting another number q of unplanned optional customers instead. In this case, only customers that have at least one valid time window for the route are selected; more precisely, the q first unplanned customers with a valid time window for this route are selected. The number of customers to remove and insert change in order to allow increasing perturbation. These numbers are fixed and not upper bounds, to avoid a large amount of moves that would remove some customers while adding too few to make it a profitable move. After such a move, the modified route might be infeasible due to time constraints, either time windows or route duration.

This neighborhood allows to choose different optional customers, and to provide diversification with regards to this choice; however, it is not very aggressive, and might be insufficient to converge in a reasonable time. We now introduce a second neighborhood which also deals with customer selection, called *optional exchange* 2. It consists in selecting a route and a position in this route, and then removing a certain amount p of optional customers from this route. This is later balanced by a specific iterative improvement procedure which tries to insert optional customer as long as it is possible without making the route infeasible. This neighborhood is much more aggressive than the previous one, and by combining both we aim at providing a decent amount of intensification and diversification. As with *optional exchange* 1, the amount of customers to be removed is not an upper bound, but a fixed value.

Finally, Table 1 indicates the order in which the shaking neighborhoods are used through the VNS.

3.4. Iterative improvement

A solution obtained through shaking is locally improved by applying an iterative improvement procedure afterwards.

For the first two shaking operators (*cross-exchange* and *optional exchange* 1) we use a subset of 3-opt (which we call 3-opt*) to re-optimize those routes that have been changed during the shaking phase. 3-opt was introduced and first used by Lin [13], to solve the Traveling Salesman Problem. It consists in iteratively improving a tour by performing 3-exchanges. A 3-exchange consists in removing three arcs (or edges in the symmetric case) from a tour, and reconnecting the tour with different arcs (edges) from those which were removed. A restriction we make consists in never considering moving chains of more than three nodes. Since we are dealing with time windows, we only consider moves that do not invert chains. A first improvement strategy is applied, which means that the iterative improvement procedure restarts immediately after an improving move is found.

For the last shaking operator (*optional exchange* 2), we use a *sequential best insertion* procedure, which considers all unplanned optional customers in sequence, and for each of them, tries to *best-insert* them as during the first construction phase. This can be seen as sequentially considering each unplanned customer and applying best insertion with only this customer. The order in which we try to insert new customers is important; however, it is not obvious to determine what would be a good choice. We decided to consider the set of unplanned customers as a queue: after being processed (i.e. used as a candidate for insertion), a customer is pushed at the end, and when a new customer insertion is required, the front of the queue is considered. The underlying idea is to mimic a tabu-like effect, thus avoiding two possible drawbacks: cycling by always trying to insert the same optional customers, and lacking intensification by using total randomness.

As mentioned earlier, the iterative improvement procedure might start with an infeasible route. This is not accepted for an incumbent solution though, so we have to reduce infeasibility as much as possible while iteratively improving. For this reason, the move acceptance decision for the 3-opt* is subject to two conditions: the total duration of the route should not be increased, and the infeasibility measure with regard to time windows should also not be increased. Additionally, at least one of these two values should be improved. The time windows infeasibility is measured in a very simple fashion, by starting from the depot as early as possible, and waiting when required. The amount of service time executed outside of time windows in such a route is the value of this infeasibility.

After a local optimum is found the feasibility of the route has to be checked anyway, since it is possible to iteratively decrease infeasibility without eliminating it. If the solution is infeasible at this stage, then it is disregarded, and a new iteration of the VNS starts; otherwise, it is considered for acceptance.

3.5. Acceptance decision

After the shaking and the local search procedures have been performed, the solution obtained is compared with the incumbent, to be able to decide whether or not to accept it. The acceptance criterion in the basic VNS is to accept only improvements. In order to avoid getting trapped in bad local optima it may be beneficial, in some situations, to also accept non-improving solutions. This is the case, e.g. of the skewed VNS (see Hansen and Mladenović [10]), in which a solution is not only evaluated by its objective value, but also by its distance to the incumbent solution, favoring more distant solutions. A skewed VNS was recently used to solve various kinds of TOPs by Vansteenwegen [28].

Instead of the basic or skewed VNS acceptance criteria, we decided to use threshold accepting. In other publications, e.g. Polacek et al. [16–18], it already provided good quality solutions and fairly reasonable diversification as a side effect. In particular, every 8000 iterations since the last acceptance of a solution, we accept also solutions deteriorating the incumbent solution value, under certain conditions. These conditions vary according to the shaking neighborhood that was used:

- When using the *cross* -neighborhood, a solution deteriorating the total distance by less than 0.5% is accepted.
- When using a neighborhood that may change the selection of visited optional customer, any deterioration is accepted. The use of the *optional exchange* 2 limits the potential negative impact of this policy.

4. Route feasibility check

In this section, we give an overview of the exact algorithm used to determine whether a given route is feasible or not with regards to time constraints, namely time windows and route duration limit. A more detailed description and a proof are provided in Appendix A.

For the sake of simplicity, we suppose that the customers in the route are now named $\{1,\ldots,n\}$. Since we are working on a route assigned to a given day of the horizon, t becomes a fixed value. Therefore we do not mention the index t in this section. For a fixed sequence of customers the travel time can be integrated into the service time; therefore we define the transformed service times and time windows (for all time windows k, i.e. $\forall k \in \Theta_i$):

- $i=1$: $s_1 \leftarrow d_{o_s,1}+s_1$ and $a_1^k \leftarrow a_1^k - d_{o_s^t,1}$.
- $1<i<n$: $s_i \leftarrow d_{i-1,i}+s_i$ and $a_i^k \leftarrow a_i^k - d_{i-1,i}$.
- $i=n$: $s_n \leftarrow d_{n-1,n}+s_n+d_{n,o_e}$ and $a_n^k \leftarrow a_n^k - d_{n-1,n} - d_{n,o_e}$.

Travel times can now be disregarded. Some preprocessing can still be performed to tighten time windows, and delete some useless ones; details on this step are given in Appendix A. This preprocessing has an $O(m)$ time complexity, where m is the total number of time windows for the customers in the route. At this point, we can guarantee that there exists a vector $\alpha=(\alpha_1,\ldots,\alpha_n)$ of departure times, such that no time window constraint is violated (else the preprocessing stops prematurely). We now want to find out the best α, i.e. the one that minimizes total route duration, which is equal to $\alpha_n-\alpha_1+s_1$. If this value is lower than the route duration limit then the route is feasible.

We now introduce the concept of *dominant solution*, which is a key to our algorithm.

Definition 4.1 (*dominant solution*). A solution $\alpha=(\alpha_1,\ldots,\alpha_n)$ is called *dominant* if there exists no other solution $\alpha'=(\alpha'_1,\ldots,\alpha'_n)$ such that $\alpha_n=\alpha'_n$ and $\alpha_1<\alpha'_1$.

In other terms, a solution is dominant if there exists no better solution with the same finishing time. Any optimal solution to our problem is obviously dominant. In order to solve this problem, it is therefore sufficient to consider only dominant solutions. Our algorithm enumerates a subset of the set of all dominant solutions; this subset is sufficient to find at least one optimal solution. We start by the earliest dominant solution; its finishing time α_n is obtained by performing the service at every customer as early as possible; then the service at each customer, from $n-1$ down to 1, is pushed as late as possible without finishing the tour later than α_n. After this initial step, other *interesting* dominant solutions are investigated, from earliest to latest. The algorithm stops when the latest interesting dominant solution has been evaluated. We now detail how we determine the next *interesting* dominant solution.

We first define a set of functions, one for each customer: $\beta_i : \alpha_i \mapsto \beta_i(\alpha_i) \in \Theta_i$ as the corresponding time window index for departure time α_i from customer i. Suppose that we have a given dominant solution ϑ and customer i is the last customer with a waiting time in the route (otherwise there are no waiting times and the route is obviously optimal in terms of duration). Then we can assume that the departure time of the predecessing customer is defined by the border of the time window τ_{i-1}: $\alpha_{i-1}=b_{i-1}^{\beta(\alpha_{i-1})}$. If we want to consult a dominant solution that finishes later, we have to switch from time window $\beta(\alpha_{i-1})$ to the next time window $\beta(\alpha_{i-1})+1$. The algorithm simply proceeds by enumerating all interesting solutions, from earliest to latest, and keeping track of the best one.

In order to provide a better understanding of our algorithm we now apply it in a small example, with four customers and two time windows per customer, shown in Fig. 1.

We detail here the different steps of the algorithm on the example presented in Fig. 1:

1. We start with a dominant solution with length 2.43. The hashed bars represent the service time (including the travel time).
2. We can see that the latest and only delay is between customers 1 and 2. To construct the next interesting solution the service

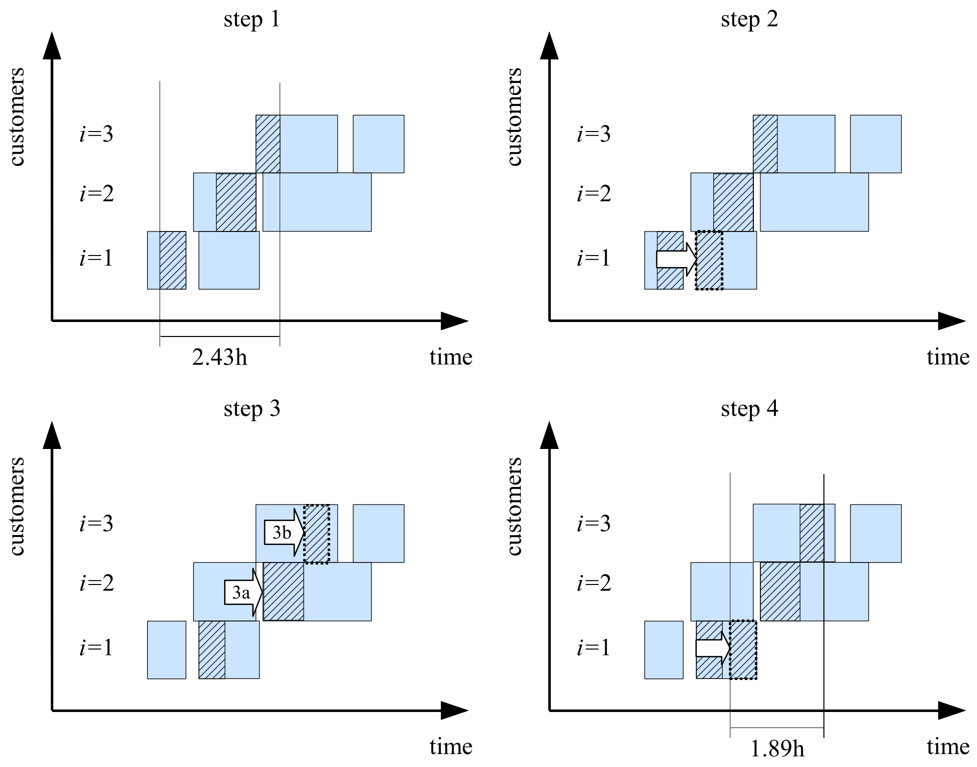

Fig. 1. Minimum route duration on a small example.

of customer 1 is switched to the earliest possible position in the next time window (dashed rectangle).

3. An intermediate solution that respects the predefined service time of customer 1 is constructed. Regarding the "fixed" service of customer 1 it is finishing as early as possible.
4. To make it a dominant solution the services of customers 1 and 2 are pulled towards the "fixed" service of customer 3. Now the length is 1.89 and therefore minimal.

A more complex example, with four time windows per customer, accompanies the proof given in Appendix A.

5. Experimental results

5.1. MuPOPTW instances

We used information provided by the industrial partner to generate 60 problem instances. The industrial partner has five sales representatives and they sell software for geomarketing applications. Since the partner wants to keep the decision of assigning customers to representatives, each instance concerns only one representative, but on a horizon of three days, which implies solutions with three routes, one per day. A pool of customer locations as well as a distance matrix were given, along with potential profits for the customers. Probabilities to gain a customer were generated by us on the basis of five different customer categories (same probability for all customers in a given category). The category with probability 1 corresponds to mandatory customers. The remaining customers are distributed in four categories with probabilities 0.25, 0.2, 0.15, 0.1. Days and time windows were generated independently from these categories. Each customer has an equivalent probability for each possible day combination in the horizon. Given the set of days in the horizon, each of its subsets represents a possible day combination, except for the empty set. In all our instances, a 3-day horizon is considered, for a total of seven possible day combinations. Then a given customer may have the same windows every day (with a probability of $\frac{2}{3}$) or different ones. These time windows are also chosen randomly, as follows:

- 20% are double time windows, which can be any of $[8-12]+[13-16]$, $[8-10]+[12-15]$, $[9-12]+[14-16]$ (dates expressed in daytime hours);
- 30% are single time windows which can last 1, 2 or 3 h, and can be placed anytime during the day, with the only constraint that the release date minutes must be a multiple of 15;
- the remaining 50% are available all day (from 8 to 18).

We expect that this setup is representative for several real-world problems of sales representatives. The depots have a 2-h wider time window (from 7 to 19), but the maximum duration for a given route $\aleph$ is 10 h, and the total maximum duration $\tilde{\aleph}$ is 24 h for the 3-day horizon.

With all this information from the industrial partner, we considered all possible combinations of 6, 8, 10 or 12 mandatory customers with 10, 20, 40, 80 or 140 optional customers, for a total of 20 combinations, i.e. 20 instance sizes. For each combination we generated three instances, for a total of 60 instances. For example, three instances were generated with eight mandatory customers and 40 optional customers, all these customers being selected randomly in the customer pool, separately for each instance. These instances range from a small-size easy ones, where it is possible to visit every customer, to large-size harder ones, where the selection of which customers should be visited is a critical decision, since only a small portion of the optional customers can be visited.

In the following experiments, we compare both the exact method and the VNS to the construction heuristic. From our knowledge of industrial practice, such a construction method is a relatively good approximation of the quality of a very good manual planner. Therefore, the quality of solutions provided by the construction heuristic

Table 2
Commercial solver vs. adapted best insertion.

Instance	Commercial solver		Best insertion	
	Profit	Comp. time (s)	profit	Gap (%)
MuPOPTW_6_10-1*	33 640.15	35	33 640.15	0.00
MuPOPTW_6_10-2*	34 057.55	0	34 057.55	0.00
MuPOPTW_6_10-3*	15 322.05	148	15 322.05	0.00
MuPOPTW_8_10-1*	44 360.70	1277	44 360.70	0.00
MuPOPTW_8_10-2*	14 063.15	1	14 063.15	0.00
MuPOPTW_8_10-3*	38 148.35	0	38 148.35	0.00
MuPOPTW_10_10-1*	25 493.20	1	25 148.70	1.35
MuPOPTW_10_10-2*	44 378.60	1	44 378.60	0.00
MuPOPTW_10_10-3*	32 646.95	45	32 646.95	0.00
MuPOPTW_12_10-2*	58 383.45	12	58 383.45	0.00
MuPOPTW_12_10-3*	38 037.55	38	38 037.55	0.00
MuPOPTW_6_20-1	18 836.30	3600	18 909.10	−0.39
MuPOPTW_6_20-2*	52 852.20	43	52 852.20	0.00
MuPOPTW_6_20-3	22 589.55	3600	24 956.65	−10.48
MuPOPTW_8_20-1*	45 116.00	2	45 116.00	0.00
MuPOPTW_8_20-2*	68 106.75	2	68 106.75	0.00
MuPOPTW_10_20-1	36 244.30	3600	36 616.90	−1.03
MuPOPTW_10_20-2*	35 860.05	591	35 860.05	0.00
MuPOPTW_10_20-3	46 272.80	3600	47 151.45	−1.90
MuPOPTW_12_20-2	51 803.40	3600	51 888.30	−0.16
MuPOPTW_6_40-1	15 021.95	3600	18 515.65	−23.26
MuPOPTW_6_40-2	33 858.30	3600	35 689.70	−5.41
MuPOPTW_6_40-3	17 176.80	3600	20 457.95	−19.10
MuPOPTW_8_40-1	30 299.80	3600	29 944.40	1.17
MuPOPTW_10_40-3	47 670.90	3600	51 769.90	−8.60
MuPOPTW_8_80-2	20 858.15	3600	31 615.45	−51.57
MuPOPTW_10_80-2	51 589.10	3600	62 687.20	−21.51
MuPOPTW_6_140-1	37 325.65	3600	47 532.20	−27.34
MuPOPTW_10_140-1	38 791.60	3600	54 699.60	−41.01

can be seen as a reference point from which one wants to achieve improvement.

In a first step, we compare the performances given by a commercial solver (CPLEX 11), and by the adapted best insertion heuristic. The commercial solver was run on a 3.06 GHz Pentium IV CPU with 2 GB of RAM, with a 60 min time limit. The best insertion heuristic systematically takes less than 1 s, therefore we consider it not relevant to report its computational time. Comparative results are reported in Table 2. Only results on instances for which the solver could find a feasible solution within the time limit are given. When a solution was proved optimal by the solver, an asterisk (*) is appended before its name. We also provide the gap between the solution provided by the two methods. It is calculated as follows: $(Solver - Bestinsertion)/Solver$. Since the solver does not always find the optimal solution, negative gaps are possible, which then means that the heuristic found a better solution than the solver.

When considering these instances for which the commercial solver could find a feasible solution, the best insertion heuristic is beaten only twice (the two rows with positive gaps). For all other instances, the gap is either null or negative. It is interesting to see that many optimal solutions are found by the best insertion; in fact, it corresponds to instances for which there exists a solution satisfying all customers, and the commercial solver did not find any optimal solution involving a choice among optional customers.

We now compare the VNS to the best insertion heuristic. The VNS was run on a 2.4 GHz CPU with 4 GB of RAM. The VNS stops as soon as one of the three following conditions is met: 100,000 iterations, 50,000 iterations without improvement, or 30 min since the beginning of the execution. Table 3 summarizes these results.

The almost negligible gap occurring with small instances (10–20 optional customers) is due to the fact that for most of these instances the best insertion heuristic finds a solution visiting every

Table 3
VNS vs. adapted best insertion: average objective values in function of the number of optional customers.

Class (# opt.)	Profit (BI)	Profit (VNS)	Comp. time (s)	Gap (%)
10	38 537.85	38 598.31	1.67	0.15
20	44 166.52	44 323.63	5.18	0.39
40	45 668.53	47 037.70	24.71	3.16
80	46 674.67	51 389.43	143.96	9.39
140	53 770.25	67 455.75	672.60	20.55
Total	45 763.57	49 760.97	169.62	6.73

customer, and therefore optimal. However, with a larger choice in optional customers, the gap grows dramatically; when considering the largest instances, the gap is more than 20%. When considering that the construction heuristic is an approximation of how well a manual planner would fare, this can be interpreted as a potential for improvement when instance size grows.

In order to provide more insight on the methods and solutions, we now look at the distribution of time between transportation, service, and waiting. Again, we compare these values for both methods. Table 4 summarizes these results.

This table shows that given a sufficient amount of optional customers, the waiting time can be reduced to almost nothing (less than 1 min over three days), even by the best insertion heuristic. However, the VNS also reduces this waiting time to negligible values without the need of a huge amount of optional customers, which the best insertion fails to achieve. In fact, the VNS also produces solutions with transportation times that are comparable with those of the best insertion method, but while providing much more profit. In order to understand better why VNS produces better solutions, we now look more precisely at two indicators: the average profit

Table 4
VNS vs. adapted best insertion: distribution of time between service, transportation and waiting.

Class (# opt.)	Best insertion			VNS		
	Service	Transportation	Waiting	Service	Transportation	Waiting
10	438.33	396.33	80.17	443.75	361.52	0.31
20	529.17	570.92	88.58	540.21	542.20	2.52
40	636.25	735.25	56.92	723.54	800.07	0.47
80	721.25	713.50	3.92	933.50	747.82	1.78
140	674.58	763.67	0.83	1014.29	767.81	0.43
Total	599.92	635.93	46.08	731.06	643.88	1.10

Times are expressed in minutes.

Table 5
VNS vs. adapted best insertion: comparison of profit per customer and transportation time per customer ratios.

Class (# opt.)	Best insertion		VNS	
	Profit	Transportation	Profit	Transportation
10	2010.35	21.26	1992.52	19.37
20	1574.82	20.73	1534.19	18.90
40	1228.23	19.85	1023.08	17.52
80	1162.95	17.97	706.73	10.31
140	1575.53	22.46	751.76	8.63
Total	1510.38	20.45	1201.66	14.95

Table 6
Compared number of unplanned customers by best insertion and VNS.

Class (# opt.)	Best ins. unplanned	VNS unplanned	Difference	$\frac{Difference}{VNSunplanned}$ (%)
10	0.17	0.00	0.00	0.00
20	1.17	0.23	0.00	0.00
40	11.50	3.16	0.09	2.49
80	47.08	15.97	0.63	4.44
140	113.58	57.48	1.63	3.21
Total	34.70	15.37	0.47	2.03

The "Difference" is a set difference, and represents the amounts of customers that, for a same instance, are planned by the best insertion but not by the VNS.

per visited customer ratio, and the average transportation time per visited customer ratio. These ratios are presented in Table 5.

Regarding transportation time, the same tendancy can be observed on both algorithms: with more optional customers, less time is spent in transportation (and more in service). However, this is much stronger on the VNS side. Therefore, we can suppose that a huge part of the profit maximization is in fact achieved by optimizing transportation time, thus freeing more time for visiting other customers. A second tendancy is that the average profit per customer decreases when given more choice with optional customers. This is also true for both methods, but again, much stronger on the VNS side. One possible interpretation would be that it is more interesting to deliver several customers with small benefit than one customer with a big one; however, other explanations are also possible. In order to investigate this matter, we look more closely at the sets of unplanned customers by both methods. More precisely, we focus our attention on the customers who, given a same instance, are visited by the best insertion solution, but not by the VNS one. We analyze these data in Table 6.

This table clearly shows that the VNS usually does not remove from the solution a customer included by the best insertion. In other words, the VNS seems to add many "small" customers to the "big" ones planned during the construction, while removing almost none of them, hence improving the objective value and reducing the average profit per visited customer at the same time. This reinforces the idea that optimizing the routes with regards to a cost or duration objective is a major issue: these small customers cannot simply be inserted in the solution, else the best insertion heuristic would have done it. These insertions are therefore made possible by the route optimization procedures, i.e. the first set of shaking operators, namely *cross-exchange* combined to the 3-opt* local search.

5.2. Orienteering with time windows and team orienteering with time windows

In order to compare our VNS to existing works, and see if it can be adapted to other problems which share common features with the MuPOPTW, we adapted it to the OPTW and TOPTW. One should note here that since the OPTW and TOPTW are special cases of the MuPOPTW, our route feasibility check works for these problems without any modification. We noted, however, that since the feasibility subproblems are simpler, they are solved faster than with the MuPOPTW.

With these observations in mind, we performed the following changes:

- Reevaluated max. number of iterations to 10^6, and max. number of iterations without improvement to 5×10^5.
- Used one day per available team member.
- For the OPTW: disable the use of the *cross*-shaking operator, since the solution has only one route.

With these changes made, we can apply our VNS to the benchmark instances from the literature. We use again a 2.4 GHz CPU with 4 GB of RAM.

The first, and to our knowledge only, heuristic method for the OPTW and TOPTW is the ILS by Vansteenwegen [28]. In this PhD thesis, the author compares the ILS values to the optimal values for the OPTW found by the exact algorithm by Righini and Salani [20]. In a second stage, he generates instances for the TOPTW, by using two vehicles instead of one on the OPTW instances; twice the optimal profit value for the associated OPTW instance is used as an upper bound. In a third stage, he generates another set of instances for the TOPTW, still using OPTW instances, but with a number of vehicles which is the smallest number sufficient to visit all points. Since these OPTW instances are based on VRPTW instances for which exact solutions are known, this number is also known.

We now compare our VNS to the ILS by Vansteenwegen, and to the known optimal solution or upper bounds. For each method we report profit (the value to maximize) and CPU time in seconds. Since the ILS by Vansteenwegen is deterministic, one run is sufficient for each instance. We ran our VNS 10 times on each instance and report the minimum, maximum and average profit values. The CPU

Table 7
OPTW: results for Solomon's test problems; $n = 50$, $m = 1$.

Instance_n	Opt		ILS		VNS			
	Profit	CPU (s)	Profit	CPU (s)	Min	Max	Avg	CPU (s)
c101_50	270	0.0	270	0.3	270	270	270	44.5
c102_50	300	1.1	300	0.3	300	300	300	65.4
c103_50	320	23.0	320	0.3	320	320	320	69.0
c104_50	340	81.4	330	0.3	340	340	**340**	80.7
c105_50	300	0.0	300	0.3	300	300	300	45.2
c106_50	280	0.0	280	0.2	280	280	280	39.7
c107_50	310	0.0	310	0.2	310	310	310	53.8
c108_50	320	0.2	320	0.3	320	320	320	42.7
c109_50	340	0.9	340	0.2	340	340	340	41.7
r101_50	126	0.0	126	0.1	126	126	126	13.6
r102_50	198	0.7	195	0.2	198	198	**198**	24.5
r103_50	214	30.0	210	0.2	214	214	**214**	26.5
r104_50	227	152.7	227	0.3	227	227	227	29.3
r105_50	159	0.0	159	0.1	159	159	159	16.1
r106_50	208	0.6	203	0.2	208	208	**208**	24.8
r107_50	220	9.8	**220**	0.3	219	220	219.9	32.5
r108_50	227	410.6	223	0.2	227	227	**227**	28.2
r109_50	192	0.2	192	0.2	192	192	192	20.1
r110_50	208	1.6	208	0.2	208	208	208	23.4
r111_50	223	1.8	223	0.2	223	223	223	25.2
r112_50	226	3.1	226	0.2	226	226	226	26.0
rc101_50	180	0.0	180	0.1	180	180	180	30.8
rc102_50	230	0.7	230	0.2	230	230	230	32.8
rc103_50	240	2.2	240	0.2	240	240	240	32.5
rc104_50	270	6.1	**270**	0.2	260	270	266	38.3
rc105_50	210	0.7	200	0.2	210	210	**210**	22.2
rc106_50	210	0.8	200	0.2	210	210	**210**	33.9
rc107_50	240	3.4	230	0.1	240	240	**240**	31.1
rc108_50	250	9.3	240	0.2	250	250	**250**	33.8

Table 8
OPTW: results for Solomon's test problems; $n = 100$, $m = 1$.

Instance_n	Opt		ILS		VNS			
	Profit	CPU (s)	Profit	CPU (s)	Min	Max	Avg	CPU (s)
c101_50	320	0.1	320	0.6	320	320	320	77.9
c102_50	360	4.1	360	0.6	360	360	360	114.7
c103_50	400	270.7	390	0.8	390	400	**397**	111.2
c104_50	420	1627.7	400	0.5	410	420	**418**	108.1
c105_50	340	0.1	340	0.4	340	340	340	84.6
c106_50	340	0.2	340	0.5	340	340	340	90.2
c107_50	370	0.3	360	0.5	370	370	**370**	82.0
c108_50	370	2.0	370	0.6	370	370	370	95.6
c109_50	380	7.0	380	0.5	380	380	380	113.3
r101_50	198	0.1	182	0.3	198	198	**198**	47.8
r102_50	286	59.9	**286**	0.4	284	286	285.2	89.8
r103_50	293	672.1	286	0.5	293	293	**293**	101.0
r104_50	303	4725.2	297	0.5	303	303	**303**	110.6
r105_50	247	0.4	247	0.4	247	247	247	80.1
r106_50	293	95.5	**293**	0.5	289	293	292.2	97.5
r107_50	299	773.3	288	0.6	297	299	**298.8**	96.6
r108_50	308	2353.7	297	0.5	306	308	**307.8**	97.4
r109_50	277	3.9	276	0.5	277	277	**277**	77.6
r110_50	284	28.6	272	0.5	284	284	**284**	97.2
r111_50	297	632.7	295	0.5	297	297	**297**	104.0
r112_50	298	811.3	292	0.6	298	298	**298**	107.8
rc101_50	219	0.4	219	0.5	219	219	219	61.2
rc102_50	266	4.8	259	0.5	266	266	**266**	54.7
rc103_50	266	69.8	265	0.8	266	266	**266**	69.2
rc104_50	301	161.2	278	0.6	301	301	**301**	62.5
rc105_50	244	2.1	221	0.5	239	244	**243.2**	65.2
rc106_50	252	2.2	239	0.4	233	252	**249.4**	67.4
rc107_50	277	18.2	268	0.5	277	277	**277**	72.4
rc108_50	298	25.5	288	0.5	298	298	**298**	71.5

time reported for the VNS is the average time over 10 runs to reach a termination criterion. We find it fair to compare the average behavior of the VNS with the ILS; therefore, the highest profit value between VNS average and ILS is highlighted for each instance. The experiments by Vansteenwegen were run on an Intel Pentium 4 at 2.80 GHz CPU with 1 GB RAM, which we see as comparable to our 2.4 GHz CPU. It is noteworthy that his ILS aims at providing good solutions very quickly (a few seconds), whereas our VNS aims at

Table 9
OPTW: results for the test problems of Cordeau, Gendreau and Laporte; $m=1$.

Instance_n	Opt		ILS		VNS			
	Profit	CPU (s)	Profit	CPU (s)	Min	Max	Avg	CPU (s)
pr01_48	308	3.9	304	0.6	308	308	**308**	77.9
pr02_96	404	79.7	385	1.2	404	404	**404**	244.6
pr03_144	394	232.2	384	1.9	368	394	**391.4**	387.2
pr04_192	489	2232.4	464	4.5	477	489	**486**	541.6
pr05_240	595	> 7200	571	14.0	582	595	**589.6**	1455.2
pr06_288	501a	> 7200	538	8.0	587	591	**589**	1633.8
pr07_72	298	7.12	288	0.7	298	298	**298**	131.0
pr08_144	463	192.6	**463**	1.9	431	463	459.5	514.1
pr09_216	493	6778.3	471	4.1	446	493	**482**	920.8
pr10_288	584a	> 7200	529	8.2	555	594	**572.9**	1534.1

Table 10
Results for Solomon's test problems; $n=50$, $m=2$.

Instance_n	UB	ILS		VNS			
		Profit	CPU (s)	Min	Max	Avg	CPU (s)
c101_50	540	480	0.8	490	490	**490**	53.4
c102_50	600	520	0.9	530	530	**530**	85.9
c103_50	640	560	0.7	570	570	**570**	72.6
c104_50	680	590	0.8	590	590	590	55.6
c105_50	600	**520**	0.7	510	520	519	63.4
c106_50	560	490	0.7	500	500	**500**	34.4
c107_50	620	530	0.7	540	540	**540**	61.8
c108_50	640	540	0.9	540	540	540	72.3
c109_50	680	570	1.0	570	570	570	47.1
r101_50	252	244	0.3	246	246	**246**	10.2
r102_50	396	343	0.5	352	357	**355.7**	17.5
r103_50	428	374	0.6	375	383	**379.2**	19.6
r104_50	454	410	0.7	409	412	**411.5**	28.7
r105_50	318	282	0.3	292	292	**292**	14.9
r106_50	416	373	0.5	373	376	**373.9**	22.1
r107_50	440	385	0.6	387	398	**392.1**	25.8
r108_50	454	412	0.7	412	417	**416.4**	27.6
r109_50	384	350	0.5	364	364	**364**	18.2
r110_50	416	384	0.5	388	393	**392**	22.9
r111_50	446	397	0.5	388	406	**399.5**	26.6
r112_50	452	417	0.6	412	428	**424.2**	23.4
rc101_50	360	360	0.4	360	360	360	36.4
rc102_50	460	**430**	0.7	420	430	426	35.9
rc103_50	480	450	0.5	440	460	**455**	27.2
rc104_50	540	**500**	0.6	470	520	499	32.7
rc105_50	420	370	0.5	400	400	**400**	37.8
rc106_50	420	400	0.5	400	410	**409**	30.6
rc107_50	480	**470**	0.5	440	470	459	24.5
rc108_50	500	480	0.5	470	490	**488**	24.9

finding near-optimal solutions even if this involves spending a few more minutes.

In a first step, we compare both methods on the OPTW instances. There are three sets. The first two sets are taken from Solomon's 100 customers VRPTW instances [23], by considering either the first 50 or 100 customers. The third set uses 10 instances by Cordeau et al. for the multi-depot VRPs [7]. In total, 58 instances are considered; in all cases, profits were assigned to customers by Righini and Salani [20]. Tables 7–9 show these results. For a few instances, the exact method by Righini and Salani could not find the optimal solutions within the assigned CPU time of 2 h; those instances are signaled by the letter "a" in the tables. In such cases, a best found solution was nonetheless provided by Righini and Salani.

As expected, our VNS is slower than the ILS by Vansteenwegen on these instances. Since there are many more cases in which our VNS average profit is better than the ILS than the other way around, we can assume that our VNS provides better results on average, at the cost of spending much more CPU time. It is interesting to note that this is especially true as the instance size grows. Moreover, in many cases the minimum profit found by the VNS over 10 runs is better than the ILS value. Finally, we note that our VNS found the optimal solution for each instance at least once over 10 runs.

We now compare both methods on TOPTW instances. Those instances are in fact the previously cited OPTW instances, but using two vehicles instead of one. Tables 10–12 show these results. Since no exact value is known for these instances, twice the exact value for the OPTW case is used as an upper bound.

The previously observed tendency is still present in this second set of experiments: the VNS is slower and provides better results. It is worthy to note that the CPU times of the VNS do not increase compared to the single-vehicle case.

In a last set of experiments, the 100 customer instances by Solomon and the instances by Cordeau et al. are used, with a number of vehicles sufficient to visit every node. In this case the optimal solution is therefore known. In his thesis, Vansteenwegen emphasizes the fact that since these problems have up to 20 tours, they should be regarded as cases of intractable TOPTWs. Tables 13 and 14 show the results for these instances.

Table 11
Results for Solomon's test problems; $n=100$, $m=2$.

Instance_n	UB	ILS		VNS			
		Profit	CPU (s)	Min	Max	Avg	CPU (s)
c101_50	640	570	1.3	590	590	**590**	70.1
c102_50	720	650	1.5	650	660	**652**	83.3
c103_50	800	690	1.6	710	720	**719**	73.2
c104_50	840	740	1.5	750	760	**759**	83.6
c105_50	680	640	1.5	640	640	640	70.6
c106_50	680	620	1.4	620	620	620	83.4
c107_50	740	**670**	1.9	650	670	667	91.9
c108_50	740	680	1.9	680	680	680	88.3
c109_50	760	710	1.8	710	720	**719**	72.4
r101_50	396	330	0.7	349	349	**349**	35.4
r102_50	572	**508**	1.6	487	504	495	64.9
r103_50	586	513	1.9	513	522	**519.5**	58.8
r104_50	606	542	2.1	531	550	**542.4**	67.9
r105_50	494	442	1.6	453	453	**453**	68.3
r106_50	586	**529**	1.5	491	529	511.3	76.6
r107_50	598	527	1.5	524	536	**533.3**	71.0
r108_50	616	**550**	2.9	530	560	548.8	73.3
r109_50	554	**521**	1.7	479	505	502.4	71.5
r110_50	568	**510**	1.8	483	525	509.3	62.7
r111_50	594	538	2.0	527	544	**538.6**	61.7
r112_50	596	523	2.2	521	544	**532.8**	82.5
rc101_50	438	**421**	1.4	419	419	419	46.0
rc102_50	532	497	1.8	504	504	**504**	53.8
rc103_50	532	514	2.7	508	524	**516.7**	61.9
rc104_50	602	549	2.1	564	575	**570.8**	57.9
rc105_50	488	459	1.9	480	480	**480**	44.7
rc106_50	504	**480**	1.9	458	483	479.9	57.0
rc107_50	554	517	1.8	485	534	**521.5**	58.7
rc108_50	596	540	1.9	507	556	**541.3**	65.3

Table 12
Results for the test problems of Cordeau, Gendreau and Laporte; $m=2$.

Instance_n	UB	ILS		VNS			
		Profit	CPU (s)	Min	Max	Avg	CPU (s)
pr01_48	616	468	0.9	476	502	**484.8**	79.8
pr02_96	808	677	5.4	693	715	**705.3**	211.4
pr03_144	788	701	7.2	686	737	**721.1**	363.2
pr04_192	978	881	24.8	899	925	**906.2**	562.0
pr05_240	1190	1019	22.0	1053	1076	**1062.4**	963.0
pr06_288	1076	985	30.6	974	1021	**999.6**	914.4
pr07_72	596	**552**	2.4	524	566	549.7	122.8
pr08_144	926	796	10.7	765	823	**797.9**	333.5
pr09_216	986	841	20.7	796	905	**850.9**	578.7
pr10_288	1168	1033	56.8	1038	1129	**1073.2**	1060.0

Our VNS still provides better results with longer CPU times on these sets. Interestingly enough though, it has shorter CPU times than with the previous, supposedly easier instances. The reason is that it reaches a local (sometimes also global) optimum faster, and then spends 5×10^5 "quick" iterations without improvement, until the termination criterion is reached. Since a solution has many tours, the pool of non-planned nodes is very small, so our second local search, which tries to greedily insert nodes into the solution, is very fast. Since less time is spent in local search, the VNS is faster overall. We also note that unlike for previous instance sets, there are many instances for which the VNS never finds the optimal solution.

Regarding the OPTW and TOPTW, we conclude that our VNS provides very good solutions, with computational times taking up to 30 min. This is better but slower than the other existing heuristic method for these problems. Both methods have advantages, one being very quick without sacrificing efficiency, and the other providing very good solutions but needing more CPU time.

5.3. *Orienteering and team orienteering*

Since the OP and TOP are special cases of the MuPOPTW, we found it interesting to also apply our VNS to benchmark instances for these problems. There is a variety of recent publications providing good results for the TOP, and in such a contest it is hard to provide a competitive method. In order to apply our VNS to these problems, we kept the previously described modifications performed for the OPTW and TOPTW, and added the following:

- Replaced the 3-opt* local search, more adapted to time windows, by a 2-opt local search.
- Removed the advanced time feasibility check; in order to check that a route does not exceed the duration limit, a simple transportation time calculation is enough since we have no time window, therefore no waiting time.

Table 13
Results for Solomon's test problems; a feasible solution visiting every customer exists.

Instance_*n*	*m*	Opt.	ILS		VNS			
			Profit	CPU (s)	Min	Max	Avg	CPU (s)
c101_50	10	1810	1740	5.7	1800	1810	**1809**	21.7
c102_50	10	1810	1800	6.6	1810	1810	**1810**	19.8
c103_50	10	1810	1810	3.7	1810	1810	1810	18.8
c104_50	10	1810	1810	3.1	1810	1810	1810	16.8
c105_50	10	1810	1770	5.2	1810	1810	**1810**	22.7
c106_50	10	1810	1750	4.8	1800	1810	**1808**	23.5
c107_50	10	1810	1800	6.8	1810	1810	**1810**	21.9
c108_50	10	1810	1810	4.0	1810	1810	1810	19.6
c109_50	10	1810	1810	3.3	1810	1810	1810	16.1
r101_50	19	1458	1455	10.2	1453	1458	**1455.8**	9.2
r102_50	17	1458	1450	5.9	1446	1455	**1451.1**	9.6
r103_50	13	1458	1445	10.0	1450	1455	**1453.3**	16.9
r104_50	9	1458	1410	10.8	1424	1453	**1443.5**	28.7
r105_50	14	1458	1445	11.2	1444	1455	**1450.1**	16.9
r106_50	12	1458	1438	11.2	1447	1455	**1451.9**	20.2
r107_50	10	1458	1427	11.9	1444	1455	**1449.7**	26.9
r108_50	9	1458	1431	5.4	1448	1457	**1453.2**	26.6
r109_50	11	1458	1420	10.2	1441	1453	**1448.3**	23.1
r110_50	10	1458	1399	7.1	1448	1455	**1450.5**	26.4
r111_50	10	1458	1425	10.8	1439	1455	**1449.2**	25.5
r112_50	9	1458	1420	7.7	1444	1455	**1451.9**	27.9
rc101_50	14	1724	1672	9.5	1690	1717	**1704.5**	18.4
rc102_50	12	1724	1668	11.8	1688	1708	**1696**	22.0
rc103_50	11	1724	1690	12.2	1710	1724	**1715.8**	22.1
rc104_50	10	1724	1706	6.8	1716	1724	**1722.7**	21.5
rc105_50	13	1724	1674	7.4	1688	1709	**1698.2**	18.1
rc106_50	11	1724	1672	8.0	1676	1711	**1694.8**	24.5
rc107_50	11	1724	1701	7.7	1701	1724	**1721**	19.8
rc108_50	10	1724	1692	7.3	1716	1724	**1721.8**	21.7

Table 14
Results for the test problems of Cordeau, Gendreau and Laporte; a feasible solution visiting every customer exists.

Instance_*n*	*m*	Opt.	ILS		VNS			
			Profit	CPU (s)	Min	Max	Avg	CPU (s)
pr01_48	3	657	589	0.7	606	616	**612.9**	69.6
pr02_96	6	1220	1165	4.1	1182	1203	**1195.3**	85.0
pr03_144	9	1788	1722	26.3	1753	1765	**1759**	86.0
pr04_192	12	2477	2420	43.4	2453	2472	**2466.4**	81.2
pr05_240	15	3351	3297	58.4	3342	3351	**3349.5**	71.7
pr06_288	18	3671	3630	177.0	3668	3671	**3670.1**	83.5
pr07_72	5	948	910	3.6	922	934	**928.3**	69.5
pr08_144	10	2006	1985	22.5	2002	2006	**2005**	63.2
pr09_216	15	2736	2732	38.8	2732	2736	**2735.6**	54.6
pr10_288	20	3850	3850	95.6	3850	3850	3850	52.4

In a first stage, we used the OP instances by Tsiligirides [26]. They consist of three problem sets, each with various duration limits, thus resulting in several test instances per problem sets. One should note that there is a mistake in the instances by Tsiligirides, and therefore two different versions of an instance exist; since all the later works on the instances by Tsiligirides used the version with the mistake, we also use it. See [6] for more details on this mistake and consequences. Two other problem sets were added by Chao [3], also with various duration limits. In total, there are 89 test instances, using five different distance matrices. The literature on these instances is relatively old. In a working paper on bi-objective orienteering, Schilde et al. [22] also provide new results on these single-objective instances, and outperform all previous methods. We decided to compare our VNS to both the old literature and these new results. In order to do so, we run the VNS on each instance 10 times, and looked at minimum, maximum and average values. Result tables do not provide any insight, and are therefore not included in the paper; these results can be downloaded at http://prolog.univie.ac.at/research/OP/. We give here the following synthesis:

- The best known solution was always found at least once over the 10 runs.
- The VNS minimum value (i.e. worst case) was different from the best known value for only two instances. In other words, the VNS finds the best known solution almost systematically.
- More precisely, the best known solution was not found on five runs over 890.
- No new best solution was found.

Many of these solutions might in fact be optimal. In addition, we note that the VNS took on average 4 s to converge, with a maximum of 82 s, 90% of the cases below 11 s, and 97% below 30 s.

We now focus on the more recently studied TOP instance sets. We use the 353 benchmark instances by Chao et al. [4]. These instances are divided into seven classes, called p1–p7. We compare the results of our VNS to five major previous works: the heuristic

Table 15
Average gap (in %) to the best known solution for various algorithms for the TOP.

Instance class	Fast VNS		Slow VNS		KAF		Other literature		MuPOPTW VNS		
	Min	Max	Min	Max	Avg	Max	TMH	CGW	Min	Max	Avg
p1	0.24	0.24	0.24	0.24	0.24	0.24	0.35	0.56	0.55	0.24	0.36
p2	0.09	0.09	0.09	0.09	0.09	0.09	0.25	0.25	0.09	0.09	0.09
p3	0.10	0.00	0.00	0.00	0.00	0.00	0.65	1.36	0.61	0.00	0.18
p4	1.17	0.23	0.30	0.02	1.76	0.26	1.89	4.18	4.15	0.92	2.32
p5	0.36	0.22	0.21	0.19	0.68	0.20	1.02	0.91	1.43	0.40	0.79
p6	0.26	0.14	0.20	0.14	1.04	0.14	0.63	0.38	0.29	0.14	0.18
p7	1.24	0.28	0.30	0.03	0.36	−0.01	0.84	2.29	3.14	0.50	1.56
Total	0.54	0.18	0.20	0.10	0.59	0.13	0.87	1.58	1.68	0.36	0.89

One line = one class of instances.

Table 16
Average and Max CPU time (in seconds) to find the last incumbent solution, for the team orienteering problem.

Instance class	Fast VNS		Slow VNS		Other literature			MuPOPTW VNS		
	Avg	Max	Avg	Max	TMH	CGW	KAF	Avg	Max	Max-Avg
p1	0.13	1.00	7.78	22.00	N.A.	15.41	7.9	2.8	14.2	10.7
p2	0.00	0.00	0.03	1.00	N.A.	0.85	3.8	1.3	4.3	3.2
p3	0.15	1.00	10.19	19.00	N.A.	15.37	8.5	4.2	18.3	14.7
p4	22.52	121.00	457.89	1118.00	796.70	934.80	51.1	48.0	195.8	141.0
p5	34.17	30.00	158.93	394.00	71.30	193.70	25.2	11.2	44.0	37.2
p6	8.74	20.00	147.88	310.00	45.70	150.10	20.3	8.7	42.7	34.9
p7	10.34	90.00	309.87	911.00	432.60	841.40	44.7	29.9	124.8	98.1

One line = one class of instances.

from the TOP original paper by Chao et al. [4], which we will now call CGW, the tabu search by Tang and Miller-Hooks [25], which we will call TMH, two of the four metaheuristics by Archetti et al. [1], called *fast VNS*, and *slow VNS*, and the recent Ant Colony Optimization (ACO) algorithm by Ke et al. [12], which we call KAF. Concerning the paper by Archetti et al., the algorithm called *slow VNS* outperformed all three other heuristics, while the one called *fast VNS* was considered by the authors to represent the best trade-off between performance and computational time; for the sake of clarity, we disregard their two other algorithms. A recent paper by Vansteenwegen et al. describes a guided local search (GLS) for the OP and TOP [27]. The method is very fast, and provides solutions almost as good as the ones by Archetti et al., and Tang and Miller-Hooks; it is not compared with the recent ACO by Ke et al. It is clearly presented as an alternative aim, which is to provide solutions that are usually slightly less good than with the other methods, although better on some occasions, with much less computational resources. Since our VNS was designed with aims similar to those of these other methods, we do not compare it to this GLS.

These last two papers yield until now the best known results for the TOP. In the paper by Archetti et al. [1], a tabu search is embedded as local search in a VNS. The whole algorithm exploits the problem structure in ways that are impossible in our case though. In fact they use the whole set of customers in a traditional vehicle routing problem fashion, and then select the set of routes providing the best solution for the associated TOP, while respecting the fleet size limit. To allow this framework to work, concepts such as *feasible* and *admissible* solution are designed, and repair procedures are produced. As a whole it works very well, but we do not see how to adapt these ideas to the MuPOPTW in an efficient manner. More precisely, the feasibility subproblem handling is not likely to be easy to integrate; to a lesser extent, the mandatory customers handling also represents a challenge. Because of these differences in problem structures between TOP and MuPOPTW, we believe it is not straightforward to adapt the algorithms by Archetti et al. to our case. The same holds concerning the paper by Ke et al.: it is again not obvious to us how one could efficiently integrate mandatory customers, and most importantly the issues linked to multiple time windows and the feasibility subproblem.

As reported by Archetti et al., on 33 instances their initial solution was optimal; on the remaining 320 instances, all tested algorithms provided the exact same result for 121 instances. This allowed them to disregard this total of 154 instances and focus on the 199 remaining ones. However, these properties are no longer true in our case, so we consider the original 353 instances, for which they also provided data. Ke et al. also used these 353 instances. Tang and Miller-Hooks and Chao et al. solved each instance once. Archetti et al. ran each of their heuristics 3 times, and reported the worst and best case (resp. min and max). Ke et al. report the best and average objective over 10 runs. We ran our VNS 10 times, and report the worst, best, and average values (resp. min, max, and avg). We summarize these results in Table 15, by considering the average gap to the best known solution, per class of instance; extensive results can again be downloaded at http://prolog.univie.ac.at/research/OP/. We use the best known solutions provided by Archetti et al. as a reference set; since some of the results presented here are newer, it is possible to have negative gaps, namely in the works of Ke et al. and our VNS. We sort these results by class, with one line per class of instances.

Table 15 shows how our VNS for the MuPOPTW behaves with standard TOP instances, compared to tailor-made algorithms for the TOP. It is not so surprising to see that the methods by Archetti et al. and Ke et al. are the two best ones. However, the goal here is to measure the adaptability of our VNS to problems sharing common features with the MuPOPTW. With regards to this objective, we note that our VNS has an average gap to the best known solution of less than 1%. More generally, the performances of this VNS are relatively close to those of the best existing methods for the TOP. It is also of interest to see that the average value is always closer to the best than to the worst.

We now compare average and maximum computational time required by the various algorithms. More precisely, we look at the duration needed by the algorithms before reaching a stopping

Table 17
New best known solutions found during all our experiments, and computational times.

Instance	Former best known	New best known	Comp. time (s)
p4.2.q	1265	1267	8788
p4.2.r	1288	1292	9851
p4.3.q	1252	1253	5157
p4.4.p	1120	1124	2183
p4.4.r	1211	1216	251
p7.3.t	1118	1120	4271

condition. These values are reported in Table 16. Chao et al. and Tang and Miller-Hooks provide only one run per instance. The algorithm by Chao et al. was run on a SUN 4/730 Workstation 25 MHz; the one of Tang and Miller-Hooks on a DEC Alpha XP1000 computer at 667 MHz. Ke et al. ran their algorithm on a 3.0 GHz Intel CPU. The authors report maximal values, but they are in fact the maximum average, i.e. for each instance the average time over 10 runs is taken, and then the maximum of these average times over a whole class is reported. For the sake of completeness, we also report this value for our VNS, which we note Max-Avg. All algorithms by Archetti et al. were run on a 2.80 GHz Intel Pentium 4 CPU, with 1 GB of RAM.

Finally, we note that during all our experiments we tried different computational times and stopping conditions; those brought us six new best known solutions, so we report them in Table 17 for the sake of completeness. We also report the computational time taken to provide each of these solutions. Most of these were found several times, in which case we report the best behavior, i.e. the shortest computational time from all those executions which led to this same objective value.

6. Conclusion

In this paper, we introduced the multi-period orienteering problem with multiple time windows, which is a problem encountered by sales representatives when they want to schedule their working week. This problem is too hard for contemporary commercial solvers, at least for interesting size instances. We therefore developed a metaheuristic (VNS) to provide good solutions in reasonable amounts of time. The MuPOPTW also includes a feasibility subproblem, featuring multiple time windows, which is a new and original aspect; we studied this subproblem, and produced an exact algorithm to solve it in polynomial time. Our VNS makes intensive use of this feasibility check algorithm. Finally, we compared this method to a state-of-the-art commercial solver, and showed that for small instances the solver is outperformed. For other instances the solver is not able to provide anything, so we had to compare our VNS to other algorithms. After minor modifications, we adapted it to the standard OPTW, TOPTW, OP and TOP benchmark instances, and compared it with the best known methods for these problems. Results showed that our tailor-made VNS algorithm for the MuPOPTW is a viable option when solving all kinds of orienteering problems, with or without time windows. The designed VNS algorithm will be integrated in a commercial decision support system.

Acknowledgments

We would like to thank Guenter Kiechle for the project management and Andreas Wagner for implementing the information system and the user interface. We also wish to thank the two anonymous referees for their valuable comments, which helped us improving the quality of this paper. Furthermore we thank Zoltan Daroczi for providing the real-world data. Financial support from the Austrian Research Promotion Agency (FFG - project Bridge #814299) is gratefully acknowledged.

Appendix A. Details on route feasibility check

This appendix deals with the problem of checking if a certain route is feasible or not. We note here that a similar problem has been studied in the past by Savelsbergh [21]. The paper deals with the single time window case, and provides an $O(n)$ algorithm to compute the minimum duration of a route, where n is the number of customers in the route. However, the algorithm of Savelsbergh does not handle multiple time windows, and it does not seem to be generalizable to handle them; therefore we developed a new algorithm.

In the first part we formalize the problem, then we propose some preprocessing steps and finally we give an algorithm that solves the problem. We assume that the triangular inequality holds for the distance matrix and that irrelevant data is already eliminated; therefore the following inequalities are valid for all i, j, k.

$$d_{ij} \leq d_{ik} + d_{kj} \tag{12}$$

$$d_{oi} + s_i + d_{io} \leq \max_{t\in\mathscr{T}}\{\aleph^t\} \tag{13}$$

$$\exists t \in \mathscr{T} : d_{o_s^t i} + s_i + d_{io_e^t} \leq \aleph^t \tag{14}$$

$$\exists t \in \mathscr{T} : a_i^t \geq d_{o_s^t i} + s_i \tag{15}$$

$$\exists t \in \mathscr{T} : b_i^t \leq \aleph^t - d_{io_e^t} \tag{16}$$

For the sake of simplicity, we suppose that the customers in the route are now named $\{1,\ldots,n\}$. Since we are working on a route assigned to a given day of the horizon, t becomes a fixed value. Therefore we do not mention the index t in this section. For a fixed sequence of customers the travel time can be integrated into the service time; this transformation is already described in Section 4, and we use this transformed service times and time windows from now on. To assert that there exists a schedule of maximum length $\aleph$ that respects service times and the time windows we reformulate constraints (7)–(10). According to this notation, the route is a feasible sequence iff the following constraints are satisfied:

$$\sum_{k\in\Theta_i} y_i^k = 1 \quad \forall i \in \{1,\ldots,n\} \tag{17}$$

$$\alpha_i + s_i \leq \alpha_{i+1} \quad \forall i \in \{1,\ldots,n\} \tag{18}$$

$$(a_i^k + s_i)y_i^k \leq \alpha_i \leq b_i^k + \mathscr{B}(1-y_i^k) \quad \forall i \in \{1,\ldots,n\}\ k \in \Theta_i \tag{19}$$

$$\alpha_n - (\alpha_1 - s_1) \leq \aleph \tag{20}$$

Constraints (17) ensure that exactly one time window is selected for each customer in the tour; constraints (18) ensure that service at a given customer does not start before the end of service at the previous customer; constraints (19) link, for each customer, the time at which service starts to the selected time window for this customer. Constraint (20) enforces that the route duration is not greater than the maximal duration allowed.

We present an algorithm that solves the problem in polynomial time and we start with presenting the preprocessing steps. We note that in the single time window case the preprocessing steps implicitly construct two candidate solutions in linear time.

A.1. Preprocessing step

To determine if a certain route is feasible we have to find out if there exists a possibility to (i) visit all its customers within their time windows, subject to service and transportation time and (ii) do so while not exceeding the maximum duration allowed for this route. The preprocessing algorithm will determine whether there exists a feasible selection of time windows, without taking into account total duration, i.e. constraints (10) and (11). These constraints will be considered in a further stage. The preprocessing tightens the time windows by exploiting the fact that service cannot start at a given customer before the earliest end of service at the previous customer; analogously, this property holds backward in time, and also allows to tighten the end of time windows. This preprocessing is therefore done in two passes:

1. Algorithm 2 tightens the beginning of time windows. We assume that $[k]_i$ is a ranking of a_i^k. i.e.: $a_i^{[1]_i} \leq \cdots \leq a_i^{[|\Theta_i|]_i}$. E.g. $\tau_i^{[1]_i}$ defines the time window in Θ_i with the earliest beginning of time window.
2. Algorithm 3 tightens the end of time windows. We assume that $[-k]_i$ is the reversed ranking of $[k]_i$. i.e.: $[-k]_i := [|\Theta_i| - k + 1]_i$. E.g. $\tau_i^{[-1]_i}$ defines the time window in Θ_i with the latest end of time window.

Algorithm 2. Preprocessing step 1: tighten beginning of time windows.

```
1:  $i \leftarrow 1$
2:  while $i < n \wedge \Theta_i \neq \emptyset$ do
3:    if $(a_i^{[1]_i} + s_i > b_i^{[1]_i}) \vee (a_i^{[2]_i} + s_i \leq b_i^{[2]_i} \wedge a_i^{[2]_i} + s_i \leq a_{i+1}^{[1]_{i+1}})$ then
4:      $\Theta_i \leftarrow \Theta_i \backslash \{\tau_i^{[1]_i}\}$
5:    else
6:      if $a_i^{[1]_i} + s_i < a_{i+1}^{[1]_{i+1}}$ then
7:        $a_i^{[1]_i} \leftarrow \min\{a_{i+1}^{[1]_{i+1}} - s_i, b_i^{[1]_i} - s_i\}$
8:      else if $a_i^{[1]_i} + s_i > a_{i+1}^{[1]_{i+1}}$ then
9:        $a_{i+1}^k \leftarrow \max\{a_i^{[1]_i} + s_i, a_{i+1}^k\}\ \forall k \in \Theta_{i+1}$
10:     endif
11:     $i \leftarrow i + 1$
12:   end if
13: end while
14: return $i = n$
```

Algorithm 3. Preprocessing step 2: tighten end of time windows.

```
1:  $i \leftarrow n$
2:  while $i > 1 \wedge \Theta_i \neq \emptyset$ do
3:    if $(b_i^{[-1]_i} - s_i > a_i^{[-1]_i}) \vee (b_i^{[-2]_i} - s_i \geq a_i^{[-2]_i} \wedge b_i^{[-2]_i} - s_i \geq b_{i-1}^{[1]_{i-1}})$ then
4:      $\Theta_i \leftarrow \Theta_i \backslash \{\tau_i^{[-1]_i}\}$
5:    else
6:      if $b_i^{[-1]_i} - s_i > b_{i-1}^{[-1]_{i-1}}$ then
7:        $b_i^{[-1]_i} \leftarrow \max\{b_{i-1}^{[-1]_{i-1}} + s_i, a_i^{[-1]_i} + s_i\}$
8:      else if $b_i^{[-1]_i} - s_i < b_{i-1}^{[-1]_{i-1}}$ then
9:        $b_{i-1}^k \leftarrow \min\{b_i^{[-1]_i} - s_i, b_{i-1}^k\} \quad k \in \Theta_{i-1}$
10:     endif
11:     $i \leftarrow i - 1$
12:   end if
13: end while
14: return $i = 1$
```

These two algorithms iteratively perform valid operations on time windows for a given route. We now explain these operations for Algorithm 2. Lines 3 and 4 aim at deleting "useless" time windows. A time window can be seen as useless in two cases:

- If the window duration is lower than the service time for this customer. This is matched by the first clause of line 3.
- If using this time window brings no benefit compared to using another time window for the same customer. This is matched by the second clause of line 3: if it is possible to use the second time window for customer *i* and still finish service at *i* before the beginning of the earliest time window at $i+1$, then the first time window for *i* is useless. More intuitively, this means that given a solution using time window 1 for customer *i*, there is always a solution which is at least as good and uses time window 2.

The following lines (6–9) aim at tightening "useful" time windows, by removing useless parts of them. Lines 6 and 7 match those cases where the beginning part of a time window is useless, since starting service during this part will cause some unnecessary waiting time, because the earliest time window for the following customer starts later. In other terms, for each solution using this early part of this time window for customer *i*, there exists another solution, at least as good, starting service at customer *i* later in this same time window. Lines 8–9 aim at tightening the beginning of time windows at customer $i+1$, by considering the earliest time at which service at customer *i* can be finished: all time windows at customer $i+1$ should never start before this earliest finishing time at customer *i*.

Within the algorithm time windows may be deleted, but that does not require a recalculation of the ranking. Since the order is kept, the ranks of the remaining time windows are shifted ($[k]_i \leftarrow [k+1]_i$).

If Algorithms 2 and 3 do not terminate prematurely we end up with tight and feasible bounds, i.e: $\alpha_i = a_i^{[1]_i} + s_i$ is a feasible solution that finishes as early as possible. Analogously, $\alpha_i = b_i^{[-1]_i}$ is a solution that finishes as late as possible. We note that the complexity of the above algorithm is $O(\sum_{i=1}^n |\Theta_i|)$, and the sorting effort to calculate $[k]_i$ and $[-k]_i$ consumes $O(|\Theta_i| \ln(|\Theta_i|))$. In the single time window case we get $O(n)$ time complexity for the preprocessing. We also remark that we only tightened or deleted time windows, therefore the time windows are still non-overlapping.

A.2. Definitions and properties

We assume that the time windows are ordered, i.e.: $\forall_{i=1}^n\ \forall k \in \Theta_i : [k]_i = k$. Now we define the minimum route duration problem, which is an optimization variant of the route feasibility problem; constraint (20) is replaced by the objective function to minimize the total route duration:

$$z^* = \min(\alpha_n - \alpha_1) \tag{21}$$

s.t. ((17), (18), (19)).

Obviously the route is feasible iff $z^* < \aleph - s_1$. We give a constructive algorithm that solves the optimization variant. In the following, we identify a solution to the minimum route duration problem as follows:

$$\alpha = (\alpha_1, \ldots, \alpha_n) \tag{22}$$

where for each node index $i \in \{1, \ldots, n\}$, α_i is the departure time at customer *i* and we define the functions $\beta_i : \alpha_i \mapsto \beta_i(\alpha_i) \in \Theta_i$ as the corresponding time window index. Since the time windows are disjoint we know that β_i is well defined. Since α_n is the finishing time of the whole route, we can express route duration as a function of α: $l(\alpha) := \alpha_n - \alpha_1 + s_1$.

In the following, we use the concept of *dominant* solution, as defined in Section 4: a solution is called *dominant* if there is no better solution with the same finishing time. It is quite obvious that any optimal solution is dominant. The strategy of the algorithm is to successively traverse dominant solutions. We start with the following solution: $\alpha_{init} = (a_i^1 + s_i)_{i=1}^n$. Because of the preprocessing steps we can assume that this solution is feasible and there is no solution that finishes earlier than $a_i^n + s_n$. The initial dominant solution is found by successively pulling the visits of all preceding customers towards $a_i^n + s_n$. Algorithm 4 constructs new candidate solutions, based on the following idea. Suppose that we have a given dominant solution ϑ and customer *i* is the last customer with a waiting time in the route

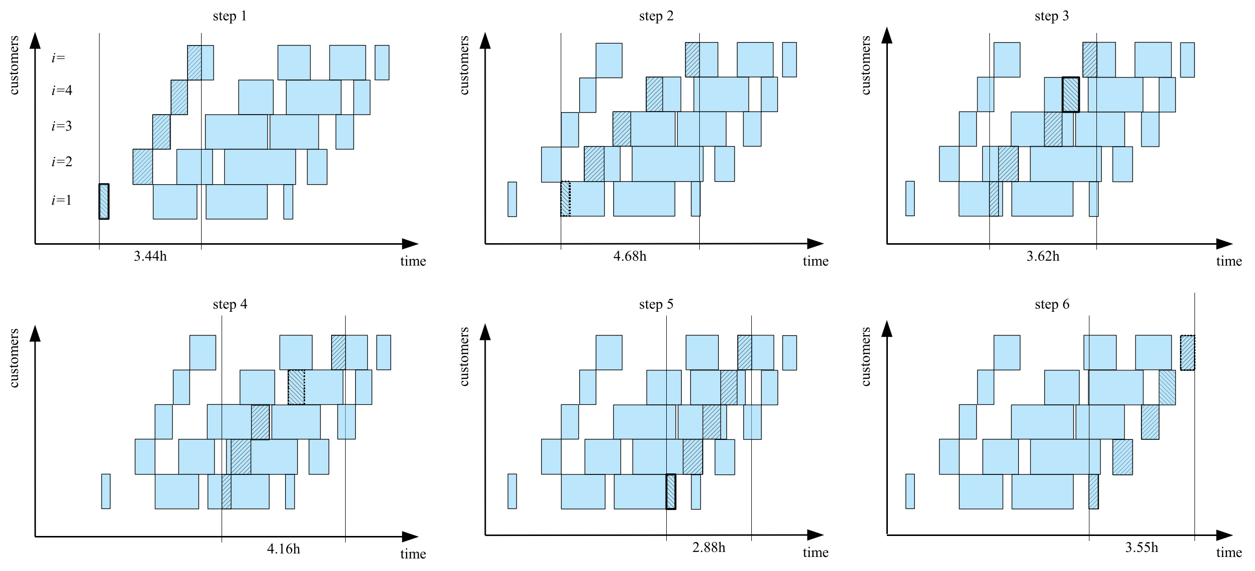

Fig. 2. Example with four time windows.

(otherwise there are no waiting times and the route is obviously optimal in terms of duration). Then we can assume that the departure time of the predecessing customer is defined by the border of the time window τ_{i-1}: $\alpha_{i-1} = b_{i-1}^{\beta(\alpha_{i-1})}$. If we want to consult a dominant solution that finishes later, we have to switch from time window $\beta(\alpha_{i-1})$ to the next time window $\beta(\alpha_{i-1})+1$. We define a *succeeding candidate solution* α' of α, claiming the following properties:

- α' is dominant.
- If $\hat{\alpha}$ dominant and $\hat{\alpha}_n \in [\alpha_n, \alpha'_n]$ then $l(\hat{\alpha}) \geq \min(l(\alpha), l(\alpha'))$.

We also suppose that we have stored the "so far" best solution α^*:

- α^* is dominant.
- If $\hat{\alpha}$ dominant and $\hat{\alpha}_n < \alpha^*_n$ then $l(\alpha^*) \leq l(\hat{\alpha})$.

It is obvious that $\alpha'_{i-1} > b_{i-1}^{\beta(\alpha_{i-1})}$ (otherwise α'_n would increase, but $\alpha'_1 = \alpha_1$ and therefore the duration of the route would increase); therefore it is sufficient to start checking the "nearest neighboring" solutions $\bar{\alpha}$, where $\bar{\alpha}_{i-1} = a^{\beta(\alpha_{i-1}+1)} + s_{i-1}$. Based on this candidate we construct an improved solution $\bar{\alpha}^*$ that also has $\bar{\alpha}^*{}_{i-1} = \bar{\alpha}_{i-1}$ but with the shortest duration. We claim that the finishing time $\bar{\alpha}^*_n$ defines the *succeeding candidate solution* for α. Now we formalize the proposed Algorithm 4.

Algorithm 4. Multi TW Min route duration.
```
$\alpha \leftarrow (a_i^1 + s_i)_{i=1}^n$
$\alpha^* \leftarrow CalculateDominantSolution(\alpha)$
$i \leftarrow LatestWaitingCustomer(\alpha)$
$v \leftarrow i > 0 \wedge \beta(\alpha_{i-1}) < |\Theta_{i-1}|$
while $v$ do
  $\bar{\alpha}^* \leftarrow SwitchTimeWindow(\alpha, i)$
  $\alpha \leftarrow CalculateDominantSolution(\bar{\alpha}^*_n)$
  if $l(\alpha) < l(\alpha^*)$ then
    $\alpha^* \leftarrow \alpha$
  end if
  $i \leftarrow LatestWaitingCustomer(\alpha)$
  $v \leftarrow i > 0 \wedge \beta(\alpha_{i-1}) < |\Theta_{i-1}|$
end while
return $\alpha^*$
```

The procedure *LatestWaitingCustomer*(α) returns the last customer with a waiting time, i.e. $i = \arg\max(j : \alpha_j - s_j > \alpha_{j-1})$. If there is no such customer then the procedure returns zero. For a given $i > 0$ where $\beta(\alpha_{i-1}) < |\Theta_{i-1}|$ the procedure *SwitchTimeWindow*(α, i) returns the solution $\bar{\alpha}^*$ that has minimal duration and departs at customer $i-1$ one time window later than in α but as early as possible, i.e. $\tau_{i-1}^{\beta(\alpha_{i-1})+1}$. This is realized in the following simple procedure that pulls the customers to the fixed service of customer $i-1$:

- $\bar{\alpha}^*_{i-1} = a_{i-1}^{\beta(\alpha_{i-1})+1} + s_{i-1}$.
- For all customers $j > i-1$: visit them as early as possible.
- For all customers $j < i-1$: visit them as late as possible.

We note that $a_{i-1}^{\beta(\alpha_{i-1})+1} + s_{i-1}$ is a feasible departure time since $a_{i-1}^{\beta(\min(\Theta_{i-1}))} + s_{i-1}$ and $b_{i-1}^{\beta(\max(\Theta_{i-1}))}$ are also feasible (preprocessing). The procedure *CalculateDominantSolution*($\bar{\alpha}^*_n$) returns a dominant solution α' with $\alpha'_n = \bar{\alpha}^*_n$. This solution is at least as short as $\bar{\alpha}$ while $\alpha'_n > \bar{\alpha}_n$. It is important to note that in case of dominant solution α we implicitly optimize subsequences $\{1, \ldots, j\}$ $(j \leq n)$, here called tails. For a fixed j the tail cannot be shortened. We also remark that if α and α' are both dominant with different finishing times $\alpha_n < \alpha'_n$ then we know that $\forall_{j=1}^n : \alpha_j \leq \alpha'_j$. Furthermore, if additionally $l(\alpha) \leq l(\alpha')$ then $\forall_{j=1}^n : \alpha_j < \alpha'_j$. Remember that the solution α and its candidate successor α' are both dominant and let us suppose that $\hat{\alpha}$ is a dominant solution that finishes earlier than α' and later than α, then $\alpha_{i-1} \leq \hat{\alpha}_{i-1} \leq \alpha'_{i-1}$. If $\hat{\alpha}_{i-1} = \alpha_{i-1}$ then we can assume that both have the same tail and therefore if $\alpha_n < \hat{\alpha}_n \Rightarrow l(\alpha) < l(\hat{\alpha})$. If $\hat{\alpha}_{i-1} = \bar{\alpha}^*_{i-1}$ then clearly $l(\alpha') \leq l(\hat{\alpha})$. Finally, we show that $\bar{\alpha}^*_{i-1} < \hat{\alpha}_{i-1} < \alpha'_{i-1}$ and $l(\alpha') > l(\hat{\alpha})$ is a contradiction: We define the following solution $\breve{\alpha} = (\bar{\alpha}^*_1, \ldots, \bar{\alpha}^*_{i-1}, \hat{\alpha}_{i-1}, \ldots, \hat{\alpha}_n)$ which is feasible and obviously shorter than $\bar{\alpha}^*$, contradicting the construction of $\bar{\alpha}^*$. We summarize the results for Algorithm 4 and we define $m = \sum_{i=1}^n (|\Theta_i| - 1)$:

Theorem A.1. *Algorithm 4 finds the optimal solution for the minimum duration problem* (21), *and its complexity is* $O((1+m)*n)$.

Proof. We already argumented that Algorithm 4 returns a schedule of minimum duration. For the complexity proposition we note that each time window and customer may be chosen at most once and the procedures *LatestWaitingCustomer*(), *SwitchTimeWindow*() and *CalculateDominant Solution*() have linear time complexity ($O(n)$). We also call *CalculateDominantSolution*() and *LatestWaitingCustomer*() in the initialization. □

Remark A.2. In the single time window case $m=0$, therefore Algorithm 4 solves the problem in $O(n)$.

Fig. 2 gives an example with four time windows for five customers ($i=1,\dots,5$). For each step of the algorithm we depicted a GANTT chart, starting in the upper left corner. Each line represents a customer, and the gray rectangles on such a line represent available time windows for a given customer. Dashed zones represent the selected time frame for service, and therefore a solution consists of one dashed zone per line. Bold-bordered rectangles represent the service at the last customer which has been output by function *LatestWaitingCustomer*(). The bold dotted rectangles mark the service after switching a time window; the beginning of this service is stored in the variable called α in Algorithm 4. We now describe the six steps of the example:

1. The algorithm starts by examining the "first" dominant solution, i.e. the one using the first time window of customer 1.
2. The dominant solution associated to time window 2 of customer 1 is investigated; first all services are shifted to the left from the first customer till the last…
3. …then once the earliest ending time for the last customer is obtained, all services are shifted to the right from last to first customer. This dominant solution has a total duration of 3.62 h.
4. Similarly, dominant solution for the third time window is computed, first by shifting service to the left from first to last customer…
5. …and then to the right from last to first customer. This dominant solution has a total duration of 2.88 h, which is the best found so far.
6. This step gives no improvement and the algorithm terminates, since there is no latest dominant solution to investigate.

Remark A.3. It is not necessary to construct $\bar{\alpha}^*$, we only need to calculate $\bar{\alpha}^*_n$. Furthermore, using the procedure for the feasibility check we can simplify Algorithm 4 to Algorithm 5. The procedure *GetFT*(α, i) returns the earliest finishing time when leaving customer $i-1$ at α_{i-1}.

Algorithm 5. Multi TW Feasibility Check.

```
$\alpha \leftarrow (a_i^1 + s_i)_{i=1}^n$
$\alpha^* \leftarrow CalculateDominantSolution(\alpha)$
$i \leftarrow LatestWaitingCustomer(\alpha)$;
$v \leftarrow (l(\alpha) > \aleph) \wedge (i > 0) \wedge (\beta(\alpha_{i-1}) < |\Theta_{i-1}|)$;
while $v$ do
    $\bar{\alpha}^*_n \leftarrow GetFT(\alpha, i)$;
    $\alpha \leftarrow CalculateDominantSolution(\bar{\alpha}^*_n)$;
    if $l(\alpha) < l(\alpha^*)$ then
        $\alpha^* \leftarrow \alpha$;
    end if
    $i \leftarrow LatestWaitingCustomer(\alpha)$;
    $v \leftarrow (l(\alpha^*) > \aleph) \wedge (i > 0) \wedge (\beta(\alpha_{i-1}) < |\Theta_{i-1}|)$;
endwhile
return $(l(\alpha^*) < \aleph)$;
```

References

[1] Archetti C, Hertz A, Speranza MG. Metaheuristics for the team orienteering problem. Journal of Heuristics 2005;39(2):188–205.
[2] Bräysy O. A reactive variable neighborhood search for the vehicle-routing problem with time windows. INFORMS Journal on Computing 2003;15:347–68.
[3] Chao I-M. Algorithms and solutions to multi-level vehicle routing problems. PhD thesis, College Park, MD, USA; 1993. [Chairman-Bruce L. Golden].
[4] Chao I-M, Golden B, Wasil EA. A fast and effective heuristic for the orienteering problem. European Journal of Operational Research 1996;88(3):457–89.
[5] Chao I-M, Golden B, Wasil EA. The team orienteering problem. European Journal of Operational Research 1996;88(2):101–11.
[6] Chao I-M, Golden BL, Wasil EA. A fast and effective heuristic for the orienteering problem. European Journal of Operational Research 1996;88(3):475–89.
[7] Cordeau JF, Gendreau M, Laporte G. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 1997;30:105–19.
[8] Feillet D, Dejax P, Gendreau M. Travelling salesman problems with profits. Transportation Science 2005;39(2):188–205.
[9] Hansen P, Mladenović N. Variable neighborhood search: principles and applications. European Journal of Operational Research 2001;130:449–67.
[10] Hansen P, Mladenović N. Variable neighborhood search. In: Pardalos PM, Resende MGC, editors. Handbook of applied optimization. Oxford: Oxford University Press; 2002. p. 221–34.
[11] Hemmelmayr VC, Doerner KF, Hartl RF. A variable neighborhood search heuristic for periodic routing problems. European Journal of Operational Research 2009;195(3):791–802.
[12] Ke L, Archetti C, Feng Z. Ants can solve the team orienteering problem. Computers & Industrial Engineering 2008;54(3):648–65.
[13] Lin S. Computer solutions of the traveling salesman problem. Bell System Technical Journal 1965;44:2245–69.
[14] Mladenović N. A variable neighborhood algorithm: a new metaheuristic for combinatorial optimization. In: Abstract of papers presented at optimization days, Montreal, 1995. p. 112.
[15] Mladenović N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24(11):1097–100.
[16] Polacek M, Doerner KF, Hartl RF, Maniezzo V. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. Journal of Heuristics 2008;14(5):405–23.
[17] Polacek M, Doerner KF, Hartl RF, Kiechle G, Reimann M. Scheduling periodic customer visits for a travelling salesperson. European Journal of Operational Research 2007;179:823–37.
[18] Polacek M, Hartl RF, Doerner KF, Reimann M. A variable neighborhood search for the multi depot vehicle routing problem with time windows. Journal of Heuristics 2004;10:613–27.
[19] Righini G, Salani M. Dynamic programming for the orienteering problem with time windows. Technical Report 91, Dipartimento di Tecnologie dell'Informazione, Universita degli Studi Milano, Crema, Italy; 2006.
[20] Righini G, Salani M. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Computers & Operations Research 2009;36: 1191–203.
[21] Savelsbergh MWP. The vehicle routing problem with time windows: minimizing route duration. ORSA Journal on Computing 1992;4(2):146–54.
[22] Schilde M, Doerner KF, Hartl RF, Kiechle G. Metaheuristics for the bi-objective orienteering problem. Swarm Intelligence, to appear.
[23] Solomon MM. Algorithms for the vehicle routing and scheduling problem with time window constraints. Operations Research 1987;35:254–65.
[24] Taillard E, Badeau P, Gendreau M, Guertin F, Potvin JY. A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science 1997;31:170–86.
[25] Tang H, Miller-Hooks E. A tabu search heuristic for the team orienteering problem. Computers & Operations Research 2005;32(6):1379–407.
[26] Tsiligirides T. Heuristic methods applied to orienteering. The Journal of the Operational Research Society 1984;35(9):797–809.
[27] Vansteenwegen P, Souffriau W, Vanden Berghe G, Van Oudheusden D. A guided local search metaheuristic for the team orienteering problem. European Journal of Operational Research 2009;196:118–27.
[28] Vansteenwegen P. Planning in tourism and public transportation. PhD thesis, Katholieke Universiteit Leuven; 2008.