A unified matheuristic for solving multi-constrained traveling salesman problems with profits

Rahma Lahyani, Mahdi Khemakhem, Frédéric Semet

Noname manuscript No.
(will be inserted by the editor)

A unified matheuristic for solving multi-constrained traveling salesman problems with profits

Rahma Lahyani · Mahdi Khemakhem · Frédéric Semet

Received: date / Accepted: date

Abstract In this paper, we address a rich Traveling Salesman Problem with Profits encountered in several real-life cases. We propose a unified solution approach based on variable neighborhood search. Our approach combines several removal and insertion routing neighborhoods and efficient constraint checking procedures. The loading problem related to the use of a multi-compartment vehicle is addressed carefully. Two loading neighborhoods based on the solution of mathematical programs are proposed to intensify the search. They interact with the routing neighborhoods as it is commonly done in matheuristics. The performance of the proposed matheuristic is assessed on various instances proposed for the Orienteering Problem and the Orienteering Problem with Time Window including up to 288 customers. The computational results show that the proposed matheuristic is very competitive compared with the state-of-the-art methods. To better evaluate its performance, we generate a new testbed including instances with various attributes. Extensive computational experiments on the new testbed confirm the efficiency of the matheuristic. A sensitivity analysis highlights which components of the matheuristic contribute most to the solution quality.

Keywords · Profitable Tour Problem with Compartments · Matheuristic · Exact Loading Neighborhoods · Approximate Routing Neighborhoods · Orienteering Problem · Orienteering Problem with Time Window

1 Introduction

Routing problems where profits are associated with the visits of customers are extensively studied in the combinatorial optimization literature. Many papers and book chapters discuss these problems, since many industrial applications are modeled according to this general framework (e.g., Laporte and Martello (1990); Gendreau et al (1998a); Fischetti et al (2007); Balas (2007)). Some surveys appeared

R. Lahyani
LAGIS, UMR CNRS 8219
Ecole Centrale de Lille, France
E-mail: rahma.lahyani@ec-lille.fr

over the last decade review variants and applications of these problems (see e.g., Feillet et al (2005); Vansteenwegen et al (2011); Archetti et al (2013)).

Traveling Salesman Problems (TSP) with Profits are single-vehicle routing problems with two conflicting objectives. One consists in maximizing the total collected profit while the other aims to reduce the total route cost. Depending on the definition of the objective function, three classes of TSP with profits can be distinguished, Feillet et al (2005). When both criteria are combined linearly in the objective function, the problem is the so-called Profitable Tour Problem (PTP) introduced by Dell'Amico et al (1995). When the total travel cost is upper-bounded and the profit is maximized, the problem is referred to as the Orienteering Problem (OP) introduced by Tsiligirides (1984) or the selective traveling salesman problem. When the objective is to minimize the travel costs and the profit collected must exceed a preset lower bound, the problem is called the Prize Collecting TSP (PCTSP). The PCTSP is originally defined by Balas (1989) by penalizing the unvisited vertices in the objective function. In this paper, an extended variant of the PTP including multiple constraints is addressed. The OP and the Orienteering Problem with Time Windows (OPTW) are considered to assess the efficiency of the proposed approach.

As stated in Lahyani et al (2014), a Rich Vehicle Routing (RVRP) Problem is a problem reflecting the complexities of a real-life context by combining various challenges faced daily. In this paper, we are interested in solving a Rich variant of the PTP with a maximum route duration, referred to as (RPTP). The proposed RPTP enriches the basic PTP in many ways. It may be considered as a time constrained capacitated profitable tour problem with multiple products and incompatibility constraints. Dealing with several complicated constraints encountered in common practical situations makes this problem more challenging. Particularly, we examine a variant of the PTP arising when a multi-compartment vehicle is involved. The use of such vehicles is relevant in several practical situations dealing with incompatible products, (e.g., to perform selective waste collection (Muyldermans and Pang (2010)), to transport animals from farms to slaughterhouses, (Oppen et al (2010)), to distribute various petroleum products to petrol stations, (Brown and Graves 1981, Cornillier et al (2012)), to deliver dry, refrigerated and frozen food (Derigs et al (2011)), to collect olive oil (Lahyani et al (2015)). In the RPTP, the request of a customer is composed of demands for different products. A profit is associated with the demand for each product. A customer may be satisfied partially by delivering one or more products of its placed request. Since feasible tours are limited in time and capacity, the vehicle might not visit all the customers. The vehicle has different compartments with different capacities. A key feature is that some products are incompatible and must be kept separated during transportation. There are also incompatibility relations between some products and some compartments. Last, a time window and a service time are associated with each customer. Waiting times at the customer site are permitted but penalized in the objective function. The total cost of a tour is the total profit minus the travel cost and the cost for the total waiting time.

Formally, the RPTP considered can be defined as follows. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected complete graph where $\mathcal{V} = \{0, \ldots, n\}$ is the vertex set and $\mathcal{E} = \{(i, j) : i, j \in \mathcal{V}, i \neq j\}$ is the edge set. Feasible tours correspond to cycles including the depot, vertex 0. Vertices $i \in \mathcal{V}' = \{1, \ldots, n\}$ correspond to poten-

tial customers. Let d_{ij} and t_{ij} denote the non-negative travel cost and the travel time associated with edge $(i,j) \in \mathcal{E}$. We set $d_{ij} = \infty$ if the edge $(i,j) \notin \mathcal{E}$. With each customer $i \in \mathcal{V}'$, are associated a hard time window $[e_i, l_i]$, within which the deliveries of i take place, and a service time s_i. In the case of early arrival at customer i, a cost is incurred corresponding to the waiting time until e_i. Time windows are also associated with the depot, and they correspond to the opening hours. There is no service time at the depot. For the sake of simplicity, the service times are included in the travel time. Service times are then disregarded in the remainder of this paper. There is a set $p \in \mathcal{P} = \{1, \ldots, P\}$ of products. Each customer $i \in \mathcal{V}'$, can place several orders, each referring to one single product p. We denote by $o_i^p \in \mathcal{O}$ the order placed by customer i for product p. With each order o_i^p we associate a demand q_i^p. A positive integer profit g_i^p is associated with each $q_i^p \neq 0$. There is at least one order for each product type $p \in \mathcal{P}$ and the delivery of any product must not be split. We consider one vehicle which can visit a subset of customers within a given time limit T^{max}. The vehicle has a capacity Q with compartments $w \in \mathcal{W} = \{1, \ldots, W\}$. Each compartment w has a capacity Q_w and is equipped with a debit meter. The set $\mathcal{IP} \subseteq \mathcal{P} \times \mathcal{P}$ denotes the incompatibility relation between products. $(p,q) \in \mathcal{IP}$ means that products p and q must not be carried together in the same compartment. The set $\mathcal{IPC} \subseteq \mathcal{P} \times \mathcal{W}$ defines incompatibilities between products and compartments, forbidding product p to be transported in compartment w.

In this paper, we tackle a challenging problem since we address a routing problem with complicated loading restrictions. Only one study devoted to a routing problem with vehicles with compartments considered the loading component according to the general case (Pirkwieser (2012)). The authors propose first-fit, best-fit and best-fit decreasing heuristics as well as a constraint programming algorithm to solve the problem of assigning products to compartments considering potential incompatibilities. Previous works (e.g., Muyldermans and Pang (2010); El Fallahi et al (2008)) consider a simple case with two compartments and two products, each being dedicated to one compartment. There is no assignment problem, and the loading problem reduces to knapsack problems. Following Pirkwieser (2012), we denote the loading subproblem as the Compartment Assignment Problem (CAP).

Considering the literature devoted to vehicle routing problems, some attempts have been made recently to propose unified models and algorithms tackling different classes of vehicle routing problems (e.g., Pisinger and Ropke (2007); Subramanian (2012); Vidal et al (2014)). Some studies describe optimization algorithms for multi-constrained OP with m vehicles, referred to as Team Orienteering Problem, (e.g., Garcia et al (2010); Tricoire et al (2010); Souffriau et al (2013)). However, no previous work has been dedicated to rich variants of TSP with profits. There are few studies dealing with basic extended variants of TSP with profits, e.g., the capacitated PTP (CPTP) (e.g., Archetti et al (2009); Jepsen (2011)), or the OPTW (e.g., Righini and Salani (2006); Tricoire et al (2010); Labadie et al (2011)). In this paper, we present a unified solution approach for a large class of TSP with profits ranging from academic problems to multi-attribute problems with no need for customization. The main contributions of this paper are the followings. We define a multi-constrained TSP with profits which is a generalization of the PTP,

the CPTP, the PTP with time windows, the OP and the OPTW. We model the CAP with incompatible products. We develop a unified matheuristic combining routing and loading neighborhoods. The proposed solution approach, referred to as VNS*, addresses a large set of instances from the OP and the OPTW literature following a generic parameters tuning. Last, we design a new testbed for the multi-constrained PTP with compartments, which may be useful for future multi-compartments routing studies. This paper is organized as follows. Section 2 describes the matheuristic approach. Section 3 presents the detailed issues related to the constructive heuristic, the routing neighborhoods, the CAP solution and the route feasibility check procedures. Computational results and an extensive sensitivity analysis on a large class of problems are reported in Section 4. Section 5 concludes and discusses some future research guidelines.

2 Matheuristic approach

In Cordeau et al (2002), the authors claim that VRP heuristics can be analyzed according to four attributes: *accuracy*, *speed*, *flexibility* and *simplicity*. Designing a unified method for RVRPs represents a considerable research challenge which makes difficult to meet these four criteria simultaneously. The matheuristic proposed in this paper will rather focus on flexibility and simplicity. Most of the VRP heuristics proposed to solve one variant concentrate rather on accuracy and speed. To maintain a good compromise between flexibility and simplicity, a special attention must be paid to the main features of the problem and to the properties of the solution.

The multi-attribute PTP is an *NP*-hard combinatorial problem since the OP is *NP*-hard (Golden et al, 1987). Moreover, Gendreau et al (1998b) outline some reasons explaining the difficulty of designing high-quality heuristics for the OP. Part of the trouble lies in the fact that profits associated with customers and the distances between them are independent and lead to define conflicting objectives. While it is, usually, difficult to select the customers that are part of the solution, considering time-windows further complicates the solution process. Therefore, the design of a simple method providing high-quality solution is quite challenging.

In this paper, we propose a matheuristic based on Variable Neighborhood Search (VNS) which includes exact procedures for the examination of loading neighborhoods. We denote this method VNS*. Our choice to select the VNS metaheuristic (Mladenovic and Hansen, 1997) to solve the RPTP is motivated by its ability to take into account a wide variety of constraints. VNS proves to be very efficient for solving complex VRPs (Wen et al, 2011; Schilde et al, 2011). This success may be explained either by its ability: (i) to escape from local optima compared with population based metaheuristics; (ii) to exploit unexplored parts of the solution space by applying different problem specific neighborhoods. The basic steps of the proposed matheuristic are provided in Algorithm 1.

The proposed matheuristic includes two main phases. The first phase corresponds to the constructive stage (Steps 5-7 in Algorithm 1) and consists in building an initial solution s thanks to an adapted Nearest Neighbor Algorithm (NNA). NNA aims to insert each order in the current circuit while satisfying the temporal, capacity and compatibility constraints (see Section 3.1). The second stage (Steps 8-25) consists in applying VNS* to diversify and improve the initial solution. The

key idea is to exploit systematically the neighborhood structures during the search. Given an initial solution s, VNS* procedure alternates between the perturbation phase and the improvement phase (see Sections 2.1 and 2.2) for a preset number $iter^{max}$ of iterations.

Figure 1 sketches the main steps of VNS*, and illustrates the solution representation. Let $\mathcal{L}$ be the set of customers served by route s and $\mathcal{M}$ be the set of customers removed from s at a given step. The set $\mathcal{M}$ is initially empty. We denote by $\mathcal{U} \subseteq \mathcal{O}$ the set of the orders o_i^p placed by customers $i \in \mathcal{V}' \setminus \mathcal{L}$. When the insertion of an order in s is examined, s may not include all orders o_i^p associated with customer $i \in \mathcal{L}$. However a deletion move removes from the route all delivered orders o_i^p placed by customer i. Since the orders placed by a customer $i \in \mathcal{V}' \setminus \mathcal{L}$ are known and not delivered, we use interchangeably the expression *inserting customers* or *inserting orders* in pool $\mathcal{U}$ in the remainder of this paper.

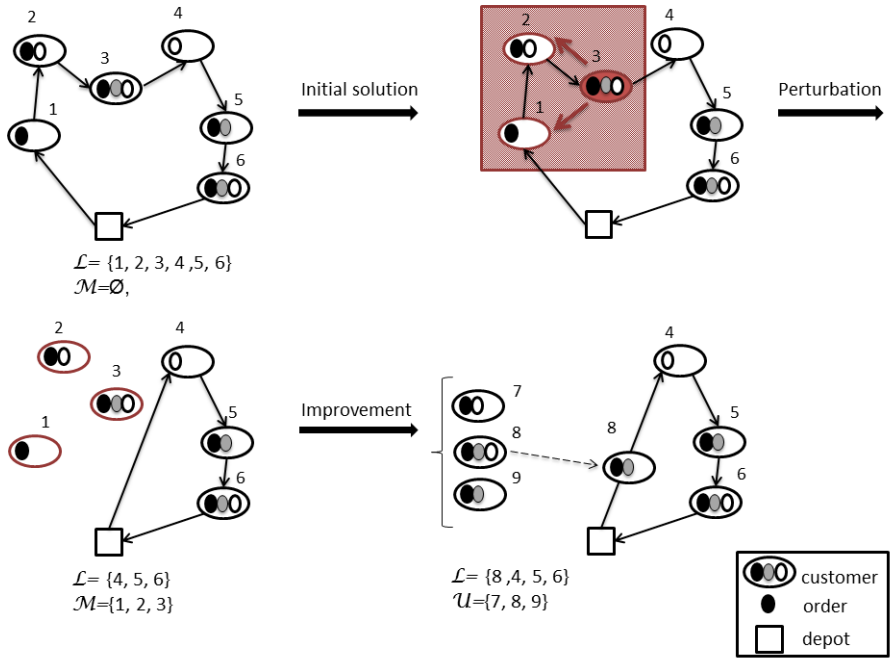

Fig. 1 VNS* main steps

2.1 Perturbation phase

The perturbation phase (Step 13), also known as the shaking phase, aims to diversify the search while maintaining the promising parts of the incumbent solution, without spending too much computational time. We propose to design the perturbation phase as an adapted Large Neighborhood Search (LNS) heuristic (Shaw, 1997, 1998). The perturbation phase of VNS* consists in partially destroying the

Algorithm 1: Variable Neighborhood Search* (VNS*)

input : (s)/* (obtained after applying the constructive NNA) */
output: (s^*)
Let s^+ be an intermediate solution;
$s^* \leftarrow s$;
$iter \leftarrow 1$;
while $iter \leq iter^{max}$ **do**
  if *(*$iter \neq 1$ *)* **then**
    $s \leftarrow$ new built solution;/* (obtained by the multi-start NNA) */
  end
  $s^+ \leftarrow s$;
  for *(*$i \leftarrow 1$ **to** 4*)*/* (Select a removal neighborhood H_i^-) */ **do**
    for *(*$j \leftarrow 1$ **to** 4*)*/* (Select an insertion neighborhood H_j^+) */ **do**
      for *(*$k \leftarrow 1$ **to** kd*)*/* (Fix the number of deleted customers) */ **do**
        $kc = 2 * kd$;/* (Fix the number of inserted customers) */
        Apply perturbation phase on s;
        Apply local search heuristic on s;
        Apply 2_*opt* local search heuristic on s;
        if *(* s *is better than* s^+*)* **then**
          goto Step 8 ;
        end
      end
    end
  end
  if *(* s^+ *is better than* s^**)* **then**
    $s^* \leftarrow s^+$;
  end
  Apply loading based improvement heuristic on s^*;
  $iter++$;
end
Waiting_Time_Optimization (s^*);
return s^*;

current solution without trying to reinsert removed customers again. Rather than using one large neighborhood as in standard LNS, we consider a set of four removal neighborhoods H_i^-, $i \in \{1, \ldots, 4\}$: the similarity removal neighborhood, the random removal neighborhood, the worst profit removal neighborhood and the spatio-temporal removal neighborhood (see Section 3.2.1 for more details). Each removal neighborhood eliminates up to kd customers from the current solution s according to a predefined criterion. The perturbation phase is followed by local search procedures aiming to improve the shaken solution gradually.

2.2 Improvement phase

The local search approach iteratively improves the solution obtained at the perturbation phase by moving from neighbors of the current solution to local optima. Two important issues must be addressed when local search heuristics are designed: (i) the quality of the solutions obtained (ii) the complexity of the local search heuristic. An efficient local search heuristic should reach a good balance. The Variable Neighborhood Descent (VND) (Hansen and Mladenovic, 2003) satisfies both criteria and has been applied successfully to many optimization problems. It enables

to reach global optima more probably than with a single neighborhood structure. Our adapted VND considers alternatively routing based neighborhoods and loading based neighborhoods. It includes three different local search procedures described in what follows.

Local search heuristic (Step 14) : The implemented local search heuristic, denoted for short by (LS), is a modified extension of the *Ruin and Recreate* heuristic introduced by Schrimph et al (2000). It consists in removing randomly one or more customers from the current solution and inserting undelivered orders of pool $\mathcal{U}$, unlike the Ruin and Recreate procedure which tries to re-insert the removed customers in better positions. The LS is a first improvement heuristic, i.e., the search procedure stops as soon as a solution improving the current objective value is found. We consider four sequential insertion neighborhoods, denoted by H_i^+, $i \in \{1, \ldots, 4\}$. Since the objective in the RPTP combines the total profit, the total travel costs and the total waiting time costs, the insertion criteria either ensure a good trade-off between two or three of these terms or focus on a single one. Given an insertion neighborhood, orders are considered sequentially to select the best feasible insertion. The selected order is then cost-effectively inserted in the solution. LS steps are parameterized according to the number of customers to remove, kd, the maximum number of customers to insert, kc, and the position in the route from which customer(s) may be removed.

2-opt Local Search (Step 15) : This heuristic attempts to reoptimize the solution obtained by the LS heuristic by decreasing the total travel time. First, it removes two arcs from a given route and reconnects the route by inserting two other arcs. When the time window constraints are considered, the orientation of the path may be reversed. The evaluation of the solution feasibility with respect to time window constraints is ensured by an effective time feasibility algorithm. If the new feasible solution is better than the current best solution in terms of total travel time, the procedure is reiterated. The algorithm stops in a local optima when no 2-opt exchange is possible. When an improved solution is identified, an attempt to insert new customers from $\mathcal{U}$, the pool of undelivered orders, according to the current insertion neighborhood H_i^+. Even if feasible 2-opt exchanges are seldom identified in the presence of time windows, experimental results demonstrate that it is worthy to keep this improvement technique.

Loading based improvement heuristic (Step 25) : Determining the loading of products for a multi-compartment vehicle while satisfying incompatibilities constraints between products and between products and compartments results in an *NP*-hard packing problem, referred to as the CAP. This vehicle loading problem is a key feature of the RPTP. We propose two exact loading neighborhoods based on the solution of mathematical programs: the Quadratic Multiple Knapsack Problem with Conflicts and the Linear Multiple Knapsack Problem with Conflicts. These neighborhoods, described in Section 3.2.2, aim to reoptimize the loading plan associated with the current routing solution. The first neighborhood explores the search space by swapping products between compartments. The second one determines the optimal loading of the vehicle given $\mathcal{L}$, the set of customers visited.

To speed up the solution evaluation, it is crucial to implement efficient feasibility check algorithms with respect to temporal and physical incompatibilities. Savelsbergh (1992) proposed fundamental algorithms for handling time windows in routing problems, later improved versions by Cornillier et al (2009). In this paper, we implemented two algorithms: *TW_Feasibility_Check* and *Waiting_Time_Optimization* based on the algorithms proposed by Cornillier et al (2009). The *TW_Feasibility_Check* is an exact algorithm checking the feasibility of the route with regards to time windows. It is called after each removal or insertion move. The purpose of the *Waiting_Time_Optimization* exact algorithm is to minimize the waiting time in a given solution. This algorithm is called at the end of the search, it postpones the departure time from the depot as much as possible. The capacity and incompatibility constraints are checked through a fast assignment heuristic that we propose to solve the CAP. It is an adapted version of the best-fit heuristic designed for the bin-packing problem. Invoked after each insertion move, it provides feasible assignment of orders while satisfying the loading constraints. Specifically, it first checks the feasibility of a potential order insertion with respect to capacity and incompatibility constraints. Then, it assigns the order to a compartment considering compartments according to their increasing residual capacity.

3 Main features of the matheuristic

3.1 Multi-start constructive heuristic

In 1994, Perttunen (1994) claimed that using initial solutions generated by constructive heuristics outperforms randomly generated initial solutions for TSPs. In this section, we propose a simple and fast NNA able to construct a feasible solution s by inserting orders of $\mathcal{U}$. Starting from an empty route, the NNA tries to insert all orders o_i^p of customer i, $p \in \mathcal{P} = \{1, \ldots, P\}$ in the last position of the route as long as the temporal, capacity and compatibility restrictions are satisfied. The insertion is repeated until no feasible insertion exists. When the insertion of order o_j^p associated with customer j (and profit g_j^p) after order o_i^p associated with customer i is considered, the insertion cost $\hat{c}_{ij}^p$ is computed as expressed in equation (3). g_{crt}, d_{crt}, t_{crt} denote respectively the current total profit, the current total distance and the current total time of the route. d_{ij} corresponds to the distance between customers i and j. $shift_{t_{ij}}^{NNA}$ and $shift_{d_{ij}}^{NNA}$ (see equations (1) and (2)) are the time and distance increases. $w_j = max\{0, e_j - a_j\}$ is the waiting time in j where a_j corresponds to the vehicle arrival time at customer j.

$$shift_{d_{ij}}^{NNA} = d_{ij} + d_{j0} - d_{i0} \tag{1}$$

$$shift_{t_{ij}}^{NNA} = t_{ij} + w_j + t_{j0} - t_{i0} \tag{2}$$

$$\widehat{c}_{ij}^p = \frac{(d_{crt} + shift_{d_{ij}}^{NNA} + t_{crt} + shift_{t_{ij}}^{NNA})}{(g_{crt} + g_j^p)} \tag{3}$$

The NNA repeatedly selects the order having the lowest coefficient $\widehat{c}_{ij}^p$. This criterion promotes orders with high profits and generating low time and distance

increases. Such criterion has been experimentally proved to be more efficient than other criteria focusing either on time, distance or profit. The complexity of this heuristic is $O(n)$ since only the last position of the current route is considered when an insertion move is evaluated.
Metaheuristics based on local optimization, usually, need some diversification to escape from local optimality. To achieve diversification, we implement a multi-start strategy in VNS* which consists in initiating the search $iter^{max}$ times from a new solution once a region of the solution space has been explored. The first solution is obtained applying the NNA as described above. For the remaining iterations, the profits associated with orders and the distance matrix between customers $i, j \in \mathcal{V}'$ are randomized. The randomization is controlled by a parameter σ which varies between $[1/2, 3/2]$.

3.2 Neighborhoods

The proposed matheuristic may be seen as an iterative sequence of *Ruin and Recreate* steps since it destroys a part of the solution in the shaking phase and repairs it by means of the local search procedures. The search may be guided either for diversification, i.e., examining new regions of the solution space, or for intensification, i.e., focusing on promising regions. The neighborhoods developed for the RPTP solution are of two types:

- Routing neighborhoods: They modify the sequence and the customers visited in a given solution. They include four insertion neighborhoods and four removal neighborhoods. Some neighborhoods are adapted from the VRP literature. They may require some parameters denoted by Greek letters which will be set in Section 4.
- Loading neighborhoods: We propose two loading neighborhoods with different objectives. In both cases, the best solution in the neighborhood is obtained through the optimal solution of a mathematical program.

The diversity of the proposed neighborhoods allows a good exploration of the solution space as discussed in Section 4.

3.2.1 Routing neighborhoods

Removal neighborhoods : The first three removal neighborhoods are adapted from the literature while the last one is new. A pseudo-code outlining the generic removal method is presented in Algorithm 2. It takes as an input an initial feasible solution s and returns a partial solution s^*. The removal neighborhoods removes up to kd customers according to a predefined neighborhood. Let $\mathcal{L}_i$ be the i^{th} customer of $\mathcal{L}$. It is noteworthy that removal neighborhoods eliminate all orders associated with a given customer from a route. The procedure is controlled by the parameter ψ and the randomization parameters ϕ.

Similarity removal neighborhood : This removal neighborhood was proposed by (Shaw, 1997, 1998) and implemented by (Ropke and Pisinger, 2006; Pisinger and Ropke, 2007; Ribeiro and Laporte, 2012; Demir et al, 2012). The aim of the similarity removal neighborhood is to remove a set of customers that are similar with

Algorithm 2: Generic removal procedure

input : (s, kd, $\mathcal{L}$, $\psi \geq 1$)
output: (s^*)
1 Initialize $\mathcal{M}$, $\mathcal{M} \leftarrow \emptyset$;
2 Eliminate randomly a customer i from s;
3 $\mathcal{M} = \{i\}$;
4 **while** ($|\mathcal{M}| \leq kd$) **do**
5 Select randomly a customer i from $\mathcal{M}$;
6 Update $\mathcal{L}$;
7 Compute the removal ratio;
8 Sort all the customers $j \in \mathcal{L}$ according to the removal ratio;
9 Generate a random number $\phi \in [0, 1]$;
10 $j \leftarrow \lfloor \phi^{\psi} |\mathcal{L}| \rfloor$;
11 Eliminate randomly $\mathcal{L}_j$ from s;
12 $\mathcal{M} = \mathcal{M} \bigcup \{\mathcal{L}_j\}$;
13 **end**
14 Remove from s customers in $\mathcal{M}$, let s^* be the resulting solution;
15 **return** s^*;

respect to a predefined similarity criterion. Removing similar customers promotes the insertion of more customers which may lead to better solutions. We define the criterion $Sim(i, j)$ between two customers i and j as follows:

$$Sim(i, j) = \phi_1 d_{ij} + \phi_2 |b_i - b_j| + \phi_3 |shift_i^g - shift_j^g| \quad (4)$$

This criterion includes three terms. The first term represents the distance d_{ij} between customers i and j. Temporal similarity is expressed through the difference between the departure times from i and j. The third term measures the difference of attractiveness between i and j. $shift_i^g$ denotes the contribution of customer i to s in terms of profit : $shift_i^g = \sum_{p \in \mathcal{P}} g_i^p, \forall o_i^p \notin \mathcal{U}$. ϕ_1, ϕ_2 and ϕ_3 are normalized weights.
The heuristic initially selects a customer i randomly and removes it from the solution s. For the subsequent $kd - 1$ iterations, the heuristic selects customers similar to the removed customer according to $Sim(i, j)$. The similarity between customers increases as the value of the criterion decreases. More precisely, at Step 8 of Algorithm 2 the customers in $\mathcal{L}$ are sorted in increasing order according to $Sim(i, j)$. Then, some randomness is introduced in the selection thanks to the parameter ϕ. For a given value of ϕ, a low value of ψ ($\psi = 1$) corresponds to complete determinism and leads to remove customer j with a low value of $Sim(i, j)$ while the probability of choosing a customer j less similar to customer i increases as the value of ψ increases. The time complexity of the neighborhood exploration in the worst cases is $O(n^2)$. Figure 2 summarizes the main steps of this heuristic when $kd = 4$. The similarity removal procedures selects randomly from the solution s customer 3 and puts it in the pool $\mathcal{M}$. Then it computes the $Sim(i, j)$ value between customer 3 and all the remaining customers in s. Customers $2, 4$ and 5 are considered as similar to customer 3 and are removed from s.

Random removal neighborhood : This neighborhood consists in removing kd customers randomly. It may be seen as a special case of the similarity removal neighborhood with $\phi 1 = \phi 2 = \phi 3 = 0$. The exploration of this neighborhood is implemented in $O(1)$.

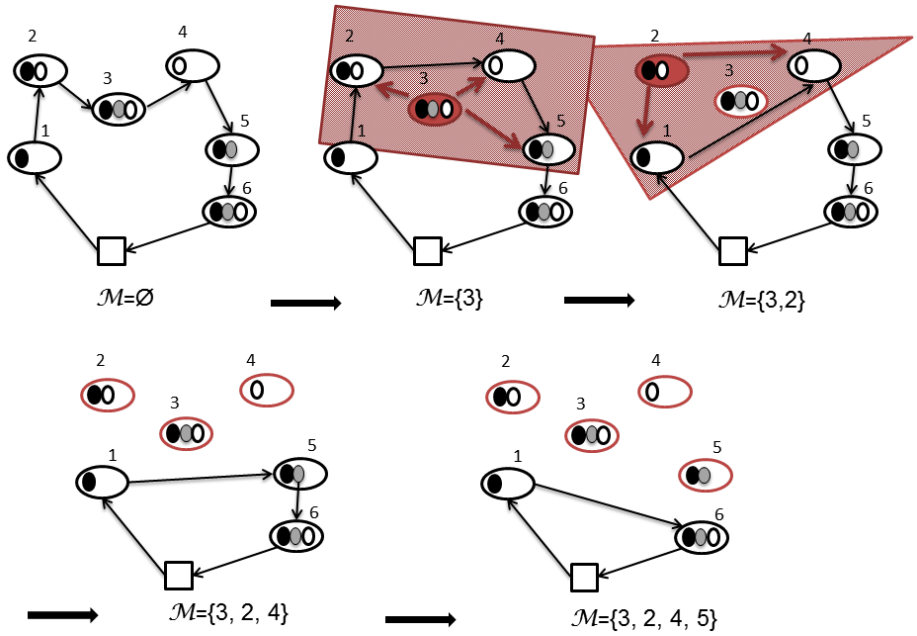

Fig. 2 Removal procedure based on the similarity removal neighborhood

Spatio-temporal removal neighbordhood : This neighborhood is another adaptation of the removal neighborhood proposed by (Shaw, 1997, 1998). It aims to remove customers similar in terms of distance and time. It differs from the similarity removal neighborhood by its removal criterion. The spatio-temporal ratio was proposed by Prescott-Gagnon et al (2009) and is defined by equation (5).

$$ST_{ij} = \frac{1}{(\frac{d_{ij}}{d_i^{max}} + \frac{1}{T_{ij}^{ST}+T_{ji}^{ST}})} \tag{5}$$

We set $d_i^{max} = max_{j \in s}\{d_{ij}\}$, $i \in s$. If customer i is visited immediately before customer j, T_{ij}^{ST} measures the proximity of time windows between i and j and is equal to $max\{1, min\{l_j, l_i + t_{ij}\} - max\{a_j, a_i + t_{ij}\}\}$ (Gendreau et al, 1995). This neighborhood is explored according to the procedure described in Algorithm 2. The customers served by the current route are sorted in decreasing order with respect to ST_{ij} value. The larger ST_{ij} is, the closer are customers i and j. The heuristic removes customer $j^* = argmax_{j \in s}\{ST_{ij}\}$. The exploration of the spatio-temporal removal neighborhood is implemented in $O(n^2)$ in the worst case.

Worst profit removal neighborhood : The worst profit removal neighborhood aims to remove the less profitable customers. The idea is to select customers that do not contribute enough to the total solution profit. First, customers $i \in \mathcal{L}$ are sorted in increasing order of the total profit of delivered orders $shift_i^g$. Then, kd customers are removed according to this order with some randomization as explained in Algorithm 2. Note that the procedure described in Algorithm 2 is slightly modified since Step 2 is useless. This heuristic is implemented in $O(n)$.

Insertion neigborhoods : Four insertion neighborhoods are used to improve the current solution. The first two neighborhoods are adapted neighborhoods proposed in the OPTW literature, whereas the last two are new ones. As a general rule, the neighborhood exploration consists in determining the best insertion among orders $o_i^p \in \mathcal{U}$ according to a predefined insertion ratio. Thus, kc orders at most are inserted in the solution. The orders placed by a given customer are served in sequence, i.e. during a single visit to the customer. Such sequences are enforced by defining null distances and times between these orders. In addition, the insertion of orders of partially served customers would be favored by the second loading neighborhood. In each insertion neighborhood, the feasibility of an insertion move is checked before it is implemented. Since the checking feasibility algorithms have different time complexities, they are invoked according to the increasing order of their complexities. The worst case time complexity of the exploration of the four neighborhoods is in $O(n^2)$. The generic pseudo-code is provided in Algorithm 3.

Algorithm 3: Generic insertion procedure

```
   input  : (s, kc)
   output: (s*)
 1 while (kc ≥0) do
 2     for (i ← 1 to |U|) do
 3         if (the compatibility and capacity constraints are satisfied) then
 4             for (j ← 1 to |L|) do
 5                 if ( e_i + t_ij <= l_j and t_crt + t_ij ≤ T^max ) then
 6                     Compute the insertion ratio;
 7                     Memorize and update the best position best_pos;
 8                     i* ← i;
 9                 end
10             end
11         end
12     end
13     Insert i* at best_pos;/* (Best insertion)                    */
14     if (the temporal feasibility of s* is maintained) then
15         Update s*;
16     else
17         remove i* from s*;
18     end
19     kc − −;
20 end
21 return s*;
```

Profit-time insertion neighborhood : This insertion neighborhood is adapted from a neighborhood described in the constructive heuristic proposed by Labadie et al (2011) for the single-product OPTW. It consists in inserting new orders while the feasibility of capacity, temporal and incompatibilities constraints are satisfied. Giving a current route s, the heuristic examines all feasible insertions of order o_i^p in s and, the position leading to the best compromise between profit and time increase is selected. More precisely, the insertion of order o_i^p in route s between orders o_j^p and o_{j+1}^p results in a new route $\overline{s}$. The route time increase can be computed as $shift_i^t = t_{ji} + w_i^s + t_{i,j+1} + w_{j+1}^{\overline{s}} - t_{j,j+1} - w_{j+1}^s$ where $w_{j+1}^{\overline{s}}$ corresponds to the

waiting time incurred at customer $j+1$ in route $\overline{s}$. Then, the profit-time insertion criterion for order o_i^p is the ratio between the asssociated profit and the route time increase : $\frac{g_i^p}{shift_i^t}$. We select the undelivered order with the best ratio. More complex ratios, including terms related to the number of customers which can be reached from a given customer, were considered but preliminary experiments did not demonstrate any improvement in the solution quality.

Availability insertion neighborhood : Vansteenwegen et al (2009b) proposed a neighborhood for solving the Team OPTW (TOPTW) based on the following insertion criterion: $\frac{(g_i^p)^2}{shift_i^t}$. Preliminary results showed that this criterion is of little interest for the RPTP since the solution quality depends on other attributes. We extend this insertion criterion by considering the time, travel cost and capacity attributes. Let $shift_i^d$ denotes the travel cost increase if order o_i^p is inserted. Ratios are computed to measure the increase for each attribute on the current route with respect to the available quantity. The insertion criterion is computed according to the expression (6). The best insertion corresponds to the order with the highest value.

$$\frac{(g_i^p)^2}{\frac{shift_i^t}{availableTime} + \frac{q_i^p}{availableCapacity} + \frac{shift_i^d}{availableDistance}} \tag{6}$$

Cost-profit insertion neighborhood : This neighborhood select orders with the best compromises between the cost and the time increases and the profit. For $o_i^p \in \mathcal{U}$, the order associated with the lowest value of the following ratio is selected : $\frac{shift_i^t + shift_i^d}{g_i^p}$.

Best profit insertion neighborhood : This neighborhood focuses on a central attribute in the RPTP objective function which is the profit. It selects orders with the highest profits while satisfying the route feasibility and attempts to insert them in the current solution at the first feasible positions.

3.2.2 Loading neighborhoods

Quadratic Multiple Knapsack Problem with Conflicts : This neighborhood aims to rearrange the loading of the vehicle to maximize the empty space while maintaining the route sequence unchanged, (see Figure 3). We model this problem as a Quadratic Multiple Knapsack Problem with incompatibility constraints between products and products and compartments. This problem is *NP*-hard, (Golumbic., 2004). Let $\mathcal{O}^s \subseteq \mathcal{O}$ be the set of orders delivered by route s. To simplify the notation, we refer to order o_i^p served by route s as order o. With each order o, is associated a customer $i \in \mathcal{L}$, a quantity q_o and a product $p_o \in \mathcal{P}$. Binary variables x_{ow} indicate whether or not order o is loaded in compartment w. The problem considered is:

$$\max \sum_{w \in \mathcal{W}} \left(Q^w - \sum_{o \in \mathcal{O}^s} x_{ow} q_o\right)^2 \tag{7}$$

subject to

$$\sum_{o \in \mathcal{O}^s} x_{ow} q_o \leq Q^w \quad w \in \mathcal{W} \tag{8}$$

$$\sum_{w \in \mathcal{W}} x_{ow} = 1 \quad o \in \mathcal{O}^s \tag{9}$$

$$x_{ow} + x_{kw} \leq 1 \quad o \in \mathcal{O}^s, k \in \mathcal{O}^s, w \in \mathcal{W}, p \in \mathcal{P}, q \in \mathcal{P}, (p_o, q_k) \in \mathcal{IP} \tag{10}$$

$$x_{ow} = 0 \quad o \in \mathcal{O}^s, w \in \mathcal{W}, (p_o, w) \in \mathcal{IPC} \tag{11}$$

$$x_{ow} \in \{0, 1\} \quad o \in \mathcal{O}^s, w \in \mathcal{W}. \tag{12}$$

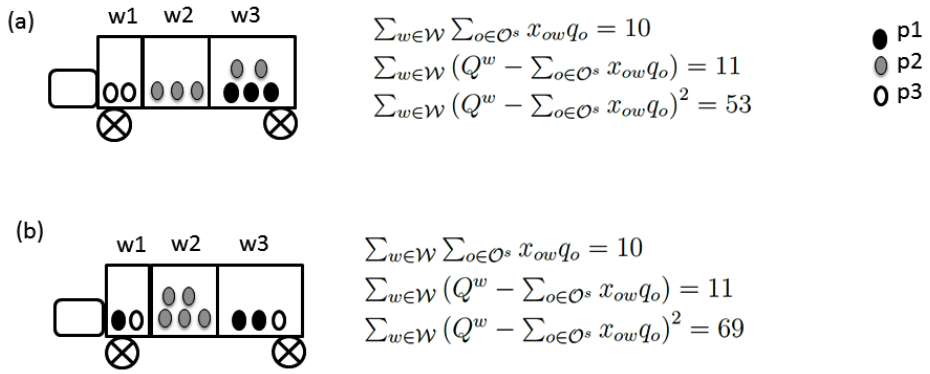

Fig. 3 Example of solution of the Quadratic Multiple Knapsack Problem with Conflicts. Suppose $\mathcal{P}$=$\{p1, p2, p3\}$, $\mathcal{O}^s$=$\{1, \ldots, 10\}$, q_o=1 $\forall o \in \mathcal{O}^s$, $\mathcal{W}$=$\{w1, w2, w3\}$, Q^{w1}=3, Q^{w2}=7, Q^{w3}=11, $\mathcal{IP}$=$\{(p2, p3)\}$ and $\mathcal{IPC}$=$\{(1, w2, p1)\}$. Figure (a) corresponds to the loading of the current solution s. Figure (b) corresponds to the solution of the associated Quadratic Knapsack Problem with Conflicts.

Defining a suitable objective function in any neighborhood search is a critical factor. In this case, the maximization of the total residual compartment capacity is useless since the objective value is equal to the total residual capacity of the initial loading (see the example described in Figure 3). In this neighborhood, the objective function focus rather on loading used compartments optimally to keep unused the remaining compartments. To do so, the proposed objective function (7) consists in maximizing the sum of the squares of the residual compartment capacities. Constraints (8) ensure that compartment capacities are respected. Constraints (9) impose that each order $o \in \mathcal{O}^s$ is assigned to exactly one compartment $w \in \mathcal{W}$. Constraints (10)–(11) express the incompatibility conditions between products and between products and compartments. This model is solved thanks to the commercial solver IBM CPLEX 12.5.

Multiple Knapsack Problem with Conflicts : This second loading neighborhood aims to determine the optimal loading of the vehicle according to the total profit

given $\mathcal{L}$, the set of visited customers. This problem is modeled as a Multiple Knapsack Problem with incompatibility constraints between products and products and compartments. This problem is also known as the disjunctively constrained Knapsack Problem (Pferschy. and Schauer., 2009) and is *NP*-hard. Let $\bar{\mathcal{O}}$ be the set of orders $\bar{o}_i^p$ placed by customers $i \in \mathcal{L}$. For sake of simplicity, we refer to $\bar{o}_i^p$ by $\bar{o}$ with which is associated a product $p_{\bar{o}} \in \mathcal{P}$, a customer $i \in \mathcal{L}$, a quantity $q_{\bar{o}}$ and a profit $g_{\bar{o}}$. The binary variable $y_{\bar{o}w}$ is equal to one if order $\bar{o}$ is loaded in compartment w. The loading neighborhood is explored by solving the following integer linear program:

$$\max \sum_{w \in \mathcal{W}} \sum_{\bar{o} \in \bar{\mathcal{O}}} y_{\bar{o}w} g_{\bar{o}} \tag{13}$$

subject to

$$\sum_{\bar{o} \in \bar{\mathcal{O}}} q_{\bar{o}} y_{\bar{o}w} \leq Q^w \quad w \in \mathcal{W} \tag{14}$$

$$\sum_{w \in \mathcal{W}} y_{\bar{o}w} \leq 1 \quad \bar{o} \in \bar{\mathcal{O}} \tag{15}$$

$$y_{\bar{o}w} + y_{kw} \leq 1 \quad \bar{o} \in \bar{\mathcal{O}}, k \in \bar{\mathcal{O}}, w \in \mathcal{W}, p \in \mathcal{P}, q \in \mathcal{P}, (p_{\bar{o}}, q_k) \in \mathcal{IP} \tag{16}$$

$$y_{\bar{o}w} = 0 \quad \bar{o} \in \bar{\mathcal{O}}, w \in \mathcal{W}, (p_{\bar{o}}, w) \in \mathcal{IPC} \tag{17}$$

$$y_{\bar{o}w} \in \{0, 1\} \quad \bar{o} \in \bar{\mathcal{O}}, w \in \mathcal{W}. \tag{18}$$

The objective (13) maximizes the total collected profit whereas constraints (14)-(18) can be interpreted as constraints (8)-(12). This model is solved using the commercial integer programming solver IBM CPLEX 12.5 and the performance on small and large instances is discussed in Section 4.3.

4 Computational experiments

To assess the efficiency of the proposed matheuristic, we report experimental results on problems related to the RPTP. This section is divided into three main subsections. In Section 4.1, we report results obtained by applying the proposed matheuristic on OP instances. In Section 4.2, computational experiments are conducted on the OPTW, a more difficult RPTP. Since the loading neighborhoods are not relevant for the OP and OPTW instances, we generate new instances under three real-life scenarios to evaluate VNS* better. Extensive computational experiments and a sensitivity analysis for the RPTP demonstrate the contribution of the main components of the matheuristic to the solution quality.

The matheuristic was coded in *C* and ran on an Intel Quad Core with 2.66 GHz and 4 GB Ram. Results are summarized in Tables 1-5 and 7-9. In these tables, results are provided by instance class and average statistics over the 5 runs are reported for each instance class. Columns headed *Class* identify the instance class, columns headed *Gap%* report the percentage gap to the best known value solution and columns headed *Time(s)* give the computational time in seconds for each instance class.

Preliminary experiments were carried out on sample instances to determine the best parameter setting with respect to speed and efficiency. $iter^{max}$, the maximum number of VNS* iterations, is fixed to 20. The maximum number of deleted

customers in the shaking phase is $kd = 2$. The local search procedures try to insert new customers while $kc \leq 2 * kd$. For the routing neighborhoods, we use the parameter setting determined in (Ropke and Pisinger, 2006), $(\phi 1, \phi 2, \phi 3)$=$(9, 3, 2)$ and (Prescott-Gagnon et al, 2009), ψ=35. Preliminary experiments also revealed that the performance of VNS* is improved when $iter^{max}$ or kd are increased. Since this improvement is at the expense of increased computational times, we do not consider such settings. We keep the same parameter values for the whole testbed to demonstrate that the performance of the proposed matheuristic is not subject to any customization, as in Vidal et al (2014).

4.1 OP instances

Tsiligirides (1984) propose 3 sets of instances (1_p21, 1_p32 and 1_p33) for the OP, which include 18, 11 and 20 instances respectively. The number of customers ranges from 21 to 33. A second testbed with larger instances was generated by Chao et al (1996). It includes 2 sets of instances (1_p64 and 1_p66) with 14 and 26 instances including 64 and 66 customers respectively. We compare the values obtained by VNS* with the optimum values published by Tsiligirides (1984) and the best known (BK) provided by Chao et al (1996). Moreover, we compare the efficiency of VNS* with those of 5 methods:

- A five steps heuristic (CGW) by Chao et al (1996)
- An Ant Colony Optimization algorithm with 20 iterations (ACO^t_{20}) by Souffriau et al (2008) (best of 3 runs),
- A deterministic Guided Local Search (GLS) by Vansteenwegen et al (2009a),
- A Pareto Ant Colony Optimization (P-ACO) and a Pareto Variable Neighborhood Search (P-VNS) by Schilde et al (2009)(best of 10 runs),
- a GRASP based algorithm with path relinking (GRASP) proposed by Campos et al (2013) (best of 10 runs).

Tables 1 and 2 summarize the computational results on both instances sets. Columns headed *Gap%* in Table 1 and Table 2 give the average percentage gaps to the optimal or BK values. Note that for the three instances sets generated by Tsiligirides (1984), both methods described by Schilde et al (2009) find the optimal values but computational times are not reported. Table 1 shows that VNS* outperforms the previously proposed methods by providing the optimal values in less than one second, 0.67 seconds on average. GLS appears to be the fastest method on average on the three classes of instances (0.44 seconds) but provides worst results with an average deviation equal to 1.74% . In Table 2, *VNS** finds the best deviation gap for the set 1_p64 in 4.64 seconds. The *CGW* identifies solutions with the same deviation gap but requires longer computation times. On the second set of instances, *VNS** achieves only slightly worse results with an average deviation gap equal to 0.08% compared with the best deviation gap (0.06%) obtained by *P_ACO*.

According to Dongarra (2013), computers used for VNS* and four methods considered have similar performance (similar enough to compare absolute run times). The exception is the machine used to run the *GLS* heuristic, which is two times slower. Note that Schilde et al (2009) and Campos et al (2013) report only running times elapsed to identify the BK values for Chao et al (1996) instances.

Class	***Gap%***			***Time(s)***		
	GLS	ACO^t_{20}	*VNS**	*GLS*	ACO^t_{20}	*VNS**
1_p21	1.34	4.78	0.00	0.25	-	0.33
1_p32	3.28	1.80	0.00	0.52	-	0.70
1_p33	0.60	1.79	0.00	0.55	-	0.98

Table 1: Comparison of VNS* with state-of-the-art methods on instances proposed by Tsiligirides (1984)

Class	***Gap%***						***Time(s)***					
	CGW	*GLS*	*P_ACO*	*P_VNS*	*GRASP*	*VNS**	*CGW*	*GLS*	*P_ACO*	*P_VNS*	*GRASP*	*VNS**
1_p64	0.07	1.09	0.13	0.17	0.14	0.07	177.04	2.18	1.97	2.56	0.08	4.64
1_p66	0.43	1.31	0.00	0.13	0.00	0.10	158.65	2.26	0.22	1.18	0.07	4.94

Table 2: Comparison of VNS* with state-of-the-art methods on instances proposed by Chao et al (1996)

4.2 OPTW instances

The OPTW is a simplified version of the RPTP where the vehicle has only one compartment w and the demand for a unique product p is known for each customer i. The OPTW has received significant attention in the literature, and a large set of instances was proposed. The instances were obtained from the data sets generated by Solomon (1987) for the VRPTW and from the data sets of Cordeau et al (1997) for the periodic MDVRPTW. Based on the Solomon instances, Righini and Salani (2006) generated 58 instances for the OPTW by considering 50 and 100 customers (c-50, r-50, rc-50, c-100, r-100, rc-100). They derived a second set of 10 instances based on Cordeau data sets (pr01-pr10). Righini and Salani (2006, 2009) solved the derived sets of the OPTW to optimality thanks to a dynamic programming approach. Montemanni and Gambardella (2009) proposed an Ant Colony System (ACS) and added 2 data sets by considering 27 Solomon instances with 100 customers (c2-100, r2-100, rc2-100) and 10 additional Cordeau instances (pr11-pr20). The Solomon instances (c2-100, r2-100, rc2-100) as well as the Cordeau instances are characterized by wide time windows and are known to be hard to solve to optimality (Righini and Salani, 2006; Labadie et al, 2011). Most approximate algorithms failed to obtain optimal solutions in a reasonable time on these instances. In total, 105 instances for the OPTW divided into 11 classes are available. The number of customers for Cordeau data sets ranges from 48 to 288 while the Solomon instances contain 50 or 100 customers. The demand of each customer in each instance represents the associated profit. The maximum route duration T^{max} is equal to the closing time of the starting point. These test instances can be downloaded from http://www.mech.kuleuven.be/en/cib/op.

The performance of the proposed matheuristic is compared with those of the ACS algorithm developed by Montemanni and Gambardella (2009), and those of the 5 following heuristics. Vansteenwegen et al (2009b) proposed a fast Iterative Local Search (ILS) based on a multi-start strategy to solve the OPTW and the

TOPTW instances. Tricoire et al (2010) designed a VNS to deal with instances of the multi-period orienteering problem with multiple time windows, the OPTW and the TOPTW. A GRASP hybridized with an evolutionary local search algorithm was proposed in Labadie et al (2011). Recently, Labadie et al (2012) developed an effective Granular VNS (GVNS). To compare the performance of the GVNS to the ILS fairly, the authors report the detailed results of a fast version of the GVNS which terminates once the solution value of the ILS is retrieved. Lin and Yu (2012) propose a fast and a slow version of an algorithm based on Simulated Annealing ((FSA) and (SSA)). The SSA outperforms the FSA due to a stopping criterion based on the number of iterations without improvement of the best encountered solution. The SSA improves the BK solutions for 4 instances: rc2-104, pr11, pr17 and pr18. In the remainder of this paper, we compute average percentage gaps for ACS, ILS, VNS, GRASP and GVNS taking into consideration these new BK values.

In this paper, the Euclidean distances for all the instances are rounded down to the second decimal as in (Righini and Salani, 2009; Montemanni and Gambardella, 2009; Labadie et al, 2011, 2012). It is worth mentioning that for the remaining methods distances are rounded down to the second digit for Cordeau instances and to the first digit for Solomon instances.

For a fair comparison, we compare the computers on which computational results were conducted (Dongarra, 2013). The performance of our computer is equivalent to the computers used by Vansteenwegen et al (2009b) and Lin and Yu (2012). For the other methods, our processor is approximately two times faster. It should be noted that ILS and SSA are deterministic algorithms and were run only once. ACS, GRASP and GVNS were executed 5 times while the results reported for VNS were obtained with 10 runs. The comparison with previous methods is provided in Tables 3-5.

Since a large variance of computational time can be observed on Solomon instances, we divide methods into slow algorithms: ACS and VNS, (see Table 3) and fast algorithms: ILS, GRASP, GVNS and SSA (see Table 4). In Table 3, we report results obtained with VNS* when we limit the CPU time to 120 and 300 seconds. In Table 4, the stopping criteria for VNS* is either $iter^{max}$ fixed to 20 or the CPU time limited to 20 seconds. Results for Cordeau instances are summarized in Table 5. Then, the computational results for VNS* are those obtained with a CPU time limit equal to 120 or 300 seconds.

As a rule, the efficiency of VNS* increases with running time. After 120 seconds of computation time, VNS* provides better average solution values than the ACS on 8 out of 9 classes. After 300 seconds, VNS* and VNS provide similar results for the different instances classes. On average, Tricoire et al (2010) report a deviation gap to BK values equal to 0.33% in 318 seconds while VNS* obtain 0.38% in 300 seconds.

When the stopping criteria is a preset number of iterations, VNS* leads to near-optimal solutions with a little computational effort. The average deviation gap to the BK solutions ranges from 0% to 2.77% over all instance classes with an average computation time equal to 7.86 seconds. VNS* outperforms GVNS in terms of solution quality and provides average deviation gap similar to GRASP (0.88% vs. 0.73% respectively). For the instances classes with 50 and 100 customers, the SSA provides slightly lower average deviation gaps than VNS* at the expense of larger computational times. Computation times indicate that ILS is certainly the fastest

Class	***Gap%***				***Time(s)***
	ACS (3600s)	*VNS*	*VNS* (120s)*	*VNS* (300s)*	*VNS*
c-50	0.00	0.00	0.00	0.00	53.63
r-50	0.00	0.00	0.33	0.33	24.18
rc-50	0.00	0.19	0.00	0.00	31.93
c-100	0.00	0.11	0.00	0.00	98.39
r-100	0.24	0.05	0.06	0.05	89.10
rc-100	0.00	0.04	0.00	0.00	65.21
c2-100	0.58	0.21	0.42	0.33	560.17
r2-100	3.16	1.05	2.10	1.53	1065.82
rc2-100	2.04	1.35	1.61	1.23	869.41

Table 3: Average results of the slow methods on Solomon instances

method, but with the worst average gap (1.93%). This is not surprising since ILS is a deterministic algorithm designed to reach good quality solution very quickly.

Considering the first seven instance classes in Table 4, VNS* achieves better results when the stopping criterion is the CPU time limit (20 seconds) than the number of iterations. For the instances classes (r2-100 and rc2-100) known to be very challenging, VNS* is able to provide good quality solution with an average gap equal to 3.05% at most. This leads us to conclude that VNS* is competitive with respect to the state-of-the-art methods on the Solomon instances.

On the Cordeau data sets, VNS* is not as efficient as on the Solomon instances. After 120 seconds and 300 seconds of computation time, the average gaps to BK solutions on (pr01-pr10) and (pr11-pr20) are 4% and 3.06% respectively. VNS* outperforms ILS (6.91%) and ACS (6.02%) with average computational times of 1.86 seconds and 3600 seconds respectively. SSA provides the best solution values on average and finds new BK solutions for three of the Cordeau instances: pr11, pr17 and pr18. For the state-of-the-art methods, the average gap from BK solutions ranges from 0.97% to 10.84%. Due to their characteristics (large TW, large time duration per route), Cordeau instances require much more diversification than intensification. Therefore, to assess the efficiency of the proposed matheuristic on more difficult instances, we propose a slightly modified version of the VNS*. This method, VNS*C, explores the solution space more effectively. It consists in increasing the number of deleted customers kd to $\frac{\#Cust}{3}$ in each iteration and by reducing $iter^{max}$ to 5. The average deviation gap decreases significantly. After 300 seconds of running time, VNS*C is as efficient as SSA: 0.99% and 2.96% versus 0.97% and 3.25% in 112.21 seconds and 162.40 seconds respectively. In addition, it is worth mentioning that VNS* improves the BK solution for instance pr11 while VNS*C improves the BK solution for instance pr13.

Altogether, the performance of the unified matheuristic is very promising. VNS* is able to compete with the current state-of-the-art methods. It provides optimal or BK solutions for slightly longer running times. Note that some procedures are invoked even though they are useless. Indeed, the code was not modified to solve OP and OPTW instances. This represents the cost to pay when the method implemented is able to solve a wider range of multi-constrained problems.

Class	Gap%						Time(s)				
	ILS	GRASP	GVNS	SSA	VNS* (20 iter)	VNS* (20s)	ILS	GRASP	GVNS	SSA	VNS* (20 iter)
c-50	0.33	0.00	-	-	0.00	0.00	0.27	7.01	-	-	0.88
r-50	0.63	0.47	-	-	0.33	0.33	0.20	0.93	-	-	0.80
rc-50	2.21	1.11	-	-	0.10	0.00	0.18	0.91	-	-	0.65
c-100	1.11	0.00	1.22	0.00	0.49	0.00	0.33	22.59	166.46	21.07	1.88
r-100	1.90	0.22	2.68	0.11	0.25	0.09	0.19	3.51	29.43	23.34	2.75
rc-100	2.92	0.40	3.51	0,00	0.19	0.00	0.23	1.99	9.80	22.19	2.03
c2-100	2.28	0.61	1.11	0.13	1.16	0.85	1.71	32.18	192.40	37.49	11.28
r2-100	2.89	1.61	3.37	1.29	2.77	3.05	1.66	11.18	33.82	45.83	29.46
rc2-100	3.43	2.20	3.96	0.96	2.65	2.60	1.63	8.21	16.01	50.25	21.05

Table 4: Average results of the fast methods on Solomon instances

Class	Gap% Time(s)									
	ILS	ACS (3600s)	VNS	GRASP	GVNS	SSA	VNS*C (120s)	VNS*C (300s)	VNS* (120s)	VNS* (300s)
pr01-10	4.72	1.20	1.08	1.44	1.61	0.97	1.20	0.99	2.41	1.77
	1.75		*822.07*	*5.03*	*12.37*	*112.21*				
pr11-20	9.11	10.84	2.92	2.92	3.81	3.25	3.45	2.96	5.60	4.36
	1.98		*1045.93*	*7.90*	*24.22*	*162.40*				

Table 5: Average results of the state-of-the-art methods on Cordeau instances

4.3 Computational results for the RPTP

4.3.1 A new testbed

Since no data sets are available for the RPTP addressed in this paper, we generate a new testbed to evaluate VNS*. The proposed testbed is based on the Solomon data sets with 50 and 100 customers and on the extended instances including 200 customers proposed for the VRPTW by Gehring and Homberger (1999). We generate 172 original instances classified according to 18 classes. We introduce three types of products $p \in \mathcal{P} = \{1, 2, 3\}$ with unit profits respectively equal to $10, 15, 20$. We split the original customer demand into three demands randomly. A customer i may not place an order for a given product p, i.e., $q_i^p = 0$. The profit g_i^p associated with each order o_i^p is obtained by multiplying the quantity q_i^p by the associated unit product profit.

Three compartments $w \in \mathcal{W} = \{1, 2, 3\}$ are considered for each vehicle. The compartments capacities (Q^1, Q^2, Q^3) are obtained by dividing the original capacity Q into three parts as follows: $Q^1 = 0.2 * Q, Q^2 = 0.3 * Q, Q^3 = 0.5 * Q$. $\mathcal{W}$ and $\mathcal{P}$ are kept identical for all instances. As in the original data sets, instances belonging to the same class have the same customers locations but have different time windows and quantity (eventually, null) of products for each customer. Last, we have to define incompatibility relations. To address different loading problems as they arise in real-life scenarios, we propose to generate three types of incompatibilities as shown in Table 6. For each type of incompatibilities, we have 172 instances divided into 18 classes.

For the first type of instances, *type A*, no incompatibility constraints are imposed, i.e., each product may be loaded in any compartment with any other product. Therefore, a solution for an instance of *type A* is feasible for the corresponding OPTW instance. *Type B* instances correspond to the distribution of liquid products to customers or animal feeds to farms or to the waste collection. In such cases, there are no product-compartment incompatibilities and all the products are in-

compatible pairwise. The third instances type, *type C*, corresponds to the general case. We choose to maintain a moderate level of incompatibility restrictions. Two products must be kept segregated during transportation and each product may be loaded in two compartments out of three.

Sets	***Type A***	***Type B***	***Type C***
$\mathcal{IP} \subseteq \mathcal{P} \times \mathcal{P}$	$\mathcal{IP}$= {∅}	$\mathcal{IP}$={(1,2),(1,3),(2,3)}	$\mathcal{IP}$={(1,3)}
$\mathcal{IPC} \subseteq \mathcal{K} \times \mathcal{W} \times \mathcal{P}$	$\mathcal{IPC}$= {∅}	$\mathcal{IPC}$={∅}	$\mathcal{IPC}$={(1,1,1),(1,2,2), (1,3,3)}

Table 6: Incompatibility scenarios in generated data sets

The instance names are as follows: the first two letters give the type of customer distribution followed by an asterisk to differentiate the instances designed for the VRPTW with compartments from the original VRPTW ones. The first digit gives the type of time windows (1: narrow, 2: large), the second three-digits indicate the number of customers.

In Table 7 the performance of VNS* on the three data sets is reported. The first two columns are the average number of customers, and the average number of orders included in the solution respectively. Columns headed *Obj* report the average objective value obtained with the standard parameter setting. These results show that solutions for instances of *type A* provide higher average objective than solutions for instances of *type B* and *type C*, although the average number of customers served remains almost the same. This could be expected since instances of class *A* are less constrained. Even if the number of visited customers is limited by the temporal and capacity constraints, the absence of incompatibility relations enables to assign products freely to compartments and to increase the objective value.

For instances of *type B*, identifying very good solutions requires larger computational times, almost equal to the double of average running time required for the solution of *type A* and *type C* instances. This is likely because the assignment of each product to any compartment induces some symmetry which cannot be broken easily. Finally, VNS* seems sensitive not only to incompatibility constraints but also to the type of the time windows associated with customers, since a few customers are inserted for instances with narrow time windows. These observations seem to be consistent with previous research works (Righini and Salani, 2006; Labadie et al, 2011).

To assess the efficiency of the loading neighborhoods, we consider two new versions of VNS*:

- VNS*= The Linear Multiple Knapsack Problem with Conflicts is solved at the end of each VNS iteration.
- VNS*I= The Quadratic Multiple Knapsack Problem with Conflicts is solved at the end of each VNS iteration. Next an insertion neighborhood attempting to insert new orders is applied. Last, we solve the Linear Multiple Knapsack Problem with Conflicts.
- VNS*II= VNS* without any loading neighborhoods.

To avoid misleading results, the remaining tests are performed on instances of *type C*. In Table 8, we report the results for each instance class.

Clearly, VNS*II provides good but not convincing results. The solutions serve almost the same number of customers as the solutions obtained when the loading neighborhoods are invoked. However, average objective values obtained by VNS*II are 10% lower than those provided by VNS* and VNS*I. As expected, optimizing the loading plan in MC-VRP has a significant impact on the solution quality.

Extensive computational experiments with VNS*I put into highlight that improving the quality solution is unlikely within a reasonable amount of computation time. When solving the Quadratic Multiple Knapsack Problem with Conflicts, we stop CPLEX when an integer feasible solution has been proved to be within 0.05% of optimality or when the CPU time limit set to 25 seconds is reached. Given the results presented in Table 8, VNS*I is two to thirty times slower than VNS* without a significant improvement on the solution quality. VNS* provides slightly better results than those provided by VNS*I compared to the results obtained without loading neighborhoods. However, we did not try to increase the CPU time limit. Indeed, deriving better results at the expense of longer run times was not our goal. To conclude, VNS* is the best configuration providing high quality solutions without a large computational effort. However, the efficiency of the VNS* varies according to the type of the time windows and to the instances size.

4.3.2 Sensitivity analysis

The proposed matheuristic embeds different components that contribute to the performance of VNS*. In order to better analyze the contribution of the main ones, we conduct some additional experiments reported in this section. In these experiments, the performance of each setting is assessed by reporting the average deviation gap from the best solution value found over the three classes of instances (c*1-100, r*1-100 and rc*1-100). The results provided by VNS* are used as reference solutions. The parameter setting is unchanged.

First, we study the impact of the multi-start constructive heuristic by running VNS* with the same initial solution for 20 iterations, i.e., lines (5-7) are removed from Algorithm 1. We obtain an average percentage gap of 1.04%, 6.33% and 4.42% for classes: c*1-100, r*1-100 and rc*1-100 respectively. These results confirm that starting from a new solution at each iteration plays an important role in the effective exploration of the solution space.

One critical component in VNS* is the *Get_Compartment* heuristic. To prove its efficiency, we solve the Quadratic Multiple Knapsack Problem with Conflicts for the final solution $s*$ obtained by VNS*. We denote the new solution $\bar{s}$. We compare the residual capacity of solutions $s*$ and $\bar{s}$. No improvement has been obtained on the sample instances. Therefore, the loading feasibility check heuristic provides a very good (most likely the optimal) assignment of products to compartments. It represents a key component in the design of VNS*. Furthermore, we test the VNS* without the $Waiting\_time\_optimization$ heuristic. The average results obtained are slightly worse than those obtained when the $Waiting\_time\_optimization$ heuristic is applied.

Class	***Type A***				***Type B***				***Type C***			
	# Cust	*# Ord*	*Obj*	*Time(s)*	*# Cust*	*# Ord*	*Obj*	*Time(s)*	*# Cust*	*# Ord*	*Obj*	*Time(s)*
c*1-50	11.33	19.89	3478.47	2.10	11.11	18.44	3360.23	5.37	11.56	21.56	3236.88	2.43
c*2-50	33.38	82.88	9900.73	13.03	33.38	77.13	8734.90	83.08	33.00	81.38	9264.00	17.52
r*1-50	8.58	20.67	2824.96	1.32	8.58	20.17	2623.35	2.62	8.75	21.67	2785.72	1.41
r*2-50	38.91	92.18	8326.97	12.67	38.64	91.64	8207.63	40.49	38.00	90.36	8063.30	15.00
rc*1-50	8.63	18.63	3128.93	1.79	9.25	18.13	2871.65	2.95	9.00	19.63	2975.40	1.51
rc*2-50	35.75	89.88	10752.40	11.27	35.50	87.75	10326.02	94.09	34.75	88.00	10216.72	14.94
c*1-100	11.11	16.56	3553.52	3.97	11.11	16.00	3478.68	6.05	11.22	20.89	3279.04	3.68
c*2-100	31.63	66.63	11447.27	28.25	31.63	71.50	10118.81	107.14	32.00	72.25	10381.67	31.76
r*1-100	11.42	23.58	3318.47	4.02	10.83	19.92	3177.76	7.41	11.67	24.08	3160.05	4.17
r*2-100	49.82	122.55	13371.22	61.29	49.45	114.73	11981.13	316.85	49.64	121.55	12311.07	71.99
rc*1-100	10.38	20.13	3343.15	4.79	10.13	17.75	3176.81	6.02	10.50	21.13	3207.39	3.46
rc*2-100	42.63	102.50	13199.75	46.43	43.00	94.75	11715.23	222.65	42.63	103.88	11962.38	52.16
c*1-200	11.60	18.40	3555.19	7.18	11.20	17.00	3449.21	10.18	11.80	21.00	3273.55	7.06
c*2-200	33.50	66.40	11799.60	64.79	33.60	73.70	10423.75	165.82	33.70	70.10	10347.67	68.44
r*1-200	20.40	23.10	3621.71	16.71	16.60	21.90	3291.21	20.60	16.00	25.60	3209.68	14.37
r*2-200	87.10	128.90	16938.42	371.27	84.80	134.00	14568.86	267.57	80.40	135.00	16034.64	397.53
rc*1-200	20.50	22.90	3642.42	14.85	18.60	24.60	3316.47	30.64	17.40	26.30	3260.44	14.04
rc*2-200	81.10	128.40	16815.84	272.69	80.90	135.00	13516.57	232.84	79.00	134.00	16012.48	332.15
Average	30.43	59.12	7945.50	52.13	29.91	58.56	7129.90	90.13	29.50	61.02	7387.89	58.53

Table 7: Average results of VNS* on RPTP data sets

Class	***VNS****				***VNS*I***				***VNS*II***			
	# Cust	*# Ord*	*Obj*	*Time(s)*	*# Cust*	*# Ord*	*Obj*	*Time(s)*	*# Cust*	*# Ord*	*Obj*	*Time(s)*
c*1-50	11.56	21.56	3236.88	2.43	11.33	23.89	3210.17	3.91	11.56	22.44	3170.84	1.55
c*2-50	33.00	81.38	9264.00	17.52	33.00	81.13	9186.04	522.33	33.00	73.63	8369.57	12.59
r*1-50	8.75	21.67	2785.72	1.41	8.75	21.75	2759.27	4.26	8.75	21.42	2762.88	0.98
r*2-50	38.00	90.36	8063.30	15.00	38.18	90.55	8066.45	458.08	37.91	84.55	7679.49	12.33
rc*1-50	9.00	19.63	2975.40	1.51	8.75	20.75	2965.76	3.29	9.00	20.38	2948.56	1.00
rc*2-50	34.75	88.00	10216.72	14.94	34.75	87.50	10194.07	481.74	34.88	80.88	9466.77	11.37
c*1-100	11.22	20.89	3279.04	3.68	11.44	21.56	3266.81	5.31	11.11	20.33	3217.79	2.97
c*2-100	32.00	72.25	10381.67	31.76	32.13	73.00	10185.00	521.78	32.38	68.63	9497.83	27.78
r*1-100	11.67	24.08	3160.05	4.17	11.75	25.00	3126.62	12.69	11.58	24.33	3101.71	3.41
r*2-100	49.64	121.55	12311.07	71.99	49.64	120.91	12286.95	580.93	49.27	107.18	11018.73	61.56
rc*1-100	10.50	21.13	3207.39	3.46	10.63	21.75	3173.95	5.76	10.50	20.88	3150.69	2.78
rc*2-100	42.63	103.88	11962.38	52.16	43.00	103.88	11932.72	558.44	42.75	94.38	10983.27	45.84
c*1-200	11.80	21.00	3273.55	7.06	11.90	21.90	3247.12	8.53	11.80	22.80	3205.34	6.35
c*2-200	33.70	70.10	10347.67	68.44	33.70	68.90	10242.72	534.96	34.60	68.80	9641.03	63.50
r*1-200	16.00	25.60	3209.68	14.37	19.00	29.70	3166.47	83.40	20.50	27.40	3036.41	13.44
r*2-200	80.40	135.00	16034.64	397.53	83.40	139.50	15923.66	907.89	86.30	126.10	13147.38	347.32
rc*1-200	17.40	26.30	3260.44	14.04	17.70	29.00	3201.31	111.79	19.10	26.50	3133.65	12.96
rc*2-200	79.00	134.00	16012.48	332.15	82.30	140.00	15912.79	846.41	84.40	129.90	13070.97	283.31
Average	29.50	61.02	7387.89	58.53	30.07	62.26	7335.99	313.97	30.52	57.81	6700.16	50.61

Table 8: Average results of VNS* variants on RPTP data sets

Table 9 summarizes the behavior of each removal and insertion neighborhood, invoked in the current version of VNS*, when used in a LNS scheme combining one insertion neighborhood and one removal neighborhood. For these tests, we compare the quality of the solution obtained by VNS* without the $Waiting\_time\_optimization$ heuristic. The first four settings describe the impact of the removal neighborhoods when the insertion neighborhood is fixed. The results show that there is no significant difference between the four removal neighborhood performance. The aim of a removal neighborhood is to diversify the solution, which seems to be ensured by the four neighborhood in a similar way. However, the combination of the four removal neighborhoods leads to improve the exploration of the solution space.

For the next experiments (settings 5, 6 and 7), we arbitrarily select the similarity removal neighborhood to perturb the solution. In these experiments, all insertion neighborhoods perform well except insertion neighborhood (1). The insertion neighborhood (3) performs better with an average gap equal to 1.32% followed by the insertion neighborhoods (4) and (2). The insertion neighborhood (1) gives the worst average gap 12.20%. Among the 8 settings described in Table 9, setting 6 turns out to be the most efficient combination on the testbed. Nevertheless, applying setting 6 combining the insertion neighborhood (3) and the similarity removal neighborhood on the OP and OPTW instances fails to provide good quality solutions while the VNS* performs particularly well. These observations support the idea that combining several removal and insertion neighborhoods may have a positive impact on the solution quality and that designing the matheuristic with different neighborhoods is definitely a good option.

Conf.	*Removal heuristics*				*Insertion heuristics*				*Gap%*
	Similarity	Worst	Random	Spatio-temporal	Insert (1)	Insert (2)	Insert (3)	Insert (4)	c*1-100 r*1-100 rc*1-100
1	•					•			3.84 3.78 3.70
2		•				•			3.19 4.81 3.65
3			•			•			3.66 4.29 3.95
4				•		•			3.78 5.12 3.61
5	•				•				5.33 12.85 18.41
6	•						•		0.75 1.88 1.32
7	•							•	3.05 3.08 3.57
VNS*	•	•	•	•	•	•	•	•	0.00

Table 9: Effectiveness of the insertion and removal heuristics

5 Conclusions

In this paper, we introduce a rich variant of the PTP including a large class of temporal and physical constraints. Each customer may place one or more orders which may be not satisfied entirely. We developed a unified matheuristic based on routing and loading neighborhoods denoted VNS*. To diversify and intensify

the search, we suggested removal and insertion neighborhoods as well as different local search procedures. We tried to focus on the loading aspect of the problem which was barely considered in the MC-VRP literature. We introduced a heuristic to assign products to compartments, and we proposed two loading neighborhoods based on the solution of mathematical programs. We incorporated these neighborhoods in the matheuristic approach to optimize the loading plan for the current solution.

Extensive experimental results show that VNS* competes with state-of-the-art methods proposed for the OP and the OPTW without any customization. To better evaluate the matheuristic performance on rich instances, we generated three data sets. Each data set includes 172 instances and describes three real-life scenarios. As expected, VNS* performs especially well on the less constrained instances. In the presence of incompatibility restrictions, VNS* produces high-quality solutions on average with a larger computational effort. The sensitivity analysis reveals that the exact loading neighborhoods contribute to the VNS* performance significantly. They lead to improve the solution quality with no significant time increase. Furthermore, starting from a different solution at each iteration and combining removal and insertion neighborhoods contribute to the solution quality.

As Feillet et al (2005) pointed out, the PTP may appear as a sub-problem in column generation algorithms for a variety of vehicle routing problems that cannot be solved efficiently with a branch-and-bound/branch-and-cut algorithms. Future works could rely on the Dantzig-Wolfe decomposition of such problems to obtain a set-partitioning model of the problem and to solve RPTPs heuristically within a branch-and-price framework. Another future work could be to test our approach on other variants of PTP considering real-world applications.

Acknowledgements This work was partially supported by the International Campus on Safety and Intermodality in Transportation, the Nord-Pas-de-Calais Region, the European Community, the Regional Delegation for Research and Technology, the French Ministry of Higher Education and Research and the National Center for Scientific Research. This support is gratefully acknowledged. The authors also thank Fabien Tricoire and Matteo Salani for their answers to our requests regarding the computational experiments.

References

Archetti C, Feillet D, Hertz A, Speranza M (2009) The capacitated team orienteering and profitable tour problems. Journal of the Operational Research Society 60(2):831–842

Archetti C, Speranza M, Vigo D (2013) Vehicle routing problems. Tech. rep., Department of Economics and Management University of Brescia, Italy

Balas E (1989) The prize collecting traveling salesman problem. Networks 19(6):621–636

Balas E (2007) The prize collecting traveling salesman problem and its applications. In: Gutin G, Punnen AP (eds) The traveling salesman problem and its variations, Kluwer Academic Publisher, pp 663–695

Campos V, Marti R, Sánchez-Oro J, Duarte A (2013) Grasp with path relinking for the orienteering problem. Journal of Operational Research Society 65(10)

Chao I, Golden B, Wasil E (1996) A fast and effective heuristic for the orienteering problem. European Journal of Operational Research 88:475–489

Cordeau F J, Gendreau M, Laporte G (1997) A tabu search heuristic for periodic and multidepot problems. Networks 30:105–119

Cordeau J, Gendreau M, Laporte G, Potvin J, Semet F (2002) A guide to vehicle routing heuristics. Journal of the Operational Research Society 53(5):512–522

Cornillier F, Boctor F, Laporte G, Renaud J (2009) The petrol station replenishment problem with time windows. Computers & Operations Research 36(3):919–935

Cornillier F, Boctor F, Renaud J (2012) Heuristics for the multi-depot petrol station replenishment problem with time windows. European Journal of Operational Research 220(2):361–369

Dell'Amico M, Maffioli F, Varbrand P (1995) On prize-collecting tours and the asymmetric travelling salesman problem. International Transactions in Operational Research 2(3):297–308

Demir E, Bektas T, Laporte G (2012) An adaptive large neighborhood search heuristic for the pollution-routing problem. European Journal of Operational Research 223(2):346–359

Derigs U, Gottlieb J, Kalkoff J, Piesche M, Rothlauf F, Vogel U (2011) Vehicle routing with compartments: applications, modelling and heuristics. OR Spectrum 33(4):885–914

Dongarra J (2013) Performance of various computers using standard linear equations software. Tech. rep., Electrical Engineering and Computer Science Department University of Tennessee and Computer Science and Mathematics Division University of Manchester

El Fallahi A, Prins C, Calvo R (2008) A memetic algorithm and a tabu search for the multi-compartment vehicle routing problem. Computers & Operations Research 35:1725–1741

Feillet D, Dejax P, Gendreau M (2005) Traveling salesman problems with profits. Transportation Science 39(2):188–205

Fischetti M, Salazar-Gonzalez J, Toth P (2007) The generalized traveling salesman and orienteering problems. In: Gutin G, Punnen AP (eds) The traveling salesman problem and its variations, Kluwer Academic Publisher, pp 609–662

Garcia A, Vansteenwegen P, Souffriau W, Arbelaitz O, Linaza M (2010) Solving multi constrained team orienteering problems to generate tourist routes. Tech. rep., Centre for Industrial Management / Traffic & Infrastructure

Gehring H, Homberger J (1999) A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In: Proceedings of the Evolutionary Algorithms in Engineering and Computer Science, University of Jyvskyl, Jyvskyl, Finland, pp 57–64

Gendreau M, Hertz A, Laporte G, Stan M (1995) A generalized insertion heuristic for the traveling salesman problem with time windows. Operations Research 46(3):330–335

Gendreau M, Laporte G, Semet F (1998a) A branch and cut algorithm for the undirected selective travelling salesman problem. Networks 32(4):263–273

Gendreau M, Laporte G, Semet F (1998b) A tabu search heuristic for the undirected selective travelling salesman problem. European Journal of Operational Research 106(2-3):539–545

Golden B, Levy L, Vohra R (1987) The orienteering problem. Naval Research Logistics 34(3):307–318

Golumbic M (2004) Algorithmic graph theory and perfect graphs. Annals of Discrete Mathematics, Rutgers University, Piscataway, NJ, USA

Hansen P, Mladenovic N (2003) Variable neighborhood search. In: Glover F, Kochenberger GA (eds) Handbook of Metaheuristics, Kluwer Academic Publisher, pp 145–184

Jepsen M (2011) Branch-and-cut and branch-and-cut-and-price algorithms for solving vehicle routing problems. Dissertation, Technical University of Denmark, URL `http://orbit.dtu.dk/fedora/objects/orbit:89357/datastreams/file_6317942/content`

Labadie N, Melechovsk J, Calvo R (2011) Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. Journal of Heuristics 17(6):729–753

Labadie N, Mansini R, Melechovsk J, Calvo R (2012) The team orienteering problem with time windows: An lp-based granular variable neighborhood search. European Journal of Operational Research 220(1):15–27

Lahyani R, Khemakhem M, Semet F (2014) Rich vehicle routing problems: From a taxonomy to a definition. European Journal of Operational Research , forthcoming

Lahyani R, Coelho LC, Khemakhem M, Laporte G, Semet F (2015) A multi-compartment vehicle routing problem arising in the collection of olive oil in tunisia. OMEGA 51:1–10

Laporte G, Martello S (1990) The selective travelling salesman problem. Discrete Applied Mathematics 26(2-3):193–207

Lin S, Yu V (2012) A simulated annealing heuristic for the team orienteering problem with time windows. European Journal of Operational Research 217(1):94–107

Mladenovic N, Hansen P (1997) Variable neighborhood search. Computers & Operations Research 24(11):1097–1100

Montemanni R, Gambardella L (2009) An ant colony system for team orienteering problem with time windows. Foundations of Computing and Decision Sciences 34(4):287–306

Muyldermans L, Pang G (2010) On the benefits of co-collection: experiments with a multi-compartment vehicle routing algorithm. European Journal of Operational Research 206(1):93–103

Oppen J, Loekketangen A, Desrosiers J (2010) Solving a rich vehicle routing and inventory problem using column generation. Computers & Operations Research 37(7):1308–1317

Perttunen J (1994) On the significance of the intial solution in travelling salesman heuristics. The Journal of Operational Research Society 45(10):1131–1140

Pferschy U, Schauer J (2009) The knapsack problem with conflict graphs. Journal of Graph Algorithms and Applications 13(2):233–249

Pirkwieser DS (2012) Hybrid metaheuristics and matheuristics for problems in bioinformatics and transportation. PhD thesis, Vienna University of Technology

Pisinger D, Ropke S (2007) A general heuristic for vehicle routing problems. Computers & Operations Research 34(8):2403–2435

Prescott-Gagnon, Desaulniers G, Rousseau L (2009) A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. Networks 54(4):190–204

Ribeiro G, Laporte G (2012) An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. Computers & Operations Research 39(3):728–735

Righini G, Salani M (2006) Dynamic programming for the orienteering problem with time windows. Tech. rep., Dipartimento di Tecnologie dell'Informazione, Universita degli Studi Milano,Italy

Righini G, Salani M (2009) Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. Computers & Operations Research 36(4):1191–1203

Ropke S, Pisinger D (2006) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science 40(4):455–472

Savelsbergh M (1992) The vehicle routing problem with time windows: Minimizing route duration. ORSA Journal on Computing 4(2):146–154

Schilde M, Doerner K, Richard F, Kiechle G (2009) Metaheuristics for the bi-objective orienteering problem. Swarm Intelligenc 3(3):179–201

Schilde M, Doerner K, Richard F, Kiechle G (2011) Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. Computers & Operations Research 38(12):1719–1730

Schrimph G, Schneider J, Stamm-Wilbrandt H, Dueck G (2000) Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics 159(2):139–171

Shaw P (1997) A new local search algorithm providing high quality solutions to vehicle routing problems. Tech. rep., Department of Computer Science, University of Strathclyde, Scotland

Shaw P (1998) Using constraint programming and local search methods to solve vehicle routing problem. In: Proceedings CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming)

Solomon M (1987) Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35(2):254–265

Souffriau W, Vansteenwegen P, Berghe G, Oudheusden D (2008) Automated parameterisation of a metaheuristic for the orienteering problem. Adaptive and Multilevel Metaheuristics 136:255–269

Souffriau W, Vansteenwegen P, Berghe G, Oudheusden D (2013) The multiconstraint team orienteering problem with multiple time windows. Transportation Science 47(1):53–63

Subramanian A (2012) Heuristic, exact and hybrid approaches for vehicle routing problems. Dissertation, Universidade Federal Fluminense, Niteroi, Brazil, URL `http://www2.ic.uff.br/PosGraduacao/Teses/532.pdf`

Tricoire F, Romauch M, Doerner K, Hartl R (2010) Heuristics for the multi-period orienteering problem with multiple time windows. Computers & Operations Research 37(2):351–367

Tsiligirides T (1984) Heuristic methods applied to orienteering. Journal of the Operational Research Society 35(9):797–809

Vansteenwegen P, Souffriau W, Berghe G, Oudheusden D (2009a) A guided local search metaheuristic for the team orienteering problem. European Journal of Operational Research 196(1):118–127

Vansteenwegen P, Souffriau W, Berghe G, Oudheusden D (2009b) Iterated local search for the team orienteering problem with time windows. Computers & Operations Research 36(12):3281–3290

Vansteenwegen P, Souffriau W, Oudheusden D (2011) The orienteering problem: a survey. European Journal of Operational Research 209(1):1–10

Vidal T, Crainic TG, Gendreau M, Prins C (2014) A unified solution framework for multi-attribute vehicle routing problems. European Journal of Operational Research 234(3):658–673

Wen M, Krapper E, Larsen J, Stidsen T (2011) A multi-level variable neighborhood heuristic for a practical vehicle routing and driver scheduling problem. Networks 58(4):311–322