Heuristic Solutions for the Vehicle Routing Problem with Time Windows and Synchronized Visits

Sohaib Afifi, Duc-Cuong Dang, Aziz Moukrim

manuscript No.
(will be inserted by the editor)

Heuristic Solutions for the Vehicle Routing Problem with Time Windows and Synchronized Visits

Sohaib Afifi · Duc-Cuong Dang · Aziz Moukrim

Received: date / Accepted: date

Abstract We present a simulated annealing based algorithm (SA-ILS) for a variant of the vehicle routing problem (VRP), in which a time window is associated with each client service and some services require simultaneous visits from different vehicles to be accomplished. The problem is called the VRP with time windows and synchronized visits (VRPTWSyn). The algorithm features a set of local improvement methods to deal with various objectives of the problem. Experiments conducted on the benchmark instances from the literature clearly show that our method is fast and outperforms the existing approaches. It produces all known optimal solutions of the benchmark in very short computational times, and improves the best results for the rest of the instances.

Keywords vehicle routing · synchronization · destruction/repair · local search · simulated annealing.

Introduction

The vehicle routing problem (VRP) [24] is a widely studied combinatorial optimization problem in which the objective is to plan optimal tours for a set of vehicles serving a set of customers geographically distributed and respecting some constraints. We are interested in a particular variant of VRP, the VRP with time windows and synchronized visits (VRPTWSyn). In this problem, each client is

A preliminary version [1] of this paper was presented at the conference Learning and Intelligent OptimizatioN (LION 7), 2013.

Sohaib Afifi · Aziz Moukrim
Université de Technologie de Compiègne
Laboratoire Heudiasyc, UMR 7253 CNRS, 60205 Compiègne, France
E-mail: {sohaib.afifi,aziz.moukrim}@hds.utc.fr

Duc-Cuong Dang
University of Nottingham
ASAP Research Group, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
E-mail: duc-cuong.dang@nottingham.ac.uk

associated with a time window, e.g. an interval of time representing the availability of the client to starting receiving the vehicle service. If the vehicle arrives too soon it should wait until the opening of the time window to serve the client; while late arrivals are not allowed. Additionally, for some clients, more than one visit (e.g. two visits from two different vehicles) are required for the service and those visits need to be synchronized, e.g. having the same start time. VRPTWSyn was first studied in [7] with an application in homecare services for elders. In such services, some operations may require more than one staff to be accomplished, for example the ones demanding heavy lifts or requiring different sets of skills. Timing and coordination are crucial for the success of the operations and the associated temporal constraints must be taken into account during the construction of the schedule.

As an extension of VRP, VRPTWSyn is clearly NP-Hard [17]. To the best of our knowledge, there were only few attempts in the literature to solve this variant of the problem [6, 7] and its generalizations [11, 20]. In those methods, solutions are obtained by approximately or optimally solving integer linear programs. Thus, constructing good solutions often require extensive computational times.

In detail, Bredström and Rönnqvist [7] studied the role of the synchronization constraints found in real world applications and proposed a mathematical formulation of VRPTWSyn. Three objectives of optimization were considered: (i) minimizing the total travel time; (ii) maximizing the sum of preferences, this is due to the fact that each client may like or dislike being served by a specific vehicle; (iii) minimizing the difference between the longest and the shortest service times among the vehicles in order to optimize the workload balance. For convenience, we shall call the three objectives respectively travel *cost*, service *reward* and attribution *fairness*. In practical applications, they can be addressed simultaneously, e.g. by aggregating them into a single objective using a set of weights. To avoid negative weights in the aggregation, the minimization of the sum of the *negative preferences*, e.g. dislike measures, is used instead of the maximization of service reward. The authors proposed a variant of the local-branching approach [14] to solve the problem. A set of benchmark instances was created to evaluate the method. For analytical purposes, e.g. identify the strengths and weaknesses of the approach, the three objectives were studied independently on those instances. As concluded in the paper, fairness appears to be the most difficult to optimize.

As a continuity of [7], the same authors proposed a branch-and-price algorithm focusing the first two objectives, the cost and reward, in [6]. The approach is influenced by the fleet assignment techniques of [15]. In the root node, the synchronization constraints are relaxed and the linear model is basically a set partitioning formulation. Then during the solving steps, the constraints are strengthened with a branch-and-bound. This was done by repeatedly adjusting the arrival times of the vehicles at the clients and by branching on time windows. The branch-and-price algorithm was able to solve 44 out of 60 proposed instances to optimality.

In [20], a similar application in homecare services was studied with a particular focus on the reward objective. The authors proposed a clustering scheme based on the client preferences and a branch-and-price algorithm to solve the problem. The algorithm was able to find good approximate solutions for instances that were not solved to the optimality. The synchronization constraints were modeled as generalized precedence constraints and then reinforced through branching. The

approach was tested on the standard instances of [7], as well as on real-world instances collected from two Danish municipalities.

Later on, Dohn et al [11] presented a generalization of VRPTWSyn, the so-called Vehicle Routing Problem with Time Windows and Temporal Dependencies (VRPTWTD). In addition to the standard synchronization, more general requirements are studied, such as the maximum/minimum overlap and/or gap between the starting time or ending time of visits. On the objective, only travel cost was considered. A branch-and-cut-and-price algorithm was proposed to solve the generalized problem and was tested on a set of instances derived from the 56 well-known instances of Solomon's benchmark for VRPTW [21]. Note that the results reported in both papers [11] and [20] are general statistics, such as the number of instances being solved to the optimality by the proposed method. Readers interested in other variants or applications are referred to [13, 18, 26]. General perspectives of temporal constraints for vehicle routing can be also found in the survey [12].

To summarize, only solving techniques based on linear programming were proposed for VRPTWSyn. Motivated by the potential applications and by the challenge of computational time, we propose, in this work, a Simulated Annealing based algorithm with dedicated local searches (SA-ILS) for solving VRPTWSyn. Our SA-ILS uses several local search methods dedicated to the problem. It produces high quality solutions in very short computational time compared to the other methods of the literature. New best solutions are discovered. A preliminary version of this paper was communicated in [1], in which only dedicated heuristics to optimize the travel cost had been developed and analyzed. The remainder of the paper is organized as follows. Section 1 briefly presents the problem formulation. In Section 2, the detailed description of the proposed SA-ILS algorithm is given. The results of the experimental studies are reported in Section 3. Finally, some concluding remarks are drawn in Section 4.

1 Problem formulation

The problem is modeled using an oriented graph $G = (V^+, A)$, where $V^+ = \{0, \ldots, n+1\}$ is the vertex set and A is the arc set. Vertices 0 and $n+1$ are the departure and arrival points respectively. The other vertices $V = \{1, \ldots, n\}$ are the visit points where each one is associated to a client. A client can have multiple visit points depending on the number of vehicles needed to delivery the service. For example, if two visit points i and j are associated to the same client, then the points are superposed and required to be visited by two distinct vehicles. These two visits must be synchronized and we use $[i, j] \in P^{sync}$ to denote the set of synchronizations. Also, for each i we denote by $P_i^{sync} = \{j \in V^+ \text{ such that } [i, j] \in P^{sync}\}$ the set of visits to be synchronized with.

A travel time δ_{ij} is associated to each arc $(i, j) \in A$. For convenience, we associate an infinite travel time $\delta_{i,i} = +\infty$ ($\delta_{i,j} = +\infty$) to non-existent arcs. Each visit point i is associated with a service time S_i and a time window $[a_i, b_i]$ where a_i and b_i specify the earliest and latest possible starting time of the service ($b_i \geq a_i \geq 0$). For a given client, these data are identical for all the associated visit points. The departure and arrival points are also associated with a time window $[E, L]$ ($[a_0, b_0] = [a_{n+1}, b_{n+1}] = [E, L]$ and $S_0 = 0$).

The fleet of vehicles is denoted by the set $K = \{1, \ldots, m\}$. Related to reward objective, P_{ik} defines the negative preference of the assignment of vehicle k to the service of the client at point i. Let $x_{ijk} \in \{0, 1\}$ $\forall k \in K$ $\forall (i, j) \in A$ be the binary routing variables: x_{ijk} is set to 1 if vehicle k travels along arc (i, j), and to 0 otherwise. Let t_{ik} be the scheduling variables which represent the service starting time of visit i by vehicle k (set to 0 if vehicle k does not serve i). We have the following mixed integer programming model due to [7].

$$\sum_{k \in K} \sum_{j:(i,j) \in A} x_{ijk} = 1 \quad \forall i \in V \tag{1}$$

$$\sum_{j:(0,j) \in A} x_{0jk} = \sum_{j:(j,n+1) \in A} x_{j(n+1)k} = 1 \quad \forall k \in K \tag{2}$$

$$\sum_{j:(i,j) \in A} x_{ijk} - \sum_{j:(j,i) \in A} x_{jik} = 0 \quad \forall k \in K \quad \forall i \in V \tag{3}$$

$$t_{ik} + (\delta_{ij} + S_i) x_{ijk} \leq t_{jk} + b_i (1 - x_{ijk}) \quad \forall k \in K \quad \forall (i, j) \in A \tag{4}$$

$$a_i \sum_{j:(i,j) \in A} x_{ijk} \leq t_{ik} \leq b_i \sum_{j:(i,j) \in A} x_{ijk} \quad \forall k \in K \quad \forall i \in V \tag{5}$$

$$a_i \leq t_{ik} \leq b_i \quad \forall k \in K \quad \forall i \in \{0, n + 1\} \tag{6}$$

$$\sum_{k \in K} t_{ik} = \sum_{k \in K} t_{jk} \quad \forall [i, j] \in P^{sync} \tag{7}$$

$$\sum_{(i,j) \in A} S_i x_{ijk} - \sum_{(i,j) \in A} S_i x_{ijl} \leq \mathcal{W} \quad \forall k \in K \quad \forall l \in K \setminus \{k\} \tag{8}$$

$$x_{ijk} \in \{0, 1\} \quad \forall k \in K \quad \forall (i, j) \in A \tag{9}$$

$$t_{ik} \geq 0 \quad \forall k \in K \quad \forall i \in V^+$$

$$\mathcal{W} \geq 0$$

While constraints (1) ensure that each visit point is served by exactly one vehicle, constraints (2) ensure that every vehicle starts from the departure point and returns to the arrival point. Constraints (3) guarantee that the same vehicle enters and leaves a given client. The connectivity of each tour is guaranteed by (4) and the time windows are respected with constraints (4), (5) and (6). Constraints (7) ensure that synchronized visits start simultaneously. Inequalities (8) record the gap between the longest and shortest service times of the fleet according to the minimization objective. Finally, (9) are variables definition constraints.

The objective is to minimize either the total travel time (10a), the sum of assigned negative preferences (10b) or the difference in the attribution of the workload (10c).

$$\min \quad \sum_{k \in K} \sum_{(i,j) \in A} \delta_{ij} x_{ijk} \tag{10a}$$

$$\min \quad \sum_{k \in K} \sum_{(i,j) \in A} P_{ik} x_{ijk} \tag{10b}$$

$$\min \quad \mathcal{W} \tag{10c}$$

Algorithm 1: Simulated annealing algorithm for VRPTWSyn

```
   Output: X_best, the best solution found so far by the algorithm;
 1 X ← BestInsertion(∅);
 2 X ← LocalSearch(X);
 3 X_best ← X;
 4 T ← T_0; iter ← 1;
 5 repeat
 6     if ( (iter mod n) = 0 ) then
 7         X' ← Diversification(X, n/2, n);
 8     else
 9         X' ← Diversification(X, 1, d);
10     X' ← LocalSearch(X');
11     Δ ← Fitness(X') − Fitness(X);
12     iter ← iter + 1;
13     T ← α × T;
14     r ∼ Unif(0, 1);
15     if (r < e^(−Δ/T)) then
16         X ← X';
17         if (Fitness(X) < Fitness(X_best)) then
18             iter ← 1;
19             X_best ← X;
20 until (iter > itermax);
```

2 Simulated annealing algorithm

Our motivation in this work is to propose a fast dedicated heuristic solution for VRPTWSyn. The global scheme of our approach is a Simulated Annealing algorithm (SA) [16] which integrates a set of dedicated local searches. Recall that SA is a stochastic local search which is often used to address discrete optimization problems. The main idea of a Simulated Annealing algorithm is to occasionally accept degraded solutions in the hope of escaping the current local optimum. The probability of accepting a newly created solution is computed as $e^{-\frac{\Delta}{T}}$, where Δ is the difference of fitness between the new solution and the current one and T is the current temperature. This parameter is initialized to some T_0 then evolved during the search by imitating the cooling process in metallurgy, e.g. controlled by the coefficient α as in a geometric progression. Successful applications of SA-ILS in VRP and its variants can be found in [4, 8, 9, 25].

Our SA-ILS is summarized in Algorithm 1. We use $Fitness()$ to denote the process of computing the objective value according to Equations (10a), (10b) and (10c). SA-ILS memorizes the best discovered solutions so far and stops after a fixed number of iterations without improvement of this solution. This number is set to $m \times n$. The other functions: $BestInsertion(X)$, $Diversification(X, d_{min}, d_{max})$ and $LocalSearch(X')$ are described as follows.

2.1 Constructive heuristic

The procedure $BestInsertion(X)$ is a constructive heuristic to build a solution from scratch ($X = \emptyset$) or from a *partial* solution. A solution is called partial if some visits are not routed. In each iteration of $BestInsertion(X)$, a visit is heuristically selected to be inserted in a route so that the increasing cost is minimized. The algorithm is stopped when no more insertion is possible. The obtained solution can be either complete, i.e. a *feasible* solution with all the visits being routed, or

still partial, i.e. an *infeasible* solution. The later can happen for the instances of VRPTWSyn [7], particularly for the ones with small or strict time windows. In that case, unrouted visits are put in a pool for later attempts and a penalty cost is added to the objective value. This penalty must be big enough to prefer feasible solutions over infeasible ones and it must be proportional to the number of visits in the pool in order to compare infeasible solutions, e.g. $10^3 \times U$ where U is the number of unrouted visits.

In order to evaluate each insertion cost in constant time, a calculation of possible positions to insert visits is first performed. Then information for each visit is archived and updated during the process as follows. For each visit i, we use $\textsc{Wait}_i$ to memorize the waiting time in case the arrival takes place before the beginning of the time window, $\textsc{MaxShift}_i$ to compute the maximal delay of the visit. Supposing that $\textsc{Arrival}_i$ and $\textsc{Start}_i$ are the arrival time and the starting time of the service respectively, it holds that

$$\textsc{Wait}_i = \textsc{Start}_i - \textsc{Arrival}_i \tag{11}$$

Because of the synchronization constraints, $\textsc{Start}_i$ for some visits may be delayed so that the client is simultaneously served by the assigned vehicles. For a given route r, we also use function $r(p)$ to denote the visit at position p in the route. We now notice that $\textsc{MaxShift}_{r(p)}$ is equal to the sum of $\textsc{Wait}_{r(p+1)}$ and $\textsc{MaxShift}_{r(p+1)}$, unless there is a time window bound.

$$\textsc{MaxShift}_{r(p)} := \min\{b_{r(p)} - \textsc{Start}_{r(p)}, \textsc{Wait}_{r(p+1)} + \textsc{MaxShift}_{r(p+1)}\} \tag{12}$$

Besides, if visits need to be synchronized, the minimal value of MaxShift is taken for all of them. Therefore, if there exists $j \in V$ such that $[r(p), j] \in P^{sync}$ we have:

$$\textsc{MaxShift}_{r(p)} := \min\{\textsc{MaxShift}_{r(p)}, \min_{j \in P^{sync}_{r(p)}} \textsc{MaxShift}_j\} \tag{13}$$

Therefore, an insertion of a visit k in a route r between p and $p+1$ will be considered to be valid if the generated shift $\textsc{Shift}_k^{r,p}$ is less than or equal to the sum of $\textsc{Wait}_{r(p+1)} + \textsc{MaxShift}_{r(p+1)}$.

$$\textsc{Shift}_k^{r,p} := \delta_{r(p)k} + \textsc{Wait}_k + S_k + \delta_{kr(p+1)} - \delta_{r(p)r(p+1)} \tag{14}$$

As mentioned earlier, the insertion with the least cost is applied in each iteration. The insertion cost is considered to be $\delta_{r(p)k} + \delta_{kr(p+1)} - \delta_{r(p)r(p+1)}$ for the case of minimizing the travel cost, P_{kr} when dealing with the preference, and the new $\mathcal{W}$ calculated using (10c) if optimizing the workload balance.

After the insertion, the update is propagated through different routes because of the synchronization constraints. The propagation may loop infinitely if the cross synchronizations are not prohibited, e.g. visiting u then v by the first vehicle, visiting i then j by the second vehicle, and finally realizing that u and j are the same client as well as v and i, e.g. $[u, j], [v, i] \in P^{sync}$. To avoid such issues, transitive closures [2] are computed to filter out cross synchronizations from the set of possible positions for insertion. We use a reduced solution $\mathcal{R}_{Sync}(X)$ to refer to a structure equivalent to solution X with only the synchronization visits (see Fig. 1). This computation is applied on $\mathcal{R}_{Sync}(X)$ and takes $O(s^3)$ steps where s is the number of synchronizations. Therefore, the complexity of constructing a solution completely from scratch (worst-case) is $O(n \cdot \max\{s^3, n^2\})$.

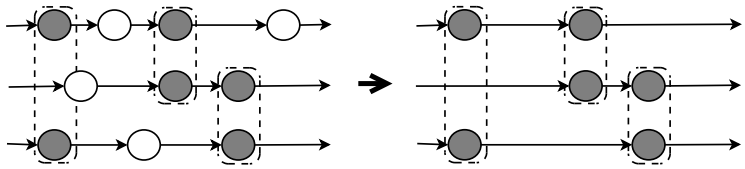

Fig. 1: Example of the $\mathcal{R}_{Sync}(X)$ solution representation

2.2 Diversification process

The function $Diversification(X, d_{min}, d_{max})$ first removes a number (randomly generated between d_{min} and d_{max}) of visits from the current solution and runs a local search procedure (described in Subsection 2.3) to improve this partial solution. A reconstruction phase is then processed using the above constructive heuristic. This iterative approach is similar to the *destruction/repair* operator used in [5, 10]. At every iteration of the algorithm, the diversification process is applied using d as the maximum number of visits to be removed. Except that after n iterations without improvement a greater diversification process is applied. This involves destroying a large part of the solution.

In addition, a dynamic priority management is also administered to identify critical visits. Each visit is associated with a priority number initialized to 0. This number is increased by 1 unit whenever the insertion of the visit cannot be done. Visits having the highest priority, i.e. frequently caused infeasible solutions, are in fact critical. Therefore, they need to be inserted during the early stages of the constructive heuristic. With this dynamic management, the search is guided back to the feasible space whenever it hits the infeasible one. In general, we observed that the ration between infeasible and feasible solutions explored by our algorithm varies from one instance to another. This generally depends on the size of the time windows, e.g. the algorithm finds infeasible solutions more frequently with instances having small time windows.

2.3 Local search procedure

The following neighborhoods were adapted to the synchronization constraints and used in our local search procedure:

2-opt*, exchanges the tails of two routes:*

In a 2-opt operator, the possibilities of exchanging two links with two others in the same route are explored to find a local improvement. For the case of multiple vehicles, we use 2-opt* [19] to denote the same principle of exchange but related to two distinct routes. This operator consequently implies the exchanges of paths between the two routes. It is particularly suitable for our case since it is hardly possible for the classical 2-opt to find an improvement due to the preserved order of visits from the time windows. Our 2-opt* is implemented as follows.

A subset of visits of size d is randomly selected and for each couple of visits $\{r(p), r'(q)\}$, we consider the arcs $(r(p), r(p+1))$ and $(r'(q), r'(q+1))$. If the exchange of these two arcs for $(r(p), r'(q+1))$ and $(r'(q), r(p+1))$ ensures the

feasibility then the associated cost is recorded. The feasibility check is handled by the same process as the one used in the constructive heuristic to avoid cross synchronizations. Therefore, the exchange cost is evaluated in a constant time for each couple $\{r(p), r'(q)\}$. After testing all the possible couples, the best one is then memorized and the improving exchange is applied.

***or-opt**, relocation of visits in the same route:*

In this operator [22], we look for the possibilities of relocating a sequence of (1, 2 or 3) visits from its original place to another one in the same route. The implementation of this operator is similar to 2-opt* operator: a random selection at the beginning then a feasibility check and the best move is applied. Although this operator does not directly improve the objective when minimizing the sum of preferences, it compacts the routes and makes room for further insertions.

***replacement**, exchanges between the routed and unrouted visits:*

In this operator, we try to insert unrouted visits by mean of exchanging them with routed visits. The operator is implemented with a full enumeration. That is to say for each routed visit we try to exchange its position with all the unrouted visits. Among the feasible exchanges that improve the objective, the best one is applied.

***single-move**, change the position of the routed visits:*

This operator tries to move every routed visit from its current position to another position so that the objective value is improved. Similar to *replacement*, a full enumeration is considered and the best improving move is applied. Unlike or-opt, the operator looks for potential positions in all routes and only one routed visit is considered at a time.

At each iteration, a random neighborhood w is chosen from the set W of unexplored neighborhoods, initialized to {2-opt*, or-opt, replacement, single-move} denoted W_0. Neighborhood w is then removed from W and applied on the current solution. If at least one improvement is detected by the current neighborhood w, the set of unexplored neighborhoods will be set back to W_0. The procedure is terminated when W is empty.

3 Experimentation

We tested our algorithm on the standard instances of [7]. The benchmark which was generated to simulate the scheduling problem in homecare services, comprises 10 data sets. The sets are grouped in 3 categories based on the number of clients. Each set has 5 varieties of instances, those are named after the width of the time windows (Small, Medium and Large). In each instance, about 10% of the visits need to be pairwisely synchronized.

Our algorithm is coded in C++ using the Standard Template Library (STL) for data structures. The program is compiled with GNU GCC in a Linux environment and all experiments were conducted on an Intel Xeon 2.67 GHz. Our configuration

is similar to the computational environment used by Bredström and Rönnqvist [6, 7]. According to the protocol proposed in [6], all the methods were tested with the three varieties as mentioned earlier. We consider the three objectives separately: minimizing the total travel time, minimizing the sum of negative preferences and minimizing the maximal difference in service times of the vehicles.

The three parameters required to be tuned in our algorithm are: the initial temperature T_0, the control parameter α of the cooling schedule and d: the maximum number of visits to be removed in the diversification phase.

The value of T_0 is set during execution so that the probability of accepting degrading solutions at the beginning is 0.95 (see [23]). First, 20 random neighbors solutions of the initial solution are generated to measure the largest gap Δ_f between their qualities. Then, T_0 is set to $-\Delta_f / \ln(0.95)$.

Concerning the two remaining parameters α and d, we explore to subset of possible settings on a subset of training instances as follows. The control parameter α was tested with values 0.9, 0.95, 0.99, 0.995, 0.999. For parameter d, we used values in $[1, 19]$. This results 95 different combinations of the pair $\{\alpha, d\}$ for the test. The training set is picked from the instances of the benchmark with more than 50 clients. For each combination of the settings, the algorithm was executed 10 times per instance. Two following quantitative measures are used to compare the combinations: the average relative gap to the best solutions found measured in each objective, denoted by *rpe* and the average computation time, denoted by *cpu*. In order to make the comparison, those measures are normalized into $[0, 1]$ interval as shown in Fig. 2. The best combination is then selected among the *non-dominated* configuration points so that the euclidean distance to the ideal point $(0, 0)$ is minimized. Using this technique, we adopt the following parameter settings: $\{\alpha = 0.99, d = 10\}$ (the highlighted point in the figure).

Comparative results

With the parameters found in the previous sections, our algorithm is then tested on the whole benchmark. Tables 1, 2 and 3 report our results and compare them with the existing methods in the literature. Column *Best* shows the best known solution collected from all methods (including ours) for each instance. A star mark (*) is used in *Best* to indicate that the solution has been proved to be optimal by an exact method. The other columns are: MIP for the results of the linear model solver reported in [7]; H for the heuristic proposed in [7] which was based on the local-branching technique [14]; BP for the results of the branch-and-price algorithms presented in [6]. They tested two variants of their algorithm when treating the travel time, we denote them by BP1 and BP2. Finally the column SA-ILS for our simulated annealing algorithm. Columns Sol and CPU report the best solution found by each method and the computational time. Bold numbers in Sol indicate that the solution quality reaches *Best*. The time unit in those tables for the objective values like travel time or fairness is in hours, and for the computational time is in seconds. Note also that, in order to establish an accurate comparison, the results of H and MIP are reported based on our reimplementation of [7] using the more recent version of CPLEX 12.6, and tested on the same environment of computation as the one used by our SA-ILS. Compared to the originally published results of [7], few improvements can be noticed.

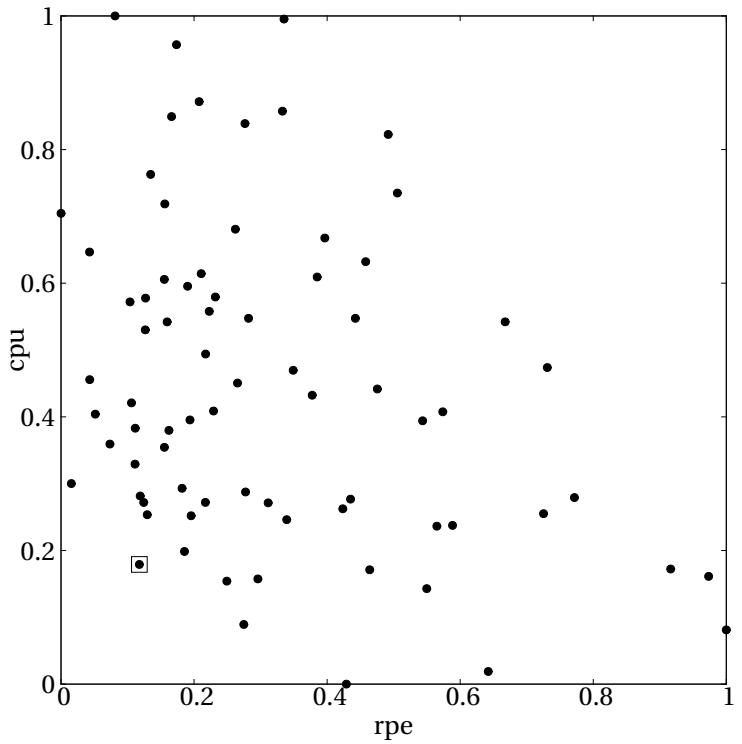

Fig. 2: Tradeoff between performance and computational time for different parameter settings

Table 1: Comparison of the solutions and computational times for the total travel time

Data	*Best*	MIP		H		BP1		BP2		SA-ILS	
		Sol	CPU	Sol	CPU	Sol	CPU	Sol	CPU	Sol	CPU
1S	3.55*	**3.55**	3.43	**3.55**	120.03	**3.55**	1.96	**3.55**	1.12	**3.55**	0.02
1M	3.55*	**3.55**	14.48	**3.55**	120.25	**3.55**	221.93	**3.55**	3.69	**3.55**	0.02
1L	3.39*	**3.39**	76.71	**3.39**	120.48	**3.39**	107.41	**3.39**	11.91	**3.39**	0.03
2S	4.27*	**4.27**	0.22	**4.27**	120.07	**4.27**	3.28	**4.27**	0.56	**4.27**	0.02
2M	3.58*	**3.58**	25.97	**3.58**	120.11	**3.58**	8.12	**3.58**	3.2	**3.58**	0.03
2L	3.42*	**3.42**	183.11	**3.42**	120.95	**3.42**	2.72	**3.42**	7.41	**3.42**	0.03
3S	3.63*	**3.63**	1.79	**3.63**	120.26	**3.63**	14.17	**3.63**	3.84	**3.63**	0.02
3M	3.33*	**3.33**	21.24	**3.33**	120.19	**3.33**	17.57	**3.33**	4.31	**3.33**	0.03
3L	3.29*	**3.29**	96.47	**3.29**	120.6	**3.29**	42.78	**3.29**	1.44	**3.29**	0.02
4S	6.14*	**6.14**	30.9	**6.14**	120.16	**6.14**	14.02	**6.14**	1.54	**6.14**	0.02
4M	5.67*	**5.67**	1380	**5.67**	120.15	**5.67**	27.53	**5.67**	2.55	**5.67**	0.05
4L	5.13*	**5.13**	3600	5.3	120.04	**5.13**	9.74	**5.13**	7.69	**5.13**	0.09
5S	3.93*	**3.93**	6.99	**3.93**	120.13	**3.93**	2.84	**3.93**	2.9	**3.93**	0.03
5M	3.53*	**3.53**	6.2	**3.53**	120.09	**3.53**	57.04	**3.53**	9.1	**3.53**	0.03
5L	3.34*	**3.34**	225.23	**3.34**	120.85	**3.34**	9.11	**3.34**	5.15	**3.34**	0.03
6S	8.14*	**8.14**	3600	**8.14**	600.94	**8.14**	3600	**8.14**	197	**8.14**	13.97
6M	7.7	8.14	3600	11.63	609.58	7.71	3600	**7.7**	3600	**7.7**	26.68
6L	7.14*	-	3600	-	624.06	**7.14**	3279	**7.14**	3600	**7.14**	15.86
7S	8.39*	-	3600	8.97	603.97	**8.39**	14.72	**8.39**	169	**8.39**	15.08
7M	7.48	12.66	3600	-	648.02	7.67	3600	7.56	3600	**7.48**	18.34
7L	6.88	12.66	3600	-	645.33	**6.88**	3600	**6.88**	3600	**6.88**	15.92
8S	9.54*	-	3600	-	657.03	**9.54**	931	**9.54**	850	**9.54**	25.13
8M	8.54*	-	3600	-	632.61	**8.54**	3600	**8.54**	3490	**8.54**	15.01
8L	8	-	3600	-	618.63	8.62	3600	8.11	3600	**8**	24.51
9S	11.93	-	3600	-	626.26	-	3600	12.21	3600	**11.93**	150.52
9M	10.92	-	3600	-	612.19	11.74	3600	11.04	3600	**10.92**	292.17
9L	10.49	-	3600	-	607.36	11.11	3600	10.89	3600	**10.49**	207.17
10S	8.60	-	3600	-	604.46	-	3600	9.13	3600	**8.60**	16.10
10M	7.62	-	3600	-	705.2	8.54	3600	8.1	3600	**7.62**	52.75
10L	7.75	-	3600	-	631.39	-	3600	-	3600	**7.75**	51.89

Table 2: Comparison of the solutions and computational times for the sum of negative preferences

Data	*Best*	MIP		H		BP		SA-ILS	
		Sol	CPU	Sol	CPU	Sol	CPU	Sol	CPU
1S	-114.03*	**-114.03**	1.05	**-114.03**	3600	**-114.03**	1.27	**-114.03**	0.03
1M	-117.8*	**-117.8**	1.04	**-117.8**	3600	**-117.8**	1.68	**-117.8**	0.02
1L	-118.51*	**-118.51**	0.52	**-117.8**	3600	**-118.51**	2.55	**-118.51**	0.04
2S	-92.09*	**-92.09**	0.58	**-92.09**	3600	**-92.09**	0.6	**-92.09**	0.05
2M	-104.81*	**-104.81**	32.94	**-104.81**	3600	**-104.81**	2.3	**-104.81**	0.04
2L	-107.64*	**-107.64**	427.74	**-107.64**	3600	**-107.64**	6.44	**-107.64**	0.38
3S	-99.49*	**-99.49**	0.95	**-99.49**	3600	**-99.49**	1.66	**-99.49**	0.02
3M	-106.59*	**-106.59**	2.79	**-106.59**	3600	**-106.59**	2.01	**-106.59**	0.07
3L	-107.87*	**-107.87**	1.95	**-107.87**	3600	**-107.87**	2.63	**-107.87**	0.14
4S	-100*	**-100**	2.22	**-100**	3600	**-100**	1.72	**-100**	0.03
4M	-106.72*	**-106.72**	68.26	**-106.72**	3600	**-106.72**	2.36	**-106.72**	0.07
4L	-109.27*	**-109.27**	170.25	**-109.27**	3600	**-109.27**	5.04	**-109.27**	0.13
5S	-76.29*	**-76.29**	0.26	**-76.29**	3600	**-76.29**	0.64	**-76.29**	0.02
5M	-76.29*	**-76.29**	1.08	**-76.29**	3600	**-76.29**	1.28	**-76.29**	0.03
5L	-84.21*	**-84.21**	16.33	**-84.21**	3600	**-84.21**	2.21	**-84.21**	0.04
6S	-370.06*	**-370.06**	1452.76	**-370.06**	3600	**-370.06**	150.63	**-370.06**	0.7
6M	-379.88*	-372.4	3600	-374.257	3600	**-379.88**	247.88	**-379.88**	25.26
6L	-387.2*	-	3600	-368.876	3600	**-387.2**	474.15	**-387.2**	16.33
7S	-401.11*	-	3600	-296.725	3600	**-401.11**	291.29	**-401.11**	0.58
7M	-406.17*	-	3600	-368.565	3600	**-406.17**	86.7	**-406.17**	6.48
7L	-407.48*	-	3600	-355.716	3600	**-407.48**	710.62	**-407.48**	2.53
8S	-380.76*	-	3600	-	3600	**-380.76**	135.39	**-380.76**	26.12
8M	-403.57*	-	3600	-	3600	**-403.57**	290.77	**-403.57**	59.34
8L	-407.48	-	3600	-	3600	**-407.48**	362.18	**-407.48**	20.51
9S	-581.12	-	3600	-	3600	-552.65	3600	**-581.12**	117.4
9M	-656.5	-	3600	-	3600	-463.82	3600	**-656.5**	10.90
9L	-666.5	-	3600	-	3600	-663.47	3600	**-666.5**	17.81
10S	-675.81	-	3600	-	3600	**-675.81**	3600	**-675.81**	162.42
10M	-686.75	-	3600	-	3600	-685.31	3600	**-686.75**	150.63
10L	-691.48	-	3600	-445.027	3600	-691.34	3600	**-691.48**	270.26

Table 3: Comparison of the solutions and computational times for the fairness objective

Data	*Best*	MIP		H		SA-ILS	
		Sol	CPU	Sol	CPU	Sol	CPU
1S	0*	**0**	444.1	0.03	3600	**0**	0.30
1M	0*	**0**	0.14	**0**	3600	**0**	0.45
1L	0*	**0**	0.16	**0**	3600	**0**	0.61
2S	0.01	0.04	3600	**0.01**	3600	**0.01**	0.30
2M	0.01	0.04	3600	**0.01**	3600	**0.01**	0.60
2L	0.01	-	3600	**0.01**	3600	**0.01**	0.81
3S	0.01	0.06	3600	**0.01**	3600	**0.01**	0.49
3M	0.01	0.06	3600	**0.01**	3600	**0.01**	0.65
3L	0.01	0.06	3600	**0.01**	3600	**0.01**	0.70
4S	0.07	0.13	3600	**0.07**	3600	**0.07**	0.61
4M	0.02	0.08	3600	0.03	3600	**0.02**	0.73
4L	0.02	0.08	3600	**0.02**	3600	**0.02**	0.46
5S	0.01	0.08	3600	**0.01**	3600	**0.01**	0.39
5M	0.01	0.08	3600	**0.01**	3600	**0.01**	0.69
5L	0.01	0.06	3600	0.03	3600	**0.01**	1.13
6S	0.11	-	3600	0.7	3600	**0.11**	2.35
6M	0.07	-	3600	0.56	3600	**0.07**	18.32
6L	0.12	-	3600	1.75	3600	**0.12**	0.57
7S	0.18	-	3600	1.59	3600	**0.18**	1.04
7M	0.15	-	3600	-	3600	**0.15**	0.97
7L	0.14	0.77	3600	-	3600	**0.14**	0.84
8S	0.36	-	3600	-	3600	**0.36**	1.36
8M	0.33	-	3600	-	3600	**0.33**	0.84
8L	0.32	-	3600	-	3600	**0.32**	1.19
9S	0.45	-	3600	-	3600	**0.45**	4.09
9M	0.23	-	3600	-	3600	**0.23**	3.43
9L	0.46	-	3600	-	3600	**0.46**	3.35
10S	0.22	-	3600	-	3600	**0.22**	3.57
10M	0.28	-	3600	-	3600	**0.28**	5.20
10L	0.25	-	3600	-	3600	**0.25**	6.22

From these results, we remark that SA-ILS finds all known optimal solutions (20 of 30 when minimizing the total travel time and 24 of 30 when minimizing the sum of preference) in very short computational times compared to the other methods. The solution quality for the remaining instances is also better than the one found in the literature.

For the total travel time, the algorithm strictly improved the best known solutions for 8 instances of the data sets. Those instances are $7M$, $8L$, $9S$, $9M$, $9L$, $10S$, $10M$ and $10L$. For the sum of negative preferences, we could improve 5 instances. Those instances are $9S$, $9M$, $9L$, $10M$ and $10L$.

For the fairness objective, only the instances with 20 clients had been reported with results in the literature. For example, the objective is completely ignored in [6]. However, since our algorithm can produce feasible solutions for all the objectives, Table 3 shows our complete results on this objective compared with the ones from the literature. Unreported results are marked with a dash. In general, we obtained solutions with better quality and within comparable computational times. To summarize, our SA-ILS is clearly fast and efficient in optimizing the three studied objectives.

4 Conclusion

In this paper, we proposed a new approach to address VRPTWSyn. The approach is based on a Simulated Annealing algorithm and featured a Local Search procedure. To the best of our knowledge, this is the first time that dedicated local search heuristics have been proposed and evaluated on this variant of VRP. The experiments conducted on the standard benchmark [7] for VRPTWSyn clearly demonstrate the efficiency and the competitiveness of our approach compared to the existing methods in the literature. The results also confirm that destruction/repair operator and local search heuristics can be efficiently adapted to support the synchronization constraints.

As a future work, we plan to develop efficient exact methods to solve VRPTWSyn since there is still unsolved instances in the benchmark. In particular, having an efficient algorithm being able to produce high quality feasible solutions in short computational times, such as our SA-ILS, is a significant advantage. For example, those solutions can be used as warm start for the exact methods. Another interesting direction could be considering the three objectives at the same time, e.g. addressing VRPTWSyn as a truly multi-objective problem, and developing a population-based algorithm from SA-ILS, as similar to [3].

Acknowledgements This work was partially supported by the Regional Council of Picardy and the European Regional Development Fund (ERDF), under PRIMA project. It was also partially supported by the National Agency for Research, under ATHENA project, reference ANR-13-BS02-0006-01. This work was carried out in the framework of the Labex MS2T, which was funded by the French Government, through the program "Investments for the future" managed by the National Agency for Research (Reference ANR-11-IDEX-0004-02).

References

1. Afifi S, Dang DC, Moukrim A (2013) A simulated annealing algorithm for the vehicle routing problem with time windows and synchronization constraints. In: Proc. of LION-7, Lecture Notes in Computer Science, vol 7997, pp 259–265
2. Aho AV, Garey MR, Ullman JD (1972) The transitive reduction of a directed graph. SIAM Journal on Computing 1(2):131–137
3. Baños R, Ortega J, Gil C, Márquez AL, De Toro F (2013) A hybrid meta-heuristic for multi-objective vehicle routing problems with time windows. Computers & Industrial Engineering 65(2):286–296
4. Baños R, Ortega J, Gil C, Fernandez A, De Toro F (2013) A simulated annealing-based parallel multi-objective approach to vehicle routing problems with time windows. Expert Systems with Applications 40(5):1696–1707
5. Bouly H, Dang DC, Moukrim A (2009) A memetic algorithm for the team orienteering problem. 4OR 8(1):49–70
6. Bredström D, Rönnqvist M (2007) A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints. NHH Dept of Finance and Management Science Discussion Paper
7. Bredström D, Rönnqvist M (2008) Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. European Journal of Operational Research 191(1):19–31

8. Chiang WC, Russell RA (1996) Simulated annealing metaheuristics for the vehicle routing problem with time windows. Annals of Operations Research 63(1):3–27
9. Czech Z, Czarnas P (2002) Parallel simulated annealing for the vehicle routing problem with time windows. Proc of 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing
10. Dang DC, Guibadj RN, Moukrim A (2013) An effective PSO-inspired algorithm for the team orienteering problem. European Journal of Operational Research 229(2):332–344
11. Dohn A, Rasmussen MS, Larsen J (2011) The Vehicle Routing Problem with Time Windows and Temporal Dependencies. Networks 58(4):273–289
12. Drexl M (2012) Synchronization in vehicle routing a survey of VRPs with multiple synchronization constraints. Transportation Science 46(3):297–316
13. El Hachemi N, Gendreau M, Rousseau LM (2013) A heuristic to solve the synchronized log-truck scheduling problem. Computers & Operations Research 40(3):666–673
14. Fischetti M, Lodi A (2003) Local branching. Mathematical Programming 98(1-3):23–47
15. Ioachim I, Desrosiers J, Soumis F, Bélanger N (1999) Fleet assignment and routing with schedule synchronization constraints. European Journal of Operational Research 119(1):75 – 90
16. Kirkpatrick S, Vecchi M, et al (1983) Optimization by simmulated annealing. science 220(4598):671–680
17. Lenstra JK, Kan A (1981) Complexity of vehicle routing and scheduling problems. Networks 11(2):221–227
18. Li Y, Lim A, Rodrigues B (2005) Manpower allocation with time windows and job-teaming constraints. Naval Research Logistics (NRL) 52(4):302–311
19. Potvin JY, Kervahut T, Garcia BL, Rousseau JM (1996) The vehicle routing problem with time windows part I: tabu search. INFORMS Journal on Computing 8(2):158–164
20. Rasmussen MS, Justesen T, Dohn A, Larsen J (2012) The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. European Journal of Operational Research 219(3):598–610
21. Solomon MM (1987) Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research 35(2):254–265
22. Solomon MM, Desrosiers J (1988) Time window constrained routing and scheduling problems. Transportation science 22:1–13
23. Tavakkoli-Moghaddam R, Gazanfari M, Alinaghian M, Salamatbakhsh A, Norouzi N (2011) A new mathematical model for a competitive vehicle routing problem with time windows solved by simulated annealing. Journal of Manufacturing Systems 30(2):83–92
24. Toth P, Vigo D (2002) An overview of vehicle routing problems. The vehicle routing problem 9:1–26
25. Van Breedam A (1995) Improvement heuristics for the vehicle routing problem based on simulated annealing. European Journal of Operational Research 86(3):480–490
26. Wen M, Larsen J, Clausen J, Cordeau JF, Laporte G (2009) Vehicle routing with cross-docking. Journal of the Operational Research Society 60(12):1708–1718