Proceedings of the 2013 Federated Conference on
Computer Science and Information Systems pp. 329–336

A Beam Search Based Algorithm for the Capacitated Vehicle Routing Problem with Time Windows

Hakim Akeb
ISC Paris Business School
22 Bd du Fort de Vaux
75017 Paris, France
Email: hakeb@iscparis.com

Adel Bouchakhchoukha[(1)(2)]
[(1)]MSE, Université Paris 1 Panthéon Sorbonne
106–112 Bd de l'Hôpital
75013 Paris, France
Email: adel.bouchakhchoukha@
malix.univ-paris1.fr

Mhand Hifi[(1)(2)]
[(2)]Université de Picardie Jules Verne
UR EPROAD, Équipe ROAD
7 rue du Moulin Neuf
80039 Amiens, France
Email: mhand.hifi@u-picardie.fr

***Abstract*—In this paper the capacitated vehicle routing problem with time windows is tackled with a beam-search based approximate algorithm. An instance of this problem is defined by a set of customers and a fleet of identical vehicles. A time window is associated with each customer and a maximum capacity characterizes a vehicle. The aim is then to serve all the customers by minimizing the number of vehicles used as well as the total distance and by respecting the time windows.**

The proposed method follows three complementary phases: (i) dividing the set of customers into disjunctive clusters, (ii) determining a feasible solution in each cluster by using beam search, and (iii) applying a local search in order to improve the quality of the solutions. The proposed method is analyzed computationally on a set of benchmarks due to Solomon. Encouraging results have been obtained.

I. Introduction

THE *Vehicle Routing Problem* (VRP) is well known and well studied in the literature. It consists, in its simplest version, to visit or deliver a set $N = \{1, ..., n\}$ of customers by using a fleet of m vehicles $V = \{v_1, ..., v_m\}$. The objective is often to minimize the number of vehicles m as well as the sum of distances traveled by these ones. Note that if $m = 1$ then the problem becomes the *Traveling Salesman Problem* (TSP). Is some cases, a time window $W_i = [e_i, l_i]$ is associated to customer i, where e_i is the earliest time to begin the service of this customer and l_i the latest time. Parameters e_i and l_i are also known as *ready time* and *due date* respectively. The aforementioned problem is known as the *Vehicle Routing Problem with Time Windows* (VRPTW). A service time s_i can be associated with customer i, this means that the arrival time at this customer must be at most $l_i - s_i$. Furthermore, each customer $i, i \in N$ has a demand d_i and a capacity may be associated with each vehicle denoting the maximum of the sum of quantities that can be put inside the corresponding vehicle. Such a problem is called *Capacitated VRPTW* (CVRPTW).

VRP was addressed by many authors and several methods and strategies were proposed to solve its different versions. These methods can be categorized into two categories: exact methods and approximate ones. In the first category, Azi et al. [3] proposed an exact algorithm, based on column generation and branch-and-price, to solve VRPTW including multiple use of vehicle, i.e., a given vehicle may be associated with several routes. The same authors [2] proposed, several years before, another exact algorithm for a single vehicle routing problem with time windows and multiple routes. Baldacci et al. [4] employed branch-and-price in order to solve a capacitated vehicle routing problem (CVRP) by using an integer programming formulation. New lower bounds were presented and an algorithm to find the optimal solution for CVRP was given. Baldacci and Maniezzo [5] proposed exact methods based on node-routing formulations to tackle the undirected arc-routing problems. Feillet et al. [11] developed an exact algorithm for the elementary shortest path problem with resource constraints where the authors indicated an application to some vehicle routing problems.

The second category of methods consists to search for approximate solutions by using essentially heuristics and metaheuristics. Solomon [19] proposed different algorithms in order to solve the vehicle routing and scheduling problems with time window constraints. A two-stage heuristic including ejection pools was for example proposed by Lim and Zhang [14] in order to tackle VRPTW. Chen et al. [9] proposed a heuristic that combines mixed integer programming and a record-to-record travel algorithm in order to solve approximately the *split delivery vehicle routing problem*, a variant of CVRP where a customer may be served by more than one vehicle. Tan et al. [22] proposed several heuristic methods, including simulated annealing, to solve VRPTW . Insertion heuristics were proposed by Campbell and Savelsbergh [7] for vehicle routing and scheduling problems. Chao et al. [8] proposed a fast heuristic for the orienteering problem, i.e., a vehicle routing problem where a profit is associated with each customer and the objective is to visit a subset of customers in order to maximize the total benefit and by respecting some constraints. Pisinger and Ropke [16] developed a general heuristic for vehicle routing problems able to solve five different variants of VRP, including CVRP and VRPTW. A genetic algorithm

978-1-4673-4471-5/$25.00

was proposed in [12], [21], and [17] for the VRPTW. Ant colony optimization is very effective and was adapted for the various variants of VRP (see for example [13] where the *open vehicle routing problem* was considered). Finally, tabu search was considered by Cordeau et al. [10] for solving VRPTW and by Brandão and Mercer [6] for solving the multi-trip vehicle routing and scheduling problem, i.e., the case where a vehicle may perform several trips.

In this work, we propose a three-phase algorithm for solving CVRPTW. The first phase consists to divide the set of customers into m clusters. After that, at phase 2, the shortest path that visits each customer once in each cluster is computed by using beam search. Each path must verify the problem constraints, i.e., must not violate the time window. In the third and last phase, a local search is applied on the solution in order to try to decrease the total distance traveled by the m vehicles.

II. Problem Statement and Mathematical Formulation

The problem to solve (CVRPTW) consists to visit n customers $i \in N = \{1, ..., n\}$, where each customer (or *vertex*) has coordinates (x_i, y_i) in the Euclidean plan. The float used contains m identical vehicles $V = \{v_1, ..., v_m\}$, each with the same maximal capacity $C_{\max}$. All the vehicles start their travel at the *depot* D, whose coordinates are (x_D, y_D) in the Euclidean plan, visit a set of distinct customers, and returns to the depot.

In addition, a demand d_i is associated with each customer $i \in N$, d_i has the same unit of measurement as the vehicle capacity, this may be a volume or a weight for example. Customer i must be served at time t_i that may be the earliest time after its ready time (e_i) and no later than its due date l_i, this corresponds to the time window $W_i = [e_i, l_i]$ associated with customer i ($l_i - e_i$ is called the *width* of the time window). A service time s_i is also defined for customer i and is equal to the time spend to serve the customer.

Each vehicle $v_j, 1 \leq j \leq m$ performs then a route R_j by visiting a number of customers or vertices. These ones correspond to a cluster denoted by $\mathcal{C}_j$. Let also denote by $\mathcal{D}_j$ the sum of distances covered by vehicle v_j. Note that the distance between to customers i and j, denoted by $dist_{ij}$ corresponds to the euclidean distance between these two points, i.e., $dist_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

CVRPTW can then be formulated as follows:

$$\min\ m \tag{1}$$

$$\min \sum \mathcal{D}_j,\ 1 \leq j \leq m \tag{2}$$

subject to

$$\sum_{i \in C_j} d_i \leq C_{\max},\ \ \forall i \in C_j \tag{3}$$

$$t_i \in W_i = [e_i, l_i]\ \ \forall i \in N \cup \{D\} \tag{4}$$

Equation 1 represents the first objective to minimize, that is the number of vehicles to use. Equation 2 is the second objective to minimize and corresponds to the total distance traveled by all vehicles. The first constraint is indicated in (3) and ensures that the sum of demands for each cluster C_j is at most equal to the vehicle capacity $C_{\max}$. The second constraint (4) means that each customers i must be served inside its time window $W_i = [e_i, l_i]$, i.e., $e_i \leq t_i \leq l_i,\ \forall i \in N \cup \{D\}$. Then the depot can be considered as a customer for which the demand is null $(d_D = 0)$ and is the only point that is visited twice. The time window $W_D = [e_D, l_D]$ associated to the depot defines the *scheduling horizon* and means that each vehicle cannot leave the depot before its opening (at time e_D) and must return to the depot before its closure (at time l_D).

III. A three-phase method for solving CVRPTW

It is well known that many methods for solving vehicle routing problems often contain three phases (steps):

P1. Consists to divide the set of customers into m disjoint clusters, i.e., $C_1, ..., C_m$ such that $C_i \cap C_j = \emptyset$ for $1 \leq i < j \leq m$.

P2. Apply a given method in order to compute the shortest path, or a path of maximum benefit inside each cluster.

P3. Try to improve the solution obtained after P2, by applying another method such as local search.

A. Clustering

The first phase in solving our problem (Capacitated Vehicle Routing Problems with Time Windows) consists to divide the n customers into m disjoint sets or clusters. Then each vehicle will visit all the customers in the clusters that is assigned to it. Fig. 1 shows an example where a set containing 16 points and a depot (D) is divided into three disjoint clusters $\{C_1, C_2, C_3\}$.

There exists several methods for clustering and many of them are based on the dispersion (distribution) of the point around a central point called *centroid*. In our case we choose the well-known *k-means* method in order to compute the clusters. Actually, this is an adaptation of k-means in order to compute m clusters each of total capacity smaller than of equal to the capacity $C_{\max}$ of the vehicle.

Algorithm 1 explain how procedure k-means works. It receives the set N of customers as input parameter. The procedure's output corresponds to m disjoint clusters respecting

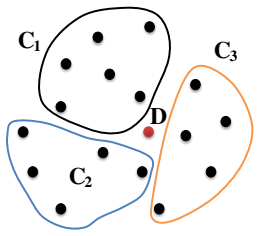

Fig. 1. An example of clustering a set of points into three disjoint clusters.

the vehicle capacity constraint. k-means begins by choosing m random points $c_1, ..., c_m$ from N (line 4), each point $c_j, (1 \leq j \leq m)$ corresponds to a centroid (belonging to cluster C_j). After that (at line 5), each not-assigned yet point $i \in N$ is assigned to the nearest centroid c_j of coordinates (x_{c_j}, y_{c_j}) in the sense euclidean distance. This means that point i of coordinates (x_i, y_i) is assigned to cluster C_j that minimizes the euclidean distance $\sqrt{(x_i - x_{c_j})^2 + (y_i - y_{c_j})^2}$, for $1 \leq j \leq m$.

After that, in the **while** loop that begins at line 6, the coordinates of the m centroids are recomputed (line 7). This is done by assigning to each $c_j, (1 \leq j \leq m)$ the center of the points belonging to cluster j. More precisely: $x_{c_j} = \frac{1}{|C_j|} \sum x_{c_k}$ and $y_{c_j} = \frac{1}{|C_j|} \sum y_{c_k}$ for all points $c_k \in C_j$. Each point $i \in N$ is after that assigned to the nearest centroid (among the new computed centroids), this is done in line 8. At line 9, if no point has moved from a centroid to another one, then a stable clustering is obtained: variable $move$ is set to the value $false$ (line 10) in order to stop recomputing of the centroids. Otherwise, this means that at least one point has moved, then instructions in lines 7–10 are repeated until a stable configuration is obtained. After that, the procedure verifies that the total capacity of each cluster does not exceed the capacity $C_{\max}$ of the vehicles. If so, then variable $found$ is set to "true" (line 14) in order to stop the procedure and return m clusters respecting the capacity constraint. If at least one cluster violates the capacity constraint, then the procedure restarts with m other random points by using the first **while** loop (line 2).

Note that the optimal number of clusters (m) is not known in the general case. So a dichotomous search can for example be used in order to test several values and determine the best one. Of course, increasing the value of m increases the probability to find clusters respecting the capacity constraint. For well-studied benchmarks, e.g. those proposed by Solomon [19], the same best value of m was found by many authors, so this value can be fixed in advance in order to save computation time.

Algorithm 1 Procedure k-means for CVRPTW

Require: Set N containing n points (customers);
Ensure: m disjoint clusters each of total capacity $\leq C_{\max}$;
1: $found \leftarrow$ false;
2: **while** ($found =$ false) **do**
3: $move \leftarrow true$;
4: Choose randomly m distinct points $\{c_1, ..., c_m\}$ from N. Let these points be the m centroids (clusters);
5: Assign each point $i \in N$ to the nearest centroid c_j, $1 \leq j \leq n$;
6: **while** ($move$ = true) **do**
7: Recompute the coordinates (x_{c_j}, y_{c_j}) of each centroid, i.e., each c_j becomes the center of the points assigned to that centroid;
8: Assign each point $i \in N$ to the nearest centroid c_j, $1 \leq j \leq n$;
9: **if** no point has moved from a cluster to another one **then**
10: $move \leftarrow$ false;
11: **end if**
12: **end while**
13: **if** the capacity of each a cluster $\leq C_{\max}$ **then**
14: $found \leftarrow$ true;
15: **end if**
16: **end while**

B. Beam search for computing the shortest paths (routes)

The second phase takes place after the m clusters were generated by the k-means procedure (algorithm 1). The objective of the second phase is to compute the shortest path in each cluster. A path correspond to a route beginning at the depot D, visiting exactly once each point (customer) are returning after that to the depot. In addition, one vehicle is associated to each cluster. Fig. 2 shows an example of solution for the example indicated in figure 1.

It is to note that the time windows associated to each customer as well as the capacity of the vehicle make the problem hard to solve, harder than the traveling salesman problem (TSP) in which there are no time windows and no limit to the vehicle capacity.

Remember that the objective in CVRPTW is to minimize the sum of distances traveled by the m vehicles. In order to compute the shortest paths, we propose to use beam search on each cluster. Beam search is a tree search and is a modified version of the well known branch-and-bound method.

Beam search was used to solve different combinatorial problems, such as Scheduling [15] and Cutting–and–Packing problems [1]. In its width–first implementation, the method starts by creating the root node which may contains an initial (starting) partial solution. After that, each node at level ℓ generates a set of descendants, these correspond to level $\ell+1$. Each node of the new level is then evaluated by using an evaluation criterion and only a subset containing the ω best nodes are retained, the other nodes are discarded. Parameter ω is known as the *beam width*. If a node contains a final solution, then this one is evaluated and stored. The corresponding node is after that deleted because no branching is possible from it (leaf). The beam search stops when no branching becomes possible from any node of the current level. The best solution, among the different solutions obtained, is then retained as the final result.

1) Content of a node in the search tree: Let $V = \{v_1, ..., v_{|C_j|}\}$ be the set of vertices (customers) in cluster C_j $(1 \leq j \leq m)$.

It is important to define clearly the content of a node in the beam search tree. Each node η_ℓ at level ℓ contains the following elements:

- The set of vertices (customers) already visited $V^+ = \{v_1^+, ..., v_\ell^+\}$.
- The set of vertices that have not yet been visited $V^- = V \backslash V^+$.
- The distance $dist$ corresponding to the length of the path

$D \rightarrow v_1^+ \rightarrow v_2^+ \rightarrow ... \rightarrow v_\ell^+$.

Note then that if a node corresponds to a complete solution, then the path obtained is $D \rightarrow v_1^+ \rightarrow ... \rightarrow v_{|C_j|}^+ \rightarrow D$ and the total distance is the sum of euclidean distances of the corresponding arcs in the path.

As a result, a node η_ℓ at level ℓ in the search tree can be designated by the elements described above, i.e, $\eta_\ell = \{V^+, V^-, dist\}$, where $|V^+| = \ell$ and $|V^-| = |C_j| - \ell$.

2) Selection criterion for the next customer to visit: As explained above, branching from a node η_ℓ (or more exactly from the last node v_ℓ^+ in the path under construction) consists to choose the successors of the vertex v_ℓ^+ among the vertices in V^-. The next vertex $v_i \in V^-$ may be for example the closest one to v_ℓ^+ in the sense of euclidean distance or the time window interval $[e_i, l_i]$. For example, for the two sets of instances examined in this work (see Section IV below), the next vertex to visit is the closest one in the sense of parameter e_i (the earliest time) in the time window. For beam search, all the successors v_i^- are ranked in increasing value of parameter e_i and then the ω first ones are chosen to create ω distinct branches.

Of course, others criteria were tested, including the latest time l_i and/or the distance between the current customer and the remaining customers to visit, but the experimentations showed that the criterion based on parameter e_i is the best one for the instances tested.

3) Algorithm Beam Search: Algorithm 2 explains how beam search works in order to compute a route. Note that the capacity constraint is not taken into account in the algorithm since the sum of the capacities of the vertices in each cluster is less or equal to the vehicle capacity. The capacity constraint is always respected after the clustering phase (see Section III-A), then only the time window constraint is checked.

Algorithm 2 receives three input parameters: the cluster C_j, i.e., the set of vertices or customers to visit (to serve), the value of the beam width ω, and the selection criterion ρ that will serve to sort the nodes at each level of the tree and then to determine the *best* ones according to this criterion. As output, the algorithm computes the best route (path) R_j of minimal distance beginning at the depot D, visiting each customer once, and then returning to the depot.

The root node η_0 of the search tree is created in line 1. This node contains the set of vertices already visited ,i.e. the depot D (so $V^+ = \{D\}$), and the set of vertices not already served $V^- = V$. Since no customer has been already visited, then the distance is equal to 0.

Set B (line 2) corresponds to the nodes at the current level of the search tree. Each node $\eta \in B$ contains a partial route (path) from the depot to a given customer (set V^+) as well as the set of remaining customers to visit V^-. The total distance to the current customer is also known since this distance is updated each time a new customer is visited and then added to the path (route). B is initialized to η_0 (line 4). Set B_{off} (line 3) contains the offspring nodes after branching from each node in B.

Algorithm 2 Beam Search for computing the shortest path in a cluster

Require: Cluster C_j, the beam width ω, and the selection criterion ρ;

Ensure: The best shortest route R_j starting from the depot D, visiting all the vertices of the cluster, and returning to the depot;

1: Let $\eta_0 \leftarrow \{\{D\}, V, 0\}$ be the root node;
2: Let B be the set containing the nodes at a given level of the tree;
3: Let B_{off} the offspring nodes (descendants of nodes in B);
4: $B \leftarrow \{\eta_0\}$;
5: $\ell \leftarrow 0$;
6: $\eta^* \leftarrow \eta_0$; (the best solution found)
7: $\eta^*.dist \leftarrow +\infty$; (best distance)
8: **while** $(B \neq \emptyset)$ **do**
9: Branch out of each node $\eta_{\ell_i} = \{V_i^+, V_i^-, dist_i\} \in B$ and create the offspring nodes B_{off} (each node in B_{off} must respect the time windows);
10: $\ell \leftarrow \ell + 1$;
11: **if** $(V_i^- = \emptyset$ for a node $\eta_{\ell_i} \in B_{\text{off}})$ **then**
12: Add vertex D (depot) to that node and compute the total distance;
13: **if** $(\eta_{\ell_i}.dist < \eta^*.dist)$ **then**
14: $\eta^* \leftarrow \eta_{\ell_i}$;
15: Remove η_{ℓ_i} from B_{off};
16: **end if**
17: **end if**
18: Sort the nodes in B_{off} according to parameter ρ and then keep only the $\min(\omega, |B_{\text{off}}|)$ first nodes, remove the other nodes from B_{off};
19: $B \leftarrow B_{\text{off}}$;
20: $B_{\text{off}} \leftarrow \emptyset$;
21: **if** there is a node $\eta_{\ell_i} \in B$ for which V_i^- contains a vertex with a violated time window **then**
22: Remove η_{ℓ_i} from B;
23: **end if**
24: **end while**

Since the current level ℓ is 0 (root node), then this is indicated in line 5, while the best solution η^* is initialized to the root node η_0 at line 6. The best distance $\eta^*.dist$ is set equal to $+\infty$ (line 7) because this value is to be minimized.

At line 8 the **while** loop starts. So at a given level ℓ of the tree, B contains at most ω distinct partial paths (routes) computed in parallel from the depot (root node). Then branching from a node η_{ℓ_i} (line 9) consists to explore the successors of the last visited vertex (customer) and to create as many nodes as there are successors with nonviolated time windows. So each node in B may have several descendants. Each descendant is then inserted into the set of offspring node B_{off}, that corresponds to level $\ell + 1$. This is why the level is incremented at the next line (10).

After that, at line 11, if there is a node in B_{off} in which all the customers were served $(V^- = \emptyset)$, then the complete

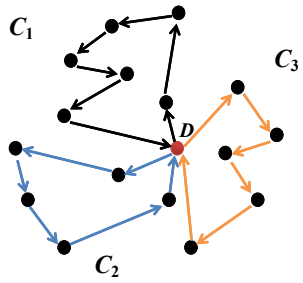

Fig. 2. An example of solution obtained after the second phase (beam search).

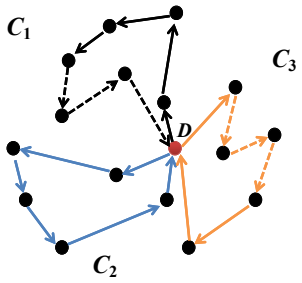

Fig. 3. A solution obtained after the third phase (local search).

solution is computed by adding the returning arc to the depot (line 12). The total distance for the obtained complete solution is then computed and compared to the best known one (line 13). If a lower distance is obtained then the best solution is updated (line 14) and the corresponding node is removed from B_{off} (line 15).

The most important instruction in beam search is certainly that given in line 18. Indeed, this step consists to sort the nodes according to the selection criterion ρ from the most important node to the least important one. Then the ω first nodes are kept and the other ones are removed from B_{off}. Note that if there are less than ω nodes in B_{off} then all the nodes are kept. After that set B_{off} is assigned to B and B_{off} reset to the empty set (lines 19–20). The last instruction in algorithm 2 consists to remove from B all the nodes that cannot lead to feasible solutions, i.e, that containing violated time windows.

The algorithm stops when set B becomes empty meaning that there is no node to explore or more precisely no customer to serve. Two cases can be distinguished: the algorithm has computed a feasible solution and this one is indicated in node η^* as well as the best corresponding distance, or there is no solution (if the distance in node η^* is equal to $+\infty$).

Fig. 2 shows an example of a solution that may be obtained after the second phase (beam search) on the example (clusters) shown in Fig. 1.

C. Local search for improving solution quality

In order to try to improve the result obtained after the second phase (beam search), a local search is performed on each cluster. This consists to execute the well-known 2-opt algorithm on each cluster (route).

2-opt is an iterative method that consists, at each iteration, to *break* two nonconsecutive arcs in the route and to link the four extremities in order to form another path and by respecting the time windows of course. The replacement is kept if the obtained solution is better.

The 2-opt method is given in algorithm 3. In each iteration, the algorithm examines each two distinct arcs $v_i \rightarrow v_{i+1}$ and $v_j \rightarrow v_{j+1}$ in the route R. These two arcs are replaced by the arcs $v_i \rightarrow v_j$ and $v_{i+1} \rightarrow v_{j+1}$ if and only if the distance decreases and the time windows are not violated. This process is repeated as long as there is improvement. Fig. 3 shows an example of improvement obtained by the 2-opt procedure on the solution of fig. 2. The arcs that had changed are indicated in dotted lines.

D. The three-phase algorithm (3PA) for solving CVRPTW

The three-phase algorithm based on clustering, beam search, and 2-opt local search is given in algorithm 4. It receives as input parameters the set of customers N, the depot D, and the number of vehicles. The algorithm's output corresponds to a set containing m feasible routes (respecting the constraints) of minimum distance, each one starts and ends at the depot D.

At line 1, the clustering phase (algorithm 1) is called in order to create m distinct clusters. The selection criterion (ρ), that serves to choose the next customer to serve is set at line 2. Then, algorithm 2 (beam search) is executed on each cluster (line 5), and this for several values of the beam width, i.e., for all values $\omega \in [1, ..., \omega_{\max}]$. The local search (algorithm 3) is then executed (line 6) on each solution computed by beam search.

Algorithm 3 2-opt algorithm

Require: A route $R = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_{|C_j|} \rightarrow v_{|C_j|+1}$;
Ensure: A route R' with a length at most equal to that of V;
1: $improvement \leftarrow$ true;
2: **while** ($improvement$ = true) **do**
3: $improvement \leftarrow$ false;
4: **for each** vertex $v_i \in R$ **do**
5: **for each** vertex $v_j \in R$ $(j \neq i-1, j \neq i+1)$ **do**
6: **if** $(\text{dist}(v_i, v_{i+1}) + \text{dist}(v_j, v_{j+1}) > \text{dist}\ (v_i, v_j) + \text{dist}(v_{i+1}, v_{j+1})$ **AND** the time windows will not be violated) **then**
7: Replace arcs $(v_i \rightarrow v_{i+1})$ and $(v_j \rightarrow v_{j+1})$ by arcs $(v_i \rightarrow v_j)$ and $(v_{i+1} \rightarrow v_{j+1})$;
8: $improvement \leftarrow$ true;
9: **end if**
10: **end for**
11: **end for**
12: **end while**

Algorithm 4 The three-phase algorithm 3PA for solving CVRPTW

Require: A set $N = \{1, ..., n\}$ of customers, the depot D, and m the number of vehicles (clusters).

Ensure: A set of routes minimizing the total distance and respecting the capacity and the time windows constraints.

1: Call the clustering phase (algorithm 1) and create m clusters $\{C_1, ..., C_m\}$ respecting the vehicle capacity constraint;
2: Define the selection criterion ρ;
3: **for each** cluster $C_j, (1 \leq j \leq m)$ **do**
4: **for** $\omega = 1$ to $\omega_{\max}$ **do**
5: Call algorithm 2: Beam-Search(C_j, ω, ρ);
6: Apply algorithm 3 (2-opt) on the solution returned by Beam-Search;
7: **end for**
8: **end for**

IV. Computational results

The proposed method is coded in C++ and the program run under Microsoft Windows environment on a computer with 2 GB of RAM and a 2.26 GHz Intel processor.

The algorithm was tested on two sets of instances, namely C1 and C2, proposed by Solomon. The characteristics of the two sets are summarized in table I. Each instance of each set contains 100 customers (column 2), they have also all the same service time (time needed to serve a customer) which is equal to 90 (column 4). The depot D has also the same coordinates for all the instances.

The first common characteristic between two distinct instances of the same set is the customer *demand*, i.e., the quantity to deliver to each customer. This value is fixed in each set for a given customer. More precisely, for two distinct instances in the same set (C1 or C2) each customer i has the same demand d_i. The second common characteristic is that a given customer i has the same coordinates (x_i, y_i) in two distinct instances of the same set (C1 or C2).

The third common characteristic appears in the scheduling horizon (column 3) of Table I, which is *short* for instances of set C1 (1236) and *large* for the instances of set C2 (3390). This means for example that the tours in instances C1 will all finish at most after 1236 units of time and at most after 3390 units of time for the instances of set C2. Finally, the fourth common characteristic concerns the vehicle capacity (column 5) of table I. In set C1, vehicles of capacity $C_{\max} = 200$ are used while this capacity is equal to 700 for instances of set C2.

TABLE I
Characteristics of the C1 and C2 instances

Set	Number of customers	Scheduling horizon	Service time	Vehicle Capacity
C1	100	1236	90	200
C2	100	3390	90	700

From these characteristics Solomon designed several instances in the same set by changing the time windows from an instance to another one. More precisely, there are nine instances C101–C109 in the first set C1 and eight instances C201–C208 in the second set C2. For two distinct instances in the same set (C1 or C2) we have:

- the same coordinates for a given customer i as well as for the depot D
- the same demand for a given customer i
- the same vehicle capacity
- different time windows

For more details, the reader can refer to Solomon's web site [20].

Then, one can surmise that less vehicles (clusters) will be needed for instances of set C2 comparing to set C1 because of the larger value of the capacity and the greater length of the time windows. This is in fact the case as proven in different works published in the literature.

Finally, note that the value of m (number of clusters or vehicles) is fixed to the best value found by several authors in the literature. More precisely, $m = 10$ for instances C1 and $m = 3$ for set C2.

Table II shows the results obtained on the 17 instances where column 1 indicates the name of each instance. Columns 2–4 contain the best known results in the literature (to our knowledge). Column 2 shows the best value for m (the number of vehicles) and column 3 the best distance. Column 4 (Ref.) indicates the reference to the paper where the best values for m and Dist were obtained. Columns 5–7 contain the results obtained by a method based on goal programming and genetic algorithm (GP-GA) proposed by Ghoseiri and Ghannadpour [12]. So columns 5 and 6 indicate the best value for m and the best distance respectively. Column 7 corresponds to the gap between the distance obtained by GP-GA (column 6) and the best known distance in the literature (column 3). This gap is computed as follows: $\text{gap} = 100\% \times (\text{Dist}_{\text{bestknown}} - \text{Dist}_{\text{GP-GA}})/\text{Dist}_{\text{bestknown}}$. Columns 8–12 summarize the results obtained by the proposed algorithm 3PA. Column 8 indicates the minimum number of vehicles m while column 9 shows the best minimum distance obtained by algorithm 3PA. Column 10 ($\omega_{\max}$) corresponds to the maximum value of the beam width used for each instance, then $\omega_{\max} = 50$ means that beam search was executed for each value $1 \leq \omega \leq 50$. The next column (time) indicates the total computation time needed for the execution of algorithm 3PA (in seconds). The last column (gap) indicates the difference (in %) between the solution obtained by the proposed method 3PA and the best known solution in the literature (column 3). More precisely $\text{gap} = 100\% \times (\text{Dist}_{\text{Best known}} - \text{Dist}_{\text{3PA}})/\text{Dist}_{\text{best known}}$. Finally, the last row of table II indicates the average gap for the two compared methods (GP-GA and 3PA). As expected, the number of vehicles needed for C1 instances (10) is larger than that needed for instances of set C2 (only 3 vehicles).

TABLE II
RESULTS OBTAINED ON INSTANCES C1 AND C2

	Best known			**GP-GA [12]**			**The proposed method (3PA)**				
Inst.	m	**Dist.**	**Ref.**	m	**Dist.**	**gap(%)**	m	**Dist.**	$\omega_{\max}$	**time (s)**	**gap (%)**
C101	10	**828.94**	[12]	10	**828.94**	0.0	10	**828.94**	50	29	0.00
C102	10	**828.94**	[12]	10	**828.94**	0.0	10	**828.94**	50	66	0.00
C103	10	**828.06**	[12]	10	**828.06**	0.0	10	828.94	50	140	-0.11
C104	10	**824.78**	[12]	10	**824.78**	0.0	10	828.94	50	183	-0.50
C105	10	**828.94**	[12]	10	**828.94**	0.0	10	**828.94**	50	36	0.00
C106	10	**828.94**	[12]	10	**828.94**	0.0	10	**828.94**	50	41	0.00
C107	10	**828.94**	[12]	10	**828.94**	0.0	10	**828.94**	50	41	0.00
C108	10	**828.94**	[12]	10	**828.94**	0.0	10	**828.94**	1600	146 600	0.00
C109	10	**828.94**	[12]	10	**828.94**	0.0	10	**828.94**	50	80	0.00
C201	3	**591.56**	[12]	3	**591.56**	0.0	3	**591.56**	50	176	0.00
C202	3	**591.56**	[12]	3	**591.56**	0.0	3	**591.56**	50	240	0.00
C203	3	**591.17**	[12]	3	**591.17**	0.0	3	**591.17**	100	13 210	0.00
C204	3	**590.60**	[17]	3	599.96	-1.58	3	591.17	100	12 390	-0.10
C205	3	**588.16**	[21]	3	588.88	-0.12	3	588.49	50	528	-0.06
C206	3	**588.49**	[17]	3	588.88	-0.07	3	**588.49**	2000	286 700	0.00
C207	3	**588.29**	[18]	3	591.56	-0.56	3	588.32	2000	291 600	-0.01
C208	3	**588.32**	[18]	3	**588.32**	0.0	3	588.88	50	618	-0.09
Average						-0.14					**-0.05**

The results of table II indicate that the proposed method 3PA reached the best known results in 11 cases out of 17. In the six other cases, the result is very close to the best known value since the gap is often smaller of equal to -0.11%, except for instance C104 where the gap reaches -0.50%. Note that even if the GP-GA method reaches the best known results in 13 cases out of 17, its average gap (-0.14%) is worst then that obtained by algorithm 3PA (-0.05%).

Concerning the computation time of algorithm 3PA, it is at most 183 seconds for instances of set C1 (except for C108 which was hard to solve). Each cluster in instances of set C1 contains about 10 customers. The computation time is generally greater for the second set C2, this is due to larger number of customers in each cluster (which is about 30) and then the number of combinations in each cluster (route) becomes larger.

But how to determine the maximum value for the beam width $\omega_{\max}$, especially for new instances. One can for example fix the maximum value to 50 or 100 or use a limited computation time.

Table III indicates, for each of the 17 solutions of table II the best value ω^* that gave the best solution for each cluster $C_j,\ j = 1,..,m$. We can see for example that $w^* = 1$ for all $j = 1,..,10$ for instance C101, but the values of ω^* are heterogeneous for instances C103 and C104 for example, meaning that these two instances are harder to solve (due to the characteristics of the time windows).

Fig. 4 shows an example of solution obtained after the second step of the algorithm (beam search), i.e., the output of algorithm 2 on instance C206. The total distance obtained after this step is equal to 597.35. Fig. 5 indicates improvement of the solution of fig. 4 (instance C206) by the 2-opt procedure (algorithm 3). The new distance was decreased from 597.35 to 588.49, i.e., an improvement of 1.48%. The 2-opt phase has changed several arcs in clusters 1 and 2 while the third cluster

TABLE III
BEST VALUE OF THE BEAM WIDTH IN EACH CLUSTER FOR INSTANCES C1 AND C2

	Cluster									
Inst.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
C101	1	1	1	1	1	1	1	1	1	1
C102	7	7	1	1	1	6	7	6	8	1
C103	39	13	1	8	7	10	44	5	8	4
C104	24	5	1	5	7	8	14	5	8	7
C105	1	1	1	1	1	1	1	1	1	1
C106	8	1	1	1	1	1	1	1	1	2
C107	1	1	1	1	1	1	1	1	1	1
C108	1518	2	3	2	2	10	1	1	3	1
C109	1	1	1	2	1	1	1	1	1	2
C201	1	1	1	–	–	–	–	–	–	–
C202	43	19	26	–	–	–	–	–	–	–
C203	43	35	68	–	–	–	–	–	–	–
C204	43	22	68	–	–	–	–	–	–	–
C205	1	1	1	–	–	–	–	–	–	–
C206	6	1687	1	–	–	–	–	–	–	–
C207	43	1928	1	–	–	–	–	–	–	–
C208	1	1	1	–	–	–	–	–	–	–

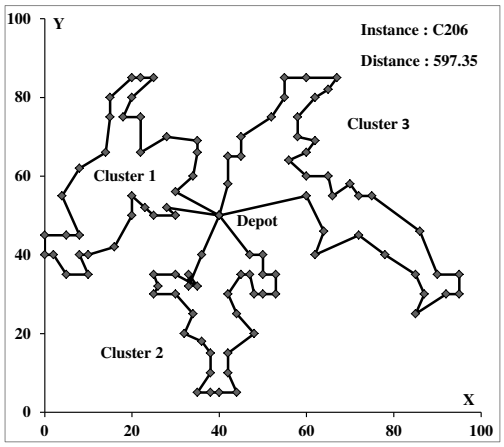

Fig. 4. Solution obtained by algorithm 3PA on instance C206 after the second step (Beam Search): $m = 3$, Distance=597.35

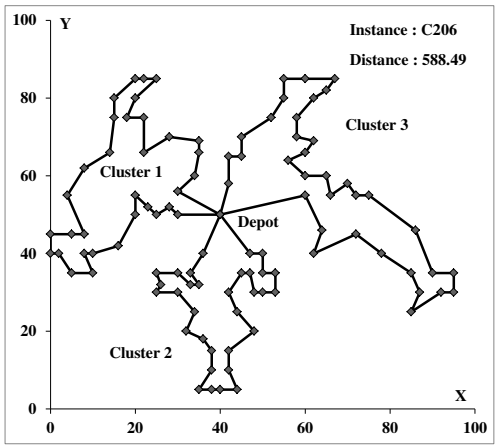

Fig. 5. Solution obtained by algorithm 3PA on instance C206 after the third step (Local Search): $m = 3$, Distance=588.49

has not changed.

V. Conclusion and future work

In this work a three-phase algorithm denoted by 3PA is proposed in order to solve the capacitated vehicle routing problem with time windows (CVRPTW). The central phase is that computing the shortest paths in a given cluster. To do so, a beam search is proposed. The most characteristic of beam search is that it explores several paths in parallel and increases then the probability to find *good* paths. The results obtained on the instances used show that the method is competitive since the computations revealed that algorithm 3PA was better than a method based on goal programming and genetic algorithm (GP-GA). Indeed, 3PA obtained a gap closer to the best known results in the literature than the gap obtained by GP-GA.

As a future work, it will be interesting to add a global evaluation criterion for beam search in order to provide solutions of better quality before calling the local search (third) phase and then to improve the overall solution.

References

[1] H. Akeb and M. Hifi. Algorithms for the circular two-dimensional open dimension problem. *International Transactions in Operational Research*, volume 15, pages 685–704, 2008.

[2] N. Azi, M. Gendreau, and J-Y. Potvin. An exact algorithm for a single vehicle routing problem with time windows and multiple routes. *European Journal of Operational Research*, volume 178, pages 755–766, 2007.

[3] 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*, volume 202, pages 756–763, 2010.

[4] R. Baldacci, E.A. Hadjiconstantinou, and A. Mingozzi. An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Reserach*, volume 52, pages 723–738, 2004.

[5] R. Baldacci and V. Maniezzo. Exact methods based on node-routing formulations for undirected arc-routing problems. Networks, volume 47, pages 52–60, 2006.

[6] J.C.S. Brandão and A. Mercer. A tabu search algorithm for the multi-trip vehicle routing and scheduling problem. *European Journal of Operational Research*, volume 100, pages 180–191, 1997.

[7] A.M. Campbell and M. Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, volume 38, pages 369–378, 2004.

[8] I.M. Chao, B.L. Golden, and E.A. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, volume 88, pages 101–111, 1996.

[9] S. Chen, B. Golden, and E. Wasil. The split delivery vehicle routing problem: Applications, algorithms, test problems, and computational results. *Networks*, volume 49, pages 661–673, 2007.

[10] J-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, volume 52, pages 928–936, 2001.

[11] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, volume 44, pages 216–229, 2004.

[12] K. Ghoseiri and S.F. Ghannadpour. Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm. *Applied Soft Computing*, volume 10(4), pages 1096–1107, 2010.

[13] X. Li and P. Tian. An ant colony system for the open vehicle routing problem. *Lecture Notes in Computer Science*, 4150, M. Dorigo et al. editions, Springer, pages 356–363, 2006.

[14] A. Lim and X. Zhang. A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, volume 19(3), pages 443–457, 2007.

[15] P.S. Ow and T.E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, volume 26(1), pages 35–62, 1988.

[16] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Computers Research*, volume 34, pages 2403–2435, 2007.

[17] J.Y. Potvin and S. Bengio. The vehicle routing problem with time windows. Part II. Genetic search. *INFORMS Journal of Computing*, volume 8, pages 165–172, 1996.

[18] Y. Rochat and E.D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, volume 1, pages 147–167, 1995.

[19] M.M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, volume 35(2), pages 254–265, 1987.

[20] M.M. Solomon. VRPTW benchmark problems. http://w.cba.neu.edu/~msolomon.

[21] K.C. Tan, Y.H. Chew, and L.H. Lee. A hybrid multiobjective evolutionary algorithm for solving vehicle routing problem with time windows. *Computational Optimization and Applications*, volume 34, pages 115–151, 2006.

[22] K.C. Tan, L.H. Lee, Q.L. Zhu, and K. Ou. Heuristic methods for vehicle routing problem with time windows. *Artificial Intelligence in Engineering*, volume 15, pages 281–295, 2001.