International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 8 (2016) pp. 355–363
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Multi-Trip Vehicle Routing Problem with Time Windows and Heterogeneous Fleet

François Despaux[1], **Sebastián Basterrech**[2]

[1]Instituto de Matématica y Estadística,
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
fdespaux@fing.edu.uy

[2]Department of Computer Science,
Faculty of Electrical Engineering and Computer Science
VŠB-Technical University of Ostrava
Ostrava-Poruba, Czech Republic
Sebastian.Basterrech.Tiscordio@vsb.cz

Abstract: **Distribution logistics comprises all activities related to the provision of finished products and merchandise to customers. Defining strategies for optimising the distribution procedure becomes a crucial issue for logistics enterprises. Normally, finding optimal strategies for a given logistic distribution problem is not straightforward since the complexity and time execution exponentially increases as the number of components of the problem increases. An efficient way that overcomes the complexity of finding optimal strategies within a reasonable time concerns the use of approximation algorithms and metaheuristics aiming to find good solutions in terms of quality and execution time. This article introduces a metaheuristic approach to solve a variation of a logistic problem named *Vehicle Routing Problem (VRP)*. Concretely, we present a solution for the *Multi-Trip VRP with Time Windows and Heterogeneous Fleet*. We add constraints to the original VRP concerning the time and the customer supply. Time constraints concerns the time windows on each customer and time horizon within which customers must be satisfied. In respect of the customer supply, we consider a heterogeneous fleet where vehicles are allowed to do multiple trips. We propose a solution for the problem using a Local Search and the Simulated Annealing technique. A set of benchmark scenarios widely used for the VRP is used in order to evaluate the performance of our approach.**

Keywords: Combinatorial Optimization, Vehicle Routing Problem, Metaheuristics, Simulating Annealing, Operational Research

I. Introduction

The need for finding optimal solutions on logistic and distribution problems have motivated the study of a wide range of *Vehicle Routing Problems (VRP)*. The main problem regarding logistic and distribution problems is the fact that complexity in terms of the execution time for finding an optimal solution exponentially increases as the number of components (vehicules, customers, routes, etc.) increases. Then, exact algorithms are not always suitable when the size of the problem grows considerably. Instead, practical methodologies such as metaheuristics, not aimed to guarantee an optimal solution but sufficient for immediate goals, are used. Metaheuristic approaches are then used when finding an optimal solution is impossible or impractical, speeding up the process of finding satisfactory solutions. In this paper, we make use of such metaheuristic approach for an extension of a well-known *Vehicle Routing Problem (VRP)*.
The goal of the VRP is to design a set of minimum-cost vehicle routes delivering goods to a set of customers where vehicle's routes start and finish at certain central depot. Over the last years, several approaches for solving the VRP and its variations have been presented in the literature [1–4]. A more recent review for two variations of the VRP problem is presented in [4] survey where publications regarding the vehicle routing problem with time windows (VRPTW) and the capacity vehicle routing problem (CVRP) are presented. In this article, we revisit the classic VRP problem and we introduce a new variation on it by adding constraints over the routing and on the distribution time in the system. We identify the problem as *Multi-Trip Vehicle Routing Problem with Time Windows and Heterogeneous Fleet (MTVRP-TWHF)*. In this problem, each customer has associated a *time windows*, that specifies a period for receiving deliveries. The total routing and scheduling cost include not only the total travel distance and time cost, but also the cost of waiting time when vehicles arrive out of the customer's time window. The problem also considers an *heterogeneous fleet* of vehicles, in which both capacity and cost differs from one vehicle to another one. The total time needed to complete these routes must not exceed a defined *time horizon* constraint.
There are basically two approaches for solving this kind of problems: exact and metaheuristics approaches, respectively. Exact algorithms have the advantage of finding a global solution for a given problem. However, and due to the nature of this kind of problems, the scalability of this approach is not straightforward and it is the main disadvantage since ve-

hicule routing problems belong to the NP-Hard class. Works on this direction for solving two variations of the vehicle routing problem are presented in [5] and [6]. Metaheuristic approaches [1] deal with the scalability problem by providing an approximate and sufficiently good solution even if this solution is not an optimal one.
In this article, we present a metaheuristic procedure for solving the MTVRP-TWHF problem based on two popular optimization techniques: *Local Search (LS)* and the *Simulated Annealing (SA)* techniques [7]. This work is a revised and a expanded version of the conference article [8].
The structure of this article is organized as follows. In Section II, we formalise the problem. Section III specifies our methodology for finding an initial solution based on a local search approach. Next, our algorithm based on Simulated Annealing is introduced. Empirical results and discussions are presented in Section IV that include a description of the benchmark instance generation, algorithmic setting parameters, and presentation and discussion about the results. Finally, conclusions and future work are presented in V.

II. Description of the problem

The system is composed by a set of vehicles and customers with specific associated demands. Two types of costs concerning vehicles are considered: fixed cost and variable cost which depends on the distances between customers. Each vehicle can cover one or more routes. The route of a vehicle is as follows. It starts from a depot, next it assists a set of customers satisfying their demands, and it return to the initial depot once customer's demand all along the route were satisfied. Once a vehicle finishes its route it can, eventually, start another one. Each customer must be assigned to one and only one vehicle caring for not exceeding the vehicle capacity.
Normally, the problem is described in graph terms where nodes represent the customers and arcs represent routes between them. Two kind of entities can be identified in the system: *customers* and *depots*. Let N_{v} be the number of customers in the system. Each customer v has an associated *demand* d_v, which must be satisfied by any vehicle within a specific *time window* $[t_v^{\min}, t_v^{\max}]$. The *service time* is the time required for serving a customer v, we denote it by s_v. We simplify the problem assuming that there are only one depot in the system which is denoted by v^{dep}. Let N_{u} be the number of vehicles in the system. Two parameters are associated to each vehicle u, its *capacity*, denoted by q_u, and its *fixed cost*, denoted by f_u. Each vehicle u has a *travel cost* and a *travel time* for traveling from the customer v_i to the customer v_j, which are denoted as $c_u(v_i, v_j)$ and $t_u(v_i, v_j)$, respectively. A route r is a circuit in the graph which starts and finishes at the depot satisfying a set of customer's demands all along the route. We denote a route r by a sequence of nodes, for instance $r = \{v^{\mathrm{dep}}, v_i, v_j, \dots, v^{\mathrm{dep}}\}$. Without loss of generality, we arbitrary enumerate the customers within a route r from 1 to N_{r} ($r = \{v_1, \dots, v_{N_{\mathrm{r}}}\}$), where N_{r} is the number of customer in r. We define two auxiliary functions associated to the routes: the demand required over a route $d(\cdot)$ and the time required to cover a route using the vehicle u $t_u(\cdot)$:

$$d(r) = \sum_{v \in r} d_v, \qquad t_u(r) = \sum_{i=1}^{N_{\mathrm{r}}-1} t_u(v_i, v_{i+1}) + s_{v_i+1}.$$

Note that, by definition $v_1 = v_{N_{\mathrm{r}}} = v^{\mathrm{dep}}$. In order to avoid trivial cases we assume that $N_{\mathrm{r}} > 1$.
Let $a_u(v)$ be the *arrival time* of the vehicle u to the customer v. A temporal global constraint called *time horizon* T^{hor} represents the planning period. All routes must be covered satisfying the customer's demands in a time lower than T^{hor}.
We consider a *solution* of the problem as a set of pairs (route, vehicle). We define the cost function for a specific solution using information about the routing costs and penalization terms. Given a solution S, we start defining the penalization terms. We define the *overtime penalization* when the solution exceeds the time horizon:

$$T(S) = \max\Big\{0, \sum_{(r,u) \in S} t_u(r) - T^{\mathrm{hor}}\Big\}. \qquad (1)$$

We define the *overload penalization* for the solution S as follows:

$$Q(S) = \sum_{(r,u) \in S} \max\{0, d(r) - q_u\}. \qquad (2)$$

Given a route r covered by u and two consecutive customers v_{i-1} and v_i in r, we define $w_u(r, v_i)$ as

$$w_u(r, v_i) = \begin{cases} 0, & \text{if } a_u(v_i) \in [\xi t_{v_i}^{\min}, t_{v_i}^{\max}], \\ t_{v_i}^{\min} - (a_u(v_{i-1}) + s_{v_{i-1}} + t_u(v_{i-1}, v_i)), & \text{if } a_u(v_i) < \xi t_{v_i}^{\min}, \\ a_u(v_{i-1}) + s_{v_{i-1}} + t_u(v_{i-1}, v_i) - t_{v_i}^{\max}, & \text{if } a_u(v_i) \geq t_{v_i}^{\max}, \end{cases}$$

where ξ is a real parameter in $(0, 1]$ that brings some flexibility allowing vehicles to arrive before $t^{\min}$ without considering this as a constraint violation ($\xi < 1$). In the case that $\xi = 1$, any vehicle arriving to customer v_i out of the time window $[t_{v_i}^{\min}, t_{v_i}^{\max}]$ will be considered as a time window constraint violation and the objective function will be penalized. Thus, the time windows violation within the route r using the vehicle u is given by

$$w_u(r, \cdot) = \sum_{i=1}^{N_{\mathrm{r}}-1} w_u(r, v_i). \qquad (3)$$

Therefore, we define the *time windows penalization* parameter as

$$W(S) = \sum_{(r,u) \in S} w_u(r, \cdot). \qquad (4)$$

The system has associated a *cost function* $C(S)$, defined as

$$\begin{aligned} C(S) = {} & \sum_{u \in U} f_u \sum_{v_i \in \mathrm{adj}(\mathrm{v^{dep}})} x_u(v^{\mathrm{dep}}, v_i) \\ & + \sum_{u \in U} \sum_{i \neq j}^{N_{\mathrm{v}}} c_u(v_i, v_j) x_u(v_i, v_j) \\ & + \lambda_1 T(S) + \lambda_2 Q(S) + \lambda_3 W(S), \end{aligned} \qquad (5)$$

where $\mathrm{adj}(\mathrm{v}^{\mathrm{dep}})$ represent the list of adjacent customers of the depot, $x_u(v_i, v_j)$ is a binary function that indicates whether the vehicle u covers the arc (v_i, v_j) (in which case$x_u(v_i, v_j) = 1$, otherwise $x_u(v_i, v_j) = 0$), and λ_1, λ_2 and λ_3 are control parameters in $[0, 1]$. The control parameters, also called *penalized terms* in [9], are updated each h iterations of the searching algorithm. We follow [9] for defining the arbitrary parameter h and updated rule of λ_1, λ_2 and λ_3.
The final solution S (a set of pairs of routes and vehicles) should be a *feasible solution*, that is, the vehicles covering the routes in the solution must not violate the time window, vehicle capacities and time horizon constraints. Besides, the solution must be locally optimal and must also satisfy the following rules: each route starts and finishes at the depot v^{dep}. Each customer must be visited only once by a unique vehicle. Given a vehicle u assigned to the route r, the whole route demand must not exceed the capacity q_u. All customer's demands must be satisfied. Customers must be satisfied before the time horizon T^{hor}. Table 1 presents the nomenclature used in this article, the table includes the main variables involved in the system. Figure 1 illustrates an example of a feasible solution with the particular case that one vehicle does not belongs to the solution while another one participates in more than one.

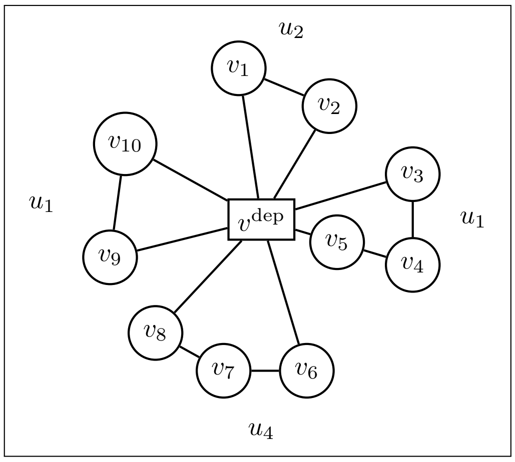


Figure. 1: Routing example. There are a set of four vehicles $\{u_1, u_2, u_3, u_4\}$ and a set of customers $\{v_1, \ldots, v_{10}\}$. The figure shows a feasiblesolution. Notice that, it would be possible that some vehicles do notparticipate in any route. In the figure, the vehicle u_3 does not participate of the solution. Inaddition, a vehicle can be able to cover more than one route. For instance, vehicle u_1 cover two routes.

III. Methodology

Our approach for solving this optimisation problem is based on the *Simulated Annealing (SA)* technique and in an ad-hoc local search procedure. The SA algorithm is a stochastic version of the gradient descent method and is normally used for solving both combinatorial and continuous optimization problems. It has been used for solving the *Traveling Salesman Problem*, as well as for optimising continuous functions in a multidimensional space [7]. The technique is mainly useful when the goal is finding a global extremum that is hidden among several local extrema. The procedure has a parameter called *temperature* (F) arising from a thermodynamics analogy. Besides, there is a constant called the *Boltzmann's constant* that relates temperature with the *energy* of the current system state.

Table 1: Table summarises the main notation used in this article.

Notation	Definition	Notation	Definition
v	customer	T^{hor}	time horizon
u	vehicle	f_u	cost of u
S	proposal solution	r	route
N_u	vehicles in the system	N_u	customer over r
v^{dep}	deposit	s_v	service time of v
q_u	capacity of u	d_v	demand of v
$[t_v^{\mathrm{min}}, t_v^{\mathrm{max}}]$	time windows of v	$T(S)$	time of S
t_r	time of route r	$a_u(v)$	arrival time of u to v
$t_u(v_i, v_j)$	time of u to travel from v_i to v_j	$c_u(v_i, v_j)$	cost of u to travel from v_i to v_j

The algorithm follows an iterative strategy: given the current solution S^{curr}, the energy (cost function) of the system $C(S^{\mathrm{curr}})$ is computed by means of the expression (5). A new solution S^{new} is selected with a probability

$$p = \min\{\exp\left(-(C(S^{\mathrm{new}}) - C(S^{\mathrm{curr}}))/kF\right), 1\}, \qquad (6)$$

where k is the Boltzmann's constant. The probability brings the capacity to jump from a local optimum to another part of the searching space, and to explore the searching space until reaching a better local optimum.
Each h iterations of our algorithm, the cost function is updated according to the information about the last h solutions. The update rule of the cost function was done according to [9]. The penalized terms of the cost function λ_1, λ_2 and λ_3 were updated as follows: we check whether the last h solutions were feasible in respect to the time horizon violation or not (T(S), see (1)). If so, then we update λ_1 as $\lambda_1 = 0.5\lambda_1$. In a similar way, if the last h solutions were feasible with respect the vehicle capacity violation (Q(S), see (2)), then we update $\lambda_2 = 0.5\lambda_2$. If the last h solutions were all feasible in respect the time windows violation (W(S), see (4)), then we update $\lambda_3 = 0.5\lambda_3$. If all the last h solutions were unfeasible in respect of time horizon, capacity, and time windows, the update rule is given by $\lambda_1 = 2\lambda_1$, $\lambda_2 = 2\lambda_2$ and $\lambda_3 = 2\lambda_3$, respectively.
The rest of this section is laid out as follows: subsection III-A describes the procedure for selecting an initial solution based on a well-known heuristic called *Solomon Insertion Heuristic* [2]. Subsection III-B, we describe how to apply SA and local search technique for obtaining a *good* solution for our problem.

A. Finding an Initial Solution

In combinatorial optimization problems, the starting points of the algorithm can impact in the final solution. Therefore we define an *initial solution* using a well-known insertion heuristic named the *Solomon Insertion Heuristic* [2]. However, the approach does not strictly depends on this initial heuristic, in the sense that it would be possible to consider also another strategy. The algorithm starts by selecting a random customer v^{new} and a vehicle u with the least fixed cost f_u. At each step, the algorithm inserts a non-visited customer in the current route until the vehicle capacity is completed. We define the selection of a non-visited customer by prioritizing those customers that are far from the depot over those closer to it. The selection of a customer is done as follows. Given a partially completed route composed by a set of customers $r = \{v^{\mathrm{dep}}, \ldots, v_{N_{\mathrm{r}}}\}$ where $v^{\mathrm{dep}} = v_{N_{\mathrm{r}}}$, the algorithm looks for a non-assigned customer v^{new}, to be inserted between two consecutively customers v_i and v_{i+1}, that minimizes the *insertion cost* $c_u^{\mathrm{ins}}(v_i, v^{\mathrm{new}})$. This cost is computed as follows. Firstly, we compute an auxiliary function $c_u^{'}(\cdot)$:

$$c_u^{'}(v_i, v^{\mathrm{new}}) = (1-\alpha)(a_u^{v^{\mathrm{new}}}(v_{i+1}) - a_u(v_{i+1})) \\ +\alpha(dist(v_i, v^{\mathrm{new}}) + dist(v^{\mathrm{new}}, v_{i+1}) - dist(v_i, v_{i+1})), \tag{7}$$

where α is a constant in $[0,1]$, $dist(\cdot)$ is some arbitrary distance function, and $a_u^{v^{\mathrm{new}}}(v_{i+1})$ denotes the arrival time to node v_{i+1} when the customer v^{new} was inserted before it. In our empirical results, the euclidean distance is used.
Then, v^{new} is selected based on the insertion cost given by

$$c_u^{\mathrm{ins}}(v_i, v^{\mathrm{new}}) = dist(v^{\mathrm{dep}}, v^{\mathrm{new}}) - c_u^{'}(v_i, v^{\mathrm{new}}). \tag{8}$$

The selection process is repeated until the capacity of the assigned vehicle is completed. Once the first route is created and in case there are still non-assigned customers, it is necessary to create a new route r^{new} in order to cover demands for all non-visited customers. The first customer to be assigned to r^{new} is the one whose time window finishes earlier. Each time a new route is created, a vehicle must also be selected. The algorithm proceed as follows: first it checks whether some of the previously considered vehicle is able to cover the new route. This check is always done from lowest to highest cost. If so, then selected vehicle is assigned to the new route. Otherwise, the algorithm selects the next lowest fixed cost vehicle from the set of unused vehicles. If there are not unused vehicles, then the selection is a random vehicle from the set of considered vehicles, probably generating a non feasible solution. This procedure provides an initial solution to start from which is not necessary a feasible solution as explained before.

B. Description of our Algorithm

Once the algorithm has an initial solution $S^{(0)}$ to start from, the following step is to improve this solution by applying the *Local Search* and *Simulated Annealing* techniques. The algorithm starts from an initial solution with an initial temperature F. At each iteration, we replace the current solution by a random nearby solution S^{new}, chosen with a probability p given by the expression (6). The temperature F decreases at each iteration until some arbitrary value F_z is reached. An important issue to consider in this type of procedures is related to the mechanism for exploring the neighborhood for a particular solution S.
By reorganizing the customers and routes, a nearby solution S^{new} is achieved. We use three *movements* that consist in some customer reorganization or by changing the assigned vehicle to some particular route. Specifically, the operations are:

- *Moves within a route*: Given a specific route r, we select a set of consecutive customers and revert the order in which they are visited. For instance, the customers (v_2, v_3, v_4) are selected from the original route $\{v_1, v_2, v_3, v_4, v_5\}$, then the operation inverts the order in which they are visited. The example is illustrated in the figure 2.
- *Moves between routes*: These moves are based on *customer exchange* between specific routes [10], there are three kinds:
  - *Move one customer*: random selection of two routes r_i and r_j. Then, a random customer v from r_i is selected, removed of r_i and inserted at the end of the customer list visited by r_j. Figure 3 shows an example of moving one customer from one route to another one. In this example, given two routes $r_1 = \{v_1, v_2, v_3\}$ and $r_2 = \{v_4, v_5, v_6\}$, the customer v_4 from r_2 is inserted at the end of the route r_1.
  - *Move two customers*: random selection of two routes r_i and r_j and two consecutive customers from r_i. Then, both customers are removed from r_i and inserted in r_j in the same order than in the original route. Figure 4 shows an example of this operation. In the figure there are two routes $r_1 = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and $r_2 = \{v_7, v_8, v_9\}$, the customers v_5 and v_6 from r_1 are inserted in the same order in the route r_2.
  - *Exchange of customers*: random selection of two routes r_i and r_j, and customers v_i and v_j from r_i and r_j respectively. Then, both customers are exchanged between the routes. Figure 5 shows an example of exchanging between two customers in two routes. Given two routes $r_1 = \{v_1, v_2, v_3, v_4\}$ and $r_2 = \{v_5, v_6, v_7\}$ the customer v_4 from r_1 is exchanged with the customer v_5 of the route r_2.
- *Assignation moves*: this operation consists in changing a vehicle assigned to a given route r with another available vehicle.

Furthermore, we define two operations (for special purposes) for controlling the constraint violation on a route r. The operations are:

- *Divide overloaded route*: it is applied when at least one route of the current solution has violated the capacity constraint of the assigned vehicle. Then, in this operation we select an overflowed route r^{ov}, and next we remove the customer with greater demand in r^{ov}. We insert the removed customer in the less loaded route in the current solution.
- *Overflow in planning*: the operation is applied if it exists at least one route which finishes its trip beyond the specified time horizon. The operation selects the overflowed route r^{ov}, picks the last visited customer and inserts it in another non-overflowed route (random selected) which ends its trip before the specified time horizon.

During the execution of the algorithm, at each step a movement operation is selected using uniformly distribution. If S is a feasible solution, moves are selected based on a uniform

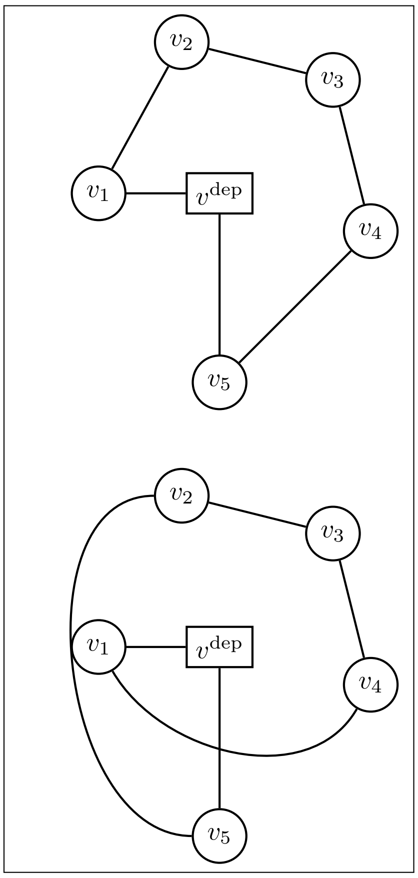


Figure. 2: Example of moves within a route. Given an original route with $\{v_1, v_2, v_3, v_4, v_5\}$ illustrated in the figure at the top, the customers $\{v_2, v_3, v_4\}$ are selected from the original route and then it is inverted its order. So the new route is $\{v_1, v_4, v_3, v_2, v_5\}$.

distribution except those for special purpose which are not considered in this case. On the other hand, if S is unfeasible, moves for special purpose are prioritized over the others in order to force solutions to remains in the feasible space of solutions. Hence, if some of the vehicles are overloaded then the movement *divide overloaded route* is prioritized, as well as the *assignation move* that would help to avoid constraint violation. In the case that some of the routes in the solution S finishes the trip beyond the specified time horizon, we apply the *overflow in planning* operation. Algorithm 1 summarizes the heuristic procedure for finding a solution of our problem. The algorithm has the following input parameters: number of transitions for each temperature $Iter$, an initial temperature $F^{(0)}$, a cooling schedule ρ in $[0, 1]$, the stop condition F_z, control of the penality adjustment h, and the output is a set of pairs (route, vehicle).

IV. Results and Discussions

In this section we present the empirical results obtained for a set of benchmark instances. Ideally, the best way for comparing our approach is to find instances for the same problem but solved with a different methodology in such a way to be able to compare our results with the ones obtained with another methodology. However, we were unable to carry out this comparison since, to the best of our knowledge, we could not find a resolution for the same problem by means of a different approach. In this way, we were forced to generate our own benchmark dataset instances and to find a way to compare our results over those instances. In the remains of this section, we explain the procedure for generating the benchmark

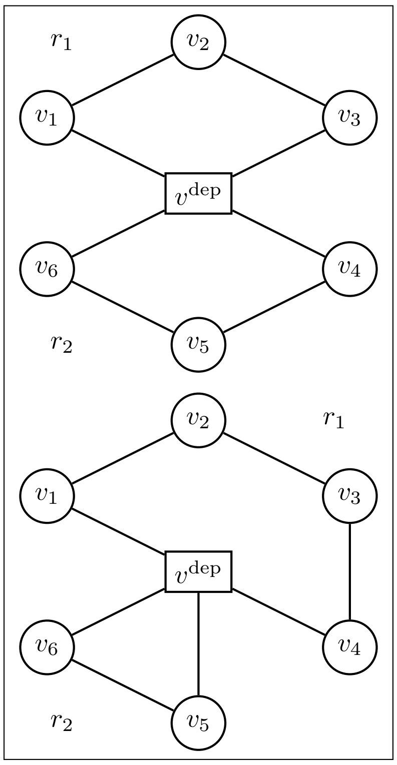


Figure. 3: Example of moving one customer. Given two routes $r_1 = \{v_1, v_2, v_3\}$ and $r_2 = \{v_4, v_5, v_6\}$ illustrated in the figure at the top. The customer v_4 from r_2 is inserted at the end of the route r_1.

Algorithm 1: Local Search and Simulated Annealing based algorithm applied to the MTVRP-TWHF.

```
1  $i \leftarrow 0$;  $F \leftarrow F^{(0)}$;
2  Compute an initial solution $S^{(0)}$ using Subsection III-A;
3  $S^{\mathrm{curr}} \leftarrow S^{(0)}$;
4  while ($F \geq F_z$) do
5      while ($i <= Iter$) do
6          Select a random nearby solution $S^{\mathrm{new}}$ using
           Subsection III-B;
7          if ($i$ is a multiple of $h$) then
8              Control $\lambda_1$, $\lambda_2$ and $\lambda_3$ following III;
9          if ($C(S^{\mathrm{new}}) \leq C(S^{\mathrm{curr}})$) then
10             $S^{\mathrm{curr}} \leftarrow S^{\mathrm{new}}$;
11         else
12             Compute $p$ using expression (6);
13             if ($rand(0, 1) < p$) then
14                 $S^{\mathrm{curr}} \leftarrow S^{\mathrm{new}}$;
15         $i \leftarrow i + 1$;
16     Decrease temperature: $F \leftarrow \rho F$;
17     $i \leftarrow 0$;
18 Return $S^{\mathrm{curr}}$;
```

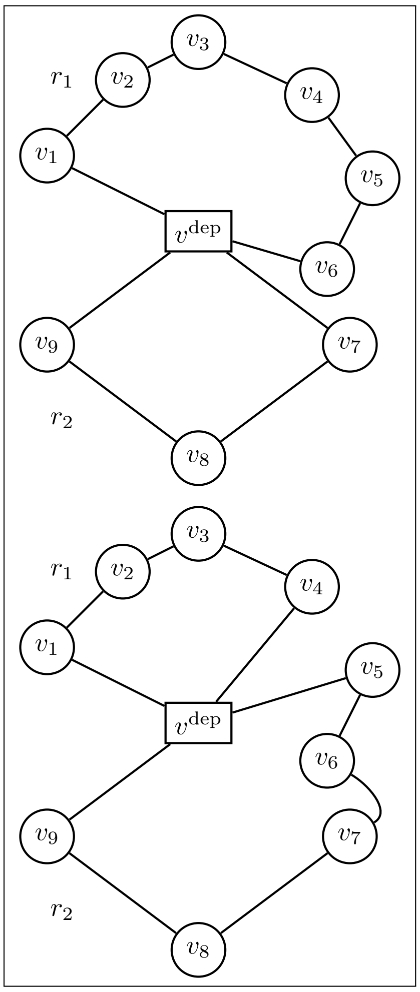


Figure. 4: Example of moving two customers. Given two routes $r_1 = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and $r_2 = \{v_7, v_8, v_9\}$ illustrated in the figure at the top. The customers v_5 and v_6 from r_1 are inserted in the same order in the route r_2.

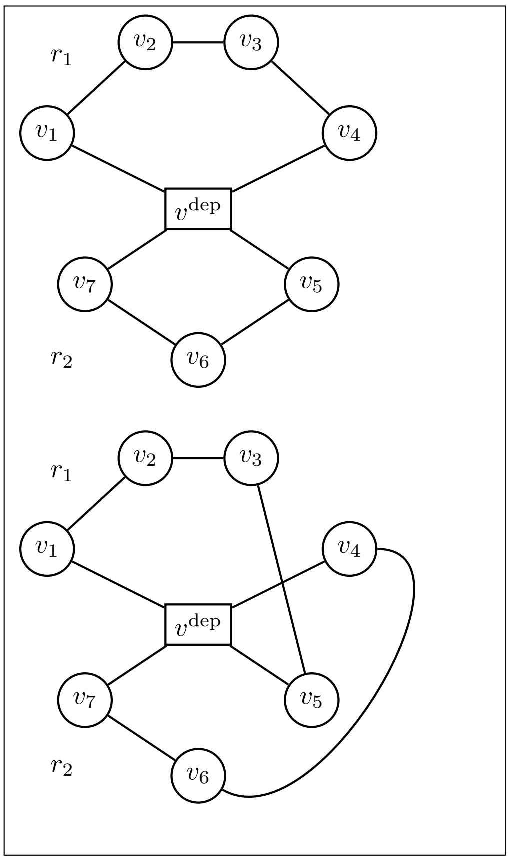


Figure. 5: Example of exchanging of customers. Given two routes $r_1 = \{v_1, v_2, v_3, v_4\}$ and $r_2 = \{v_5, v_6, v_7\}$ illustrated in the figure at the top. The customer v_4 from r_1 is exchanged with the customer v_5 of the route r_2.

instances. Next, how our algorithm was set up by calibrating its parameters. Finally, we present our algorithm's results.

A. Description of Benchmark Instance

As it was already mentioned above, the lack of existing works for the problem presented in this paper has restrained us to compare our results with existing ones for the same problem. In this way, we were forced to define a strategy for generating the set of instances and to have references values for comparing our results. Therefore, what we have done was to use a set of instances proposed by Golden *et al.* [11] and Taillard [12] and reference results from Gendreau [9] and Tarantilis [13] over these instances, were taken as reference values. The proposed instances were originally defined for a particular VRP where neither time windows nor time horizon restrictions were taken into account. In this way, it was necessary for us to adapt these instance to our problem by including both time windows and time horizon restrictions. Next, we explain how these instances were adapted to our problem. A total of 30 instances involving 20, 50, 75 and 100 customers were selected. An instance correspond to a particular configuration scenario. To define an instance we should specify the capacity of each vehicle, the fixed cost of them, client's demand, time windows for each customer as well as the T^{hor}. The Tables 2 and 3 show the number of nodes and number of vehicles (respectively) defined in each instance B_i for all i from 1 to 10. The number of nodes and vehicles of the instances ${B_i}^{\text{tw1}}$ and ${B_i}^{\text{tw2}}$ is the same that the number in B_i.

In the following we present how we generated the benchmark instances. Time window restrictions were added in the following way: for each benchmark instance B_i, two instances ${B_i}^{\text{tw1}}$ and ${B_i}^{\text{tw2}}$, each one with different time window, were created. The instance B_i defines the same time window for each customer. The instance ${B_i}^{\text{tw1}}$ defines two time windows $w^{\text{tw1}} = [t_0, t_k]$ and $w^{\text{tw2}} = [t_l, T^{\text{hor}}]$ assigning w^{tw1} to a subset of customers and w^{tw2} to the rest. The values chosen for t_k and t_l were:

$$t_k = \frac{1}{3}T^{\text{hor}} \qquad \text{and} \qquad \mathrm{t_l} = \frac{2}{3}\mathrm{T}^{\text{hor}}.$$

The instance ${B_i}^{\text{tw2}}$ defines three time windows w^{tw1} and w^{tw2} defined as above and $w^{\text{tw3}} = [t_h, t_p]$, where t_h and t_p were chosen as

$$t_h = \frac{1}{4}T^{\text{hor}} \qquad \text{and} \qquad \mathrm{t_p} = \frac{3}{4}\mathrm{T}^{\text{hor}}.$$

The set of customers is divided in three equals subsets where we have assigned to each of them the time windows w^{tw1}, w^{tw2} and w^{tw3}.

B. Parameter Calibration

As described in Algorithm 1, there are a set of input parameters concerning the Simulating Annealing algorithm. In order to evaluate a solution we have to take into account, not only the quality in terms of the cost, but also the execution time taken for the algorithm to achieve it. A high computational time can be prohibitive in several applications where achieving a reasonably solution in a short period of time, is mandatory. On the other hand, an algorithm that finds an optimal solution in terms of the execution time, may not be a cost optimal solution. Therefore, a good balance of both quality and execution time is needed. In the case of our algorithm, the quality in terms of cost and execution time are strictly bounded to the set of input parameters and the calibration of them is crucial to obtain "good" solutions. In this way, we have done multiple runs by combining different input parameters in order to obtain a set of them that provide balanced time-accuracy solutions. The input parameters are the number of transitions $Iter$, the cooling schedule ρ, the initial temperature $F^{(0)}$, the stop condition F_z, and the frequency or updating control h. The following sets were chosen for each of them: $Iter = \{1, 5\}$, $\rho = \{0.99, 0.999, 0.9999\}$ and $F = \{10, 100, 1000\}$, $F_z = \{0.01, 0.001\}$, and a random set of instances were selected in order to run the parameter calibration process. The penalized terms of the cost function λ_1, λ_2 and λ_3 were updated following [9]. The frequency of penalized terms update was set to $h = 5$.
Executions were done by permuting each of these parameters and evaluating both cost and time spent. After finishing this process we obtained the best configuration of parameters for the algorithm that gives balanced solutions in terms of cost and execution time. Respective values are $F = 10$, $Iter = 1$, $\rho = 0.9999$, and $F_z = 0.01$.

C. Empirical Results

Results of our algorithm were compared using the GAP measure with the set of reference values obtained by [9, 13]. Being CV_i the comparative value for instance i and being z_i the solution obtained by the algorithm, then

$$GAP(z_i) = \frac{z_i - CV_i}{CV_i}. \tag{9}$$

For each benchmark instance, 10 independent runs of the algorithm were performed. We call *Initial GAP* to the worst and average values obtained by the initial solution. In the same way, we call *Final GAP* to the average and worst GAP value of the solution obtained by the SA method. The results are summarized in Table 4. Table 4 shows the name of the instance i, *GAP* between the reference value CV_i obtained by [9, 13] over the *VRPHF*, *Initial Gap*, *Final Gap*, and the execution time (in seconds) that takes to reach the final solution. The computer used to simulate the algorithm has 2 gigabytes of RAM and a *AMD Athlon 64 x 2 Dual-Core Processor TK-57, 1.90 GHz* processor. The algorithm was implemented in *C#*.
To evaluate the effectiveness of our algorithm in a practical scenario, we have adapted a widely used set of instances defined for the VRPHF problem [11]. In this way, we cannot make a directly comparison between our results and those obtained by [9, 13], since their results were obtained for a less restrictive problem. Nevertheless, and since reference values were obtained for a problem with less constraints than the MTVRP-TWHF, we can use these values as lower bound for those obtained in our empirical results. Using the *GAP* measure we can assess how far from these lower bounds our results are. According to the results in Table 4, we can remark the effectiveness of the Algorithm 1 for improving the initial solution described in Subsection III-A. This can be seen by comparing the average from *Initial GAP* and *Final GAP* columns. In all cases, the algorithm is improving the first initial solution. Regarding the *Final GAP* results, *a priori*, is not straightforward to determine whether the solution is qualitatively *good* or *not* since we do not know where the optimum value for MTVRP-TWHF for the current solution is. However, we can evaluate the quality of the solution in terms of cost by looking at the comparative value CV_i for each instance i. Let z^{opt} be the global optimal solution for

Table 2: Number of nodes for each benchmark instance.

Number of nodes	Benchmark instances
20	B_1, B_2
50	B_3, B_7, B_8
75	B_4, B_5, B_9
100	B_6, B_{10}

Table 3: Number of vehicles for each benchmark instance.

Number of vehicles	Benchmark instances
3	B_1, B_2, B_3, B_7, B_8, B_{10}
4	B_4
6	B_5, B_6, B_9

Table 4: Empirical results reached for the MTVRP-TWHF using Simulating Annealing for several benchmark instances.

Instance	Initial GAP		Final GAP		Time
ID	Average	Worst Case	Average	Worst Case	(sec)
B_1	47.1	57.5	21.3	24.6	34
B_1^{tw1}	48	50.8	22.1	23.8	35
B_1^{tw2}	47.9	52.4	44.4	49.9	37
B_2	23.8	27.6	13.9	16.6	32
B_2^{tw1}	155334.7	175400	14.3	16.3	39
B_2^{tw2}	157199	166557	21.1	22.7	33
B_3	7277.4	76770.9	1.8	5.1	139
B_3^{tw1}	762	779	5.4	9.4	130
B_3^{tw2}	2942.1	3136	7.1	9.1	124
B_4	18338	19135	3.8	9.4	265
B_4^{tw1}	43491.5	43915.6	14.4	16.8	255
B_4^{tw2}	648409.4	715872.2	9.5	13.6	248
B_5	158847.1	165021.8	5.0	13.4	251
B_5^{tw1}	1179376	1271993.7	36.4	40	252
B_5^{tw2}	269683.1	272924.4	9.5	13.8	257
B_6	75.5	81.0	7.5	10.3	489
B_6^{tw1}	58.9	62.7	15.3	18.1	419
B_6^{tw2}	90575.1	93170.2	12.3	14.1	437
B_7^{tw1}	73	80.4	5.8	8.4	176
B_7^{tw1}	57.1	65.2	10.7	12.2	149
B_7^{tw2}	58	62.3	8.5	11.5	149
B_8	57.5	71.8	4.8	15	190
B_8^{tw1}	2009614.7	2168118.1	10.1	14.3	131
B_8^{tw2}	2631736.7	2873025.5	14.4	22.5	126
B_9	197714.4	202380.5	22.1	24.2	211
B_9^{tw1}	1355381.8	1502515.1	33.4	36.0	194
B_9^{tw2}	427733.4	432048.3	34.1	39.9	207
B_{10}	62.3	68.2	15.5	20	482
B_{10}^{tw1}	2667581.1	2678442	20.1	25.1	424
B_{10}^{tw2}	261982.2	262449	23.4	26.7	391

the MTVRP-TWHF, and let z_i be the local solution found using our approach. We assume that the following expression always holds

$$CV_i <= z^{\mathrm{opt}} <= z_i, \tag{10}$$

then, the narrower the interval $[CV_i, z_i]$ is, the closer to the optimum value the solution is. Thus, those results having small GAP when comparing them with the reference value are likely to be next to the optimal value z^{opt} and therefore, likely to be a good solution in terms of cost.
Figure 6 illustrates the relationship among the values on the expression (10). Regarding the execution times, we can see that even for big instances (100 customers) the algorithm obtains execution times with an average time of 440 seconds which is a reasonable time value for instances of that size. For small instances (20 customers), the algorithm obtains GAPs of the order of 13% and 14% with an average time of 35 seconds.

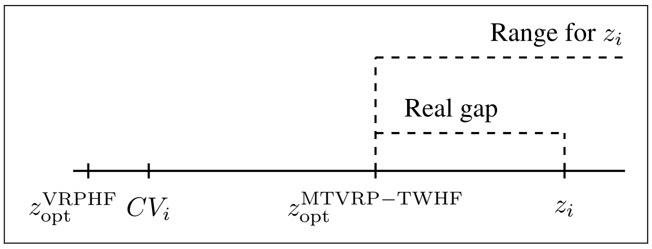

Range for z_i
Real gap
$z_{\mathrm{opt}}^{\mathrm{VRPHF}}$
CV_i
$z_{\mathrm{opt}}^{\mathrm{MTVRP-TWHF}}$
z_i

Figure. 6: Relationship between the reached values of the VRPHF optimal value and the optimum value of the problem analysed in this article. The horizontal line represents the time. The diagram shows the relationship among the values on the expression (10).

V. Conclusions and Future Works

The *Vehicle Routing Problem (VRP)* and its variations probably are the most significant problems in the history of *Operations Research (OR)* area. Over the last 60 years many works studying those problems have been introduced in the community. In this article, we revisited the VRP introducing new challenges. We called this new variation of VRP as *Multi-Trip Vehicle Routing Problem with Time Windows and Heterogeneous Fleet (MTVRP-TWHF)*. We added new constraints to the original problem, which concern to the delivery time and the routing in the system. We introduced a hybrid algorithm that mixes concepts of *Simulating Annealing* and *Local Search* techniques for solving it. In order to evaluate the performance of our approach, we generated a benchmark dataset based on instances widely used in the OR community. According to the early empirical results, our algorithm presents a promising performance. Finally, and considering the fact that no benchmark solutions exist for the presented problem, this paper provides a benchmark dataset which may be used for future references.
As a future work, we plans to test our algorithm with a new benchmark instances with a larger number of nodes. In addition, some adaptations can be made to the algorithm. Particularly, there is a variation of the Simulated Annealing in which the algorithm parameters that control the temperature schedule is dynamically adjusted during the execution of the algorithm. This makes the algorithm more efficient and less sensitive to user defined parameters than the original SA. A second variation of the Simulated Annealing is known as Simulated Annealing with threshold accepting. Besides, we are interesting in comparing the reached performance by our approach with other metaheuristic techniques for solving combinatorial problems. The approach presented in this paper is clearly useful in industry. As a consequence, we plan to verify our algorithm using real benchmark problems.

Acknowledgments

This work was supported by Grant of SGS No. SP2016/97, VŠB - Technical University of Ostrava, Czech Republic.

References

[1] G. Clarke, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations research : the journal of the Operations Research Society of America.*, 1964.

[2] M. M. Solomon and M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations Research*, vol. 35, pp. 254–265, 1987.

[3] E. Zachariadis, C. Tarantilis, and C. Kiranoudis, "Designing vehicle routes for a mix of different request types, under time windows and loading constraints," *European Journal of Operational Research*, no. 2, pp. 303–317, 2013.

[4] S. N. Kumar, "A Survey on the Vehicle Routing Problem and Its Variants," *Intelligent Information Management*, vol. 04, no. 03, pp. 66–74, 2012.

[5] N. Azi, M. Gendreau, and J.-Y. Potvin, "An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles," *European Journal of Operational Research*, vol. 202, no. 3, pp. 756 – 763, 2010.

[6] R. Macedo, C. Alves, J. V. de Carvalho, F. Clautiaux, and S. Hanafi, "Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model," *European Journal of Operational Research*, vol. 214, no. 3, pp. 536 – 545, 2011, http://dx.doi.org/10.1016/j.ejor.2011.04.037.

[7] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C*, 2nd ed. Cambridge, UK: Cambridge University Press, 1992.

[8] F. Despaux and S. Basterrech, "A Study of the Multi-Trip Vehicle Routing Problem with Time Windows and Heterogeneous Fleet," in *IEEE International Conference on Intelligent Systems Design and Applications (ISDA)*, December 2014, pp. 7–12, doi: 10.1109/ISDA.2014.7066280.

[9] G. L. Michel Gendreau, Alain Hertz, “A tabu search heuristic for the vehicle routing problem,” *Management Science*, no. 10, pp. 1276–1290, 1994.

[10] A. Van Breedam, “Improvement heuristics for the vehicle routing problem based on simulated annealing,” *European Journal of Operational Research*, vol. 86, no. 3, pp. 480–490, 1995.

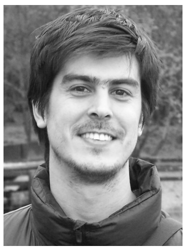
[11] B. L. Golden, A. A. Assad, L. Levy, and F. Gheysens, “The fleet size and mix Vehicle Routing Problem,” *Computers & OR*, pp. 49–66, 1984.

[12] E. D. Taillard, “A Heuristic Column Generation Method for the Heterogeneous Fleet VRP,” *RAIRO- Operations Research - Recherche Opérationnelle*, vol. 33, no. 1, pp. 1–14, 1999, url: http://eudml.org/doc/105182.

[13] C. Tarantilis, C. Kiranoudis, and V. Vassiliadis, “A list based threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem,” *Journal of the Operation Research Society*, no. 1, pp. 65–71, 2003.

Author Biographies

François Despaux has obtained his computer engineer degree at the Republic University, Montevideo, Uruguay in February 2009. He has continued its career at the “Laboratoire Lorraine de Recherche en Informatique (LORIA)” of Nancy, France, and he has obtained its PhD degree in Computer Sciences at the University of Lorraine, Nancy, France in September 2015. Currently, he is a member of the RT team at the “Institute de Recherche en Informatique” of Toulouse, France. The main research interests of F. Despaux concern the performance evaluation and quality of services in Wireless Sensor Networks, as well as the synchronisation and localisation protocols in UWB-based Wireless Sensor Networks.



Sebastián Basterrech received a M.Sc. degree in Applied Mathematics in 2008 from the Aix-Marseille University, France. From 2009 to 2012 he received a doctoral fellowship from the National Institute for Research in Computer Science and Control (INRIA), France. He obtained the Ph.D. degree in Computer Sciences in November, 2012 from INRIA-Rennes and University of Rennes 1, France. In July, 2013 he received the M.A. degree in Computer Arts at the University of Rennes 2 in France. In the period of May, 2013 until December, 2015 he was a postdoctoral researcher at the National Super Computer Center, Ostrava, Czech Republic. Since January 2016, he is an Assistant Professor at the VŠB-Technical University of Ostrava. He is a TC member of the IEEE SMC Soft-Computing Society. The main research interests of S. Basterrech include Spatio-temporal Data Mining, Recurrent Neural Networks and Bio-inspired Algorithms for Machine Learning Applications.