RAIRO-Oper. Res. 42 (2008) 415–431
DOI: 10.1051/ro:2008021

RAIRO Operations Research
www.rairo-ro.org

A MEMETIC ALGORITHM FOR THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

Nacima Labadi, Christian Prins and Mohamed Reghioui[1]

Abstract. This article deals with the vehicle routing problem with time windows (VRPTW). This problem consists in determining a least-cost set of trips to serve customers during specific time windows. The proposed solution method is a memetic algorithm (MA), a genetic algorithm hybridised with a local search. Contrary to most papers on the VRPTW, which minimize first the number of vehicles, our method is also able to minimize the total distance travelled. The results on 56 classical instances are compared to those of the best metaheuristics. The efficiency of the MA is similar for the classical criterion, but it becomes the best algorithm available for the total distance, being much faster and improving 20 best-known solutions.

Résumé. Cet article concerne le problème de tournées de véhicules avec fenêtres horaires (Vehicle Routing Problem with Time Windows ou VRPTW). Ce problème consiste à déterminer un ensemble de tournées de coût total minimal pour servir des clients dans des fenêtres horaires spécifiques. Nous proposons un algorithme mémétique (MA, algorithme génétique hybridé avec une recherche locale) pour le résoudre. Contrairement à la plupart des articles sur le VRPTW, qui minimisent en priorité le nombre de véhicules, notre méthode peut aussi minimiser la distance totale parcourue. Les résultats sur 56 problèmes-tests classiques sont comparés à ceux des meilleures métaheuristiques. Pour le critère classique, le MA offre des performances similaires, mais il devient le meilleur algorithme disponible pour la distance totale, en étant bien plus rapide et en améliorant 20 des meilleures solutions connues.

Keywords. memetic algorithm, vehicle routing problem, time window.

Mathematics Subject Classification. 90B06.

Received July 27, 2007. Accepted April 01, 2008.

[1] Inst. Charles Delaunay, Univ. Technologie Troyes, FRE CNRS 2848, BP 2060, 10010 Troyes Cedex, France; {nacima.labadi,christian.prins,mohamed.reghioui_hamzaoui}@utt.fr

Article published by EDP Sciences

1. Introduction

The VRPTW is a generalization of the well-known capacitated vehicle routing problem in which the service of each customer must begin within a specified time window. It is defined on a complete undirected graph $G = (V, E)$ with a node set $V = \{0, 1, 2, ..., n\}$ and an edge set E. Node 0 represents a depot where a fleet of identical vehicles of capacity W is located. The n other nodes correspond to the customers. Each customer i has a demand q_i and a time window $[e_i, l_i]$, where e_i and l_i are respectively the earliest and the latest service time. Arriving at i earlier than e_i is allowed but induces a waiting time a_i, while arriving later than l_i leads to infeasibility. A traversal cost (distance) $d_{ij} = d_{ji}$ and a traversal time $t_{ij} = t_{ji}$ are associated with each edge $[i, j]$. The objective is to build a set of vehicle trips of minimum total cost, such that each trip starts and ends at the depot and services a subset of customers within their time windows. Each customer must be visited by a single trip, *i.e.*, split deliveries are not allowed.

The VRPTW was studied for the first time by Solomon [22]. This author proposed benchmark problems and constructive heuristics to minimize the number of vehicles and, in case of ties, the total distance travelled. This hierarchical objective function, called Solomon's objective in the sequel, is unusual in vehicle routing: for instance, most articles on the VRP minimize the total distance without taking the number of vehicles used into account [20]. However, almost all authors after Solomon have used the same objective to compare their results. Note that minimizing the number of vehicles is already difficult, since the existence of a feasible solution for a given number of vehicles is already NP-complete [22].

Apart from exact methods, which are beyond the scope of this paper, several metaheuristics are available to solve the VRPTW with the objective of Solomon. Rochat and Taillard [21] and Taillard *et al.* [23] designed tabu search methods. Evolutionary algorithms are more widely used, especially genetic algorithms: Blanton and Wainwright [4], Thangiah [27], Potvin and Bengio [19], Berger *et al.* [3], Tan *et al.* [24,25], Jung and Moon [15], and Berger *et al.* [2]. More recent frameworks known as evolutionary strategies were developed by Homberger and Gehring [13] and by Mester [17]. There exist also some hybrid approaches combining genetic and tabu search algorithms: Gehring and Homberger [9,10], Wee Kit *et al.* [28], and Homberger and Gehring [14].

Two recent papers of Bräysy and Gendreau [6,7] offer a good survey of the different heuristic methods developed for the VRPTW with the objective of Solomon. Bräysy *et al.* [5] published another survey, dedicated to evolutionary methods. To the best of our knowledge, only five studies consider the total distance as main objective: Tan *et al.* [24–26], Jung and Moon [15] and Alvarenga *et al.* [1]. Note that nobody has tried to minimize the total duration of trips: travelling times are only used to check time windows. However, all existing methods can be easily adapted to handle this criterion.

In our opinion, Solomon's objective is not very realistic since the fleet of vehicles is often purchased before planning the routes. Moreover, distribution may be partly subcontracted and, in that case, the number of vehicles is not critical and

the cost mainly depends on the distance travelled. These reasons motivated us to develop a flexible memetic algorithm, able to tackle Solomon's objective or the total distance, while being competitive with already published methods.

The article presents this memetic algorithm (MA) for the VRPTW. Genetic algorithms (GA), introduced for the first time by Holland [12], are not aggressive enough for combinatorial optimization problems, compared to other metaheuristics like tabu search. Memetic algorithms proposed by Moscato [18] are more powerful versions which apply a local search procedure to each new solution. The main components of our MA are described in Section 2. In Section 3, numerical experiments are conducted to evaluate the performances of the method for the two objective functions. Some conclusions and perspectives are given in the last section.

2. Memetic algorithm for the VRPTW

2.1. Chromosomes and evaluation

Each chromosome is coded as a sequence S containing the n customers but no trip delimiter. This list can be viewed as a giant tour which ignores vehicle capacity and time windows. Since no special symbol is included to delimit successive trips, all chromosomes have the same length and may be combined using classical crossover operators designed for the TSP (Travelling Salesman Problem). This encoding also allows a significant reduction of solution space. The best VRPTW solution, subject to the sequence defined by S, can be computed using an optimal splitting procedure called *Split* and described in the sequel.

The cost of S is defined by equation (1), in which $TD(S)$ and $NV(S)$ denote the total distance and the number of vehicles used after splitting. M is a parameter set to 0 when minimizing the total distance, or to a large positive value when the number of vehicles must be minimized in priority (Solomon's objective).

$$F(S) = M \times NV(S) + TD(S). \tag{1}$$

The *Split* procedure, initially designed by Prins [20] for the VRP, is here extended to tackle time windows. *Split* computes a shortest path in an auxiliary graph H containing one dummy node 0 and n other nodes corresponding to the n customers. Each subsequence of customers $(S_i, S_{i+1}, \ldots, S_j)$ corresponding to a feasible trip is modelled by a weighted arc $(i-1, j)$ in H. If $M = 0$, the weight assigned to this arc is the trip length. Otherwise, the arc is weighted by the sum of M and the trip length. Note that any trip which violates time windows or vehicle capacity is discarded at this level.

The shortest path from node 0 to node n in H can be computed using Bellman's algorithm for directed acyclic graphs. It indicates where to split S to get an optimal VRPTW solution, subject to the order imposed by S. If $M = 0$, the total length of the routes is minimized. Otherwise, the number of arcs of the shortest path (vehicles used) is minimized, followed by the total length in case of ties.

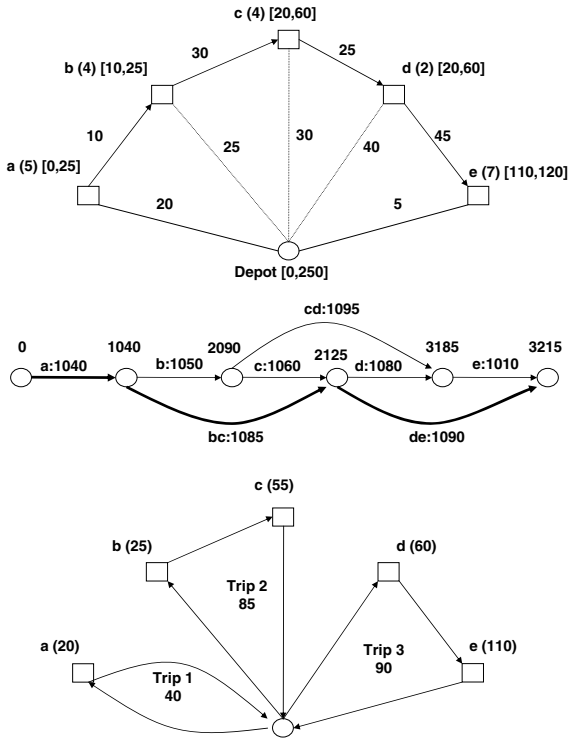

FIGURE 1. Example of *Split* for the VRPTW.

Figure 1 gives a example of splitting for the instance defined by Table 1. To simplify, travel times are here equal to distances, *i.e.*, $t_{ij} = d_{ij}$ for each edge $[i, j]$. The goal is to minimize the number of vehicles used and, in case of ties, the total length of the trips (Solomon's objective), by setting $M = 1000$. The upper part of the figure shows a chromosome $S = (a, b, c, d, e)$, viewed as a giant tour. Each customer has a demand in brackets and a time window in square brackets.

The auxiliary graph is given in the middle of the figure, assuming a vehicle capacity $W = 10$. Each arc represents a feasible trip. For instance, arc a models the trip reduced to customer a: its length 40 is twice the length of edge $[0, a]$. The only feasible trip with a waiting time serves customers d and e: it leaves the depot at time 20, node d at 60, and waits 5 units of time at node e before leaving it at time 110. However, since the goal is not to minimize the total duration, arc de is weighted by the trip length (90) and not its duration (95). Arc ab is not represented, since customer b is already closed when a vehicle can reach it, at time 30. All the other trips which are not represented violate vehicle capacity.

Bellman's algorithm gives a shortest path with a cost $F(S) = 3215$. The labels computed are given above each node. The number of vehicles used $NV(S) = 3$ and the total distance $TD(S) = 215$ are respectively the quotient and the remainder of

TABLE 1. Data for the instance used in Figure 1.

Node	e_i	l_i	q_i	Edge	$d_{ij} = t_{ij}$
0	0	250	0	[0, a]	20
a	0	25	5	[0, b]	25
b	10	25	4	[0, c]	30
c	20	60	4	[0, d]	40
d	20	60	2	[0, e]	5
e	110	120	7	[a, b]	10
				[b, c]	30
				[c, d]	25
				[d, e]	45

the integer division of $F(S)$ by M. The lower part of the figure shows the resulting solution, with departure times in brackets.

It is important to note that the resulting solution is optimal for the order defined by S. In fact, there always exists an optimal chromosome, *i.e.*, one giving a globally optimal VRPTW solution after splitting. Indeed, consider an optimal solution T and concatenate its trips to form a chromosome S: *Split* will obviously find this optimal solution when applied to S. Thus, there is no loss of information when the MA explores the set of customer permutations (giant tours) instead of the larger set of feasible VRPTW solutions. The only price to pay is the running time of *Split*. The complexity of Bellman's algorithm is $O(m)$, where m is the number of arcs in the auxiliary graph. In the worst case, $m = n(n+1)/2$ and *Split* is then in $O(n^2)$. In practice, m is much smaller because most long subsequences are infeasible.

2.2. POPULATION

The population is stored in a table *Pop* with a fixed number ns of chromosomes, sorted in increasing order of costs. The initial population includes three good solutions, computed by the sequential heuristic of Solomon [22] and two methods inspired by classical VRP heuristics: the savings algorithm of Clarke and Wright [8] and the sweep heuristic of Gillett and Miller [11].

The heuristic of Solomon starts from a solution reduced to a dummy loop on the depot. At each iteration, for each customer not yet serviced, the heuristic evaluates all feasible insertions in non-empty trips plus the insertion, always feasible, in a new empty trip. The best insertion found is executed. The Clarke and Wright heuristic for the VRP starts from a trivial solution with one trip per customer. At each iteration, all feasible mergers (concatenations of two trips) are evaluated and the one that provides the maximal saving is performed. These mergers stop when additional concatenations would violate vehicle capacity or increase total cost. For the VRPTW, time windows must be also checked when evaluating each merger.

In the sweep heuristic, clusters of customers compatible with vehicle capacity are generated by rotating a half-line centered on the depot. One vehicle route is

then computed in each cluster, using a TSP heuristic. This method gives poor results for the VRPTW, because it is difficult to know in advance if all customers of a cluster can be visited by a tour without time window violation. To bypass this problem, we combined the two phases of the heuristic as follows. Starting from the customer with the smallest polar angle, the first tour is built iteratively by adding the nearest customer with compatible demand and time window. When no further insertion is possible, the current tour is closed and a new tour is started from the customer not yet serviced with the smallest polar angle.

The three initial heuristic solutions are improved by a local search procedure based on the cross-exchange operator of Taillard *et al.* [23]. This move consists in exchanging two sequences of customers between two different trips. Each sequence moved may be as long as the trip which contains it. The $ns - 3$ other initial solutions of *Pop* are generated by randomizing Solomon's heuristic: instead of inserting the customer with the best insertion cost, a customer is randomly selected among the five best. Each initial solution is converted into a chromosome by concatenating its trips and the resulting chromosome is evaluated by *Split*, which often brings an additional improvement by shifting several trip limits.

To guarantee a sufficient diversity, a simple cost spacing rule is used: an initial solution or a solution generated by crossover is added to the population if there is no solution in *Pop* with a cost difference smaller than a given threshold Δ. In other words, a solution S is accepted in *Pop* if and only if equation (2) holds. Recall that the cost $F(S)$ of solution S is defined by equation (1)

$$\forall\, T \in Pop : |F(S) - F(T)| \geq \Delta. \tag{2}$$

2.3. Selection and crossover

Parents are selected using the binary tournament method: two chromosomes are randomly selected in *Pop* and the best one is the first parent, P_1. This process is repeated to get the second parent, P_2. These parents are then combined using the order crossover or *OX*, a classical crossover for the travelling salesman problem.

OX randomly chooses two cutting points i and j ($1 \leq i \leq j \leq n$) in P_1. Customers $P_1(i)$ to $P_1(j)$ are then copied into child C, at the same positions. The child solution is completed by browsing circularly the second parent, from position $j + 1$ to position i. The insertion of customers into the child is also performed in a circular way, from position $j + 1$ onwards. OX usually produces two children, by interchanging the role of the two parents. In our algorithm, only one child is generated, by choosing randomly the order of parents. Figure 2 gives an example of OX for two chromosomes with 11 customers.

2.4. Local search

In Moscato's MA model [18], the local search is systematically applied to each new child. In our implementation, it is called with a given probability p_{ls} and

```
P1:  1   2   5  10  |  7   6   8  |  9  11   4   3
P2:  9   5   6  11  |  4   2   7  |  1   3   8  10
C :  5  11   4   2  |  7   6   8  |  1   3  10   9
```

FIGURE 2. Example of OX crossover.

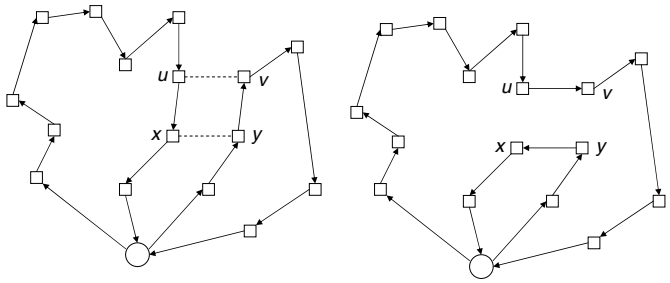

FIGURE 3. Example of 2-opt* move.

operates on a complete solution obtained by *Split*. Each iteration of the local search evaluates the four following moves:

(1) Or-opt: relocate one or two consecutive customers in one or two trips.
(2) 2-opt: invert a chain of customers in one trip.
(3) Exchange: swap two customers pertaining to one or two trips.
(4) 2-opt*: interchange the last parts of two trips as shown in Figure 3.

In Figure 3, there exist another way to repair the routes when arcs (u, x) and (y, v) are removed: add (u, y) and (v, x). However, this move inverts the traversal direction of one subsequence in each route, which often raises a time window violation. As it is rarely feasible, this move was not selected.

An additional move is used when the main objective is to minimize the number of vehicles or trips (case $M > 0$). This move tries to empty the least loaded trip T by moving its customers to other routes. A necessary condition is checked first: the sum of residual capacities in the other routes must not be smaller than the load of T. Then, the move consists in performing the best possible relocation (the one with the smallest cost variation) for each customer of T. If this sequence of relocations succeeds, we try to empty the least loaded remaining trip.

The problem is that this emptying process can fail. Moreover, the relocations already performed have often increased the total length of the routes, without reducing the number of vehicles. In that case, the solution is restored in its initial state. Since this move is time-consuming, it is tested separately at the beginning of the local search, before evaluating the other moves.

Each iteration of the local search (neighborhood exploration) searches for the first improving move and executes it if found. The local search stops when no

further improvement is possible. The trips of the obtained solution are then concatenated to return a chromosome without trip delimiters. *Split* is finally called to deduce the fitness and sometimes obtain an additional improvement.

Two procedures described in Kindervater and Savelsbergh [16] are used to update departure times and check time windows when a customer is inserted or removed in a route (swap and exchange moves). Both are based on the following remark: if the order of customers after the modification is preserved, we just have to check that the arrival time at each node is still compatible with its time window.

More precisely, consider a route $R = (R_1, \ldots, R_u, \ldots, R_v)$ and let T_k be the departure time at a node R_k in R. If the vehicle leaves R_u earlier at $T_u - \theta$, for instance if a move removes node R_{u-1}, a *Pull* procedure reduces the departure time of each subsequent node R_k by $B_k = \min\{B_{k-1}, T_k - e_k\}$, with $B_u = \theta$. Such a shift is always feasible but *Pull* is necessary to keep correct departure times.

If the departure from R_u is delayed by θ, for instance when a move inserts one customer before it, the *Push* procedure consists in augmenting the departure times of each subsequent node R_k by $F_k = \max\{F_{k-1} - a_k, 0\}$, where a_k is the waiting time at R_k and $F_u = \theta$. The time window of R_k is violated if $T_k + F_k > l_k$. The feasibility check when a move delays T_u can be done in $O(1)$, by pre-computing for each node R_k the maximum delay without violating a time window after it: $D_k = \min\{l_i - T_i + a_i : i = k+1, \ldots, v\}$. Hence, T_u can be delayed by θ if $\theta \leq D_u$.

We designed similar tests in $O(1)$ for 2-opt and 2-opt* moves, although some trip segments are inverted. The arrays required by all these tests are initialized at the beginning of the local search and updated after each improving move. Using these techniques, the complexity of one neighborhood exploration can be kept in $O(n^2)$, like for the classical VRP. In fact, the local search is often faster in the VRPTW, because many moves are infeasible and can be rejected before computing their cost variations.

2.5. General structure of the memetic algorithm

The general structure of the memetic algorithm is given by Algorithm 1. It starts by building an initial population *Pop* of ns chromosomes, as explained in Section 2.2. Each iteration of the *repeat* loop chooses two parents P_1 and P_2 using the binary tournament and applies the OX crossover to get one child C which is evaluated by *Split*. The resulting VRPTW solution T undergoes the local search with a given probability p_{ls} and its trips are concatenated to give an improved chromosome. Contrary to a classical GA, there is no mutation operator: population diversity is here guaranteed by the cost dispersal rule.

The chromosome $Pop(b)$ to be replaced by C is randomly selected in the worst half of *Pop*, *i.e.*, among the $\lfloor ns/2 \rfloor$ last solutions. This chromosome is actually replaced if the cost dispersal rule is respected or if the current best solution $Pop(1)$ is improved. Children that do not match these conditions are discarded. The population is re-sorted in increasing cost order after each successful replacement.

The *for* main loop performs a fixed number of phases np. A phase is a short MA stopping after a maximum number of iterations α_{max} or a maximum number

of iterations without improvement β_{max} (*repeat* loop). The first phase operates on the initial population. All phases end with a partial renewal of population: the best nc chromosomes are kept and the others are replaced by new solutions computed with the randomized version of Solomon's heuristic. Like in the initial population, new solutions must satisfy the cost spacing rule.

Algorithm 1 : *General structure of memetic algorithm.*

build the initial population Pop of ns chromosomes
sort Pop in increasing order of cost
for $phase := 1$ **to** np **do**
  $\alpha, \beta := 0$;
  repeat
    $\alpha := \alpha + 1$
    select two parents P_1 and P_2 in Pop by binary tournament
    apply OX crossover to the parents to get one child C
    evaluate C with *Split* and keep the resulting VRPTW solution T
    if random $\leq p_{ls}$ **then**
      apply local search to T
      concatenate the trips of T to rebuild C and evaluate it with *Split*
    end if
    select randomly a chromosome $Pop(b)$ to be replaced, with $b \geq \lfloor ns/2 \rfloor$
    if $F(C) < F(Pop(1))$ **then**
      $\beta := 0$
    else
      $\beta := \beta + 1$
    end if
    if $F(C) < F(Pop(1))$ or $\forall S \in Pop \setminus \{Pop(b)\} : |F(C) - F(S)| \geq \Delta$ **then**
      $Pop(b) := C$
      shift $Pop(b)$ to keep population Pop sorted
    end if
  until $(\alpha = \alpha_{max})$ or $(\beta = \beta_{max})$
  if $phase < np$ **then**
    apply the partial renewal procedure, keeping the nc best solutions
    re-sort Pop
  end if
end for

3. Computational results

3.1. Implementation and instances

All algorithms were written in Delphi and tested on a 3 GHz PC with Windows XP Pro. The evaluation is based on 56 instances of 100 customers proposed by

Solomon [22] and available at: http://w.cba.neu.edu/~msolomon/problems.htm. There exist also 56 problems with $n = 25$ customers and 56 with $n = 50$, but they were discarded because all are now solved by exact methods.

The instances are partitioned into six classes: R1, R2, C1, C2, RC1 and RC2. Customers are randomly located in a 100×100 square in classes R1 and R2 and clustered in C1 and C2. Problems in sets RC1 and RC2 mix clustered and randomly distributed customers. The positions of customers are identical within each class: only time windows differ. Classes R2, C2 and RC2 have wider time windows and require less vehicles than R1, C1 and RC1. All demands, coordinates and window limits are integers. However, both travel times and distances are equal to the Euclidean distance between nodes, computed using double precision real numbers.

3.2. Parameters settings

The memetic algorithm can minimize either the total distance alone or the objective of Solomon (number of vehicles, then total distance). The parameter M is respectively set to 10 000 and 0 in the two versions, called MA1 and MA2 in the sequel. The other parameters result from several preliminary tests and are identical for both versions: a population containing $ns = 30$ chromosomes, a local search rate $p_{ls} = 0.1$, a minimum cost spacing $\Delta = 0.2$, a maximum number of iterations per phase $\alpha_{max} = 3000$, a maximum number of iterations per phase without improvement $\beta_{max} = 2000$, only one solution kept in partial renewals ($nc = 1$), and a number of phases $np = 10$.

The initial versions used a single phase with $\alpha_{max} = 30\,000$ iterations, but we observed better results if this number of crossovers is spread over 10 phases, with a partial renewal of solutions between two consecutive phases. The resulting structure which alternates between reproduction phases and diversification can be viewed as a transition form towards scatter search methods.

The small size of the population compared to classical GAs is typical of such memetic algorithms: with a larger population, the percentage of unproductive iterations (when the child C is rejected) becomes significant and the algorithm wastes time in useless crossovers. The most important component is the local search, which consumes 90% of the total running time while being essential for solution quality. Since the local search is not systematic, solutions not improved by this procedure can be inserted in the population, contributing to a better diversity in conjunction with the cost spacing rule.

The other components are less critical than the local search but the choice of OX as crossover operator and of the binary tournament as selection rule brings a slight improvement of solution costs, compared to other strategies.

3.3. Results for the total distance – MA1

As mentioned in introduction, only five published algorithms minimize the total distance. MA1 is compared here with the best of them, the Column Generation Heuristic (CGH) of Alvarenga *et al.* [1]. CGH is a sophisticated cooperative

TABLE 2. Total distance per instance for classes C1 and C2.

	CGH		MA1		Time(s)	Gap(%)
	NV	TD	NV	TD		
C101	10	828.937	10	828.937	63.33	0.00
C102	10	828.937	10	828.937	82.08	0.00
C103	10	828.065	10	828.065	82.38	0.00
C104	10	824.777	10	824.777	102.44	0.00
C105	10	828.937	10	828.937	66.31	0.00
C106	10	828.937	10	828.937	72.81	0.00
C107	10	828.937	10	828.937	59.58	0.00
C108	10	828.937	10	828.937	83.70	0.00
C109	10	828.937	10	828.937	90.03	0.00
Mean C1	10.00	828.378	10.00	828.378	78.07	0.00
C201	3	591.557	3	591.557	128.28	0.00
C202	3	591.557	3	591.557	133.13	0.00
C203	3	591.173	3	591.173	132.30	0.00
C204	3	590.599	3	590.599	149.14	0.00
C205	3	588.876	3	588.876	155.22	0.00
C206	3	588.493	3	588.493	179.11	0.00
C207	3	588.286	3	588.286	215.69	0.00
C208	3	588.324	3	588.324	197.48	0.00
Mean C2	3.00	589.858	3.00	589.858	161.29	0.00
Mean C	6.70	716.133	6.70	716.133	159.83	0.00

method, alternating between a genetic algorithm, which builds a set of high-quality trips, and the resolution of a partitioning problem with a public domain linear programming solver (GLPK), to select a subset of trips covering all customers.

Jung and Moon [15] published the best and average results of a hybrid GA, but for 100 runs. Their best results slightly outperform CGH but their average results are clearly inferior. This is why our comparison is limited to MA1 and CGH, which are evaluated in the same conditions, using one run only.

Tables 2 to 4 respectively give the results of CGH and MA1 on classes C, R and RC, using a common format. The first column indicates the instance name. Columns 2 and 3 provide the number of vehicles *NV* and the total distance *TD* obtained by CGH. The results obtained by MA1 are shown in columns 4 and 5. The two last columns give the running times in seconds of MA1 and the deviation in percent between MA1 and CGH, computed as follows: $(\textit{MA1 distance}/\textit{CGH distance} - 1) \times 100$. The average values for each column are given after the last file of each class. An additional row at the end of Table 4 gives the average values over the 56 instances.

MA1 improves CGH 20 times, gives identical results on 20 instances and inferior results on 16 other problems. The results of both methods are identical for clustered problems (classes C1 and C2), which seem to be the easiest ones. MA1

TABLE 3. Total distance per instance for classes R1 and R2.

	CGH		MA1		Time(s)	Gap(%)
	NV	TD	NV	TD		
R101	20	1642.870	20	1644.045	87.95	0.07
R102	18	1472.620	18	1472.815	130.92	0.01
R103	14	1213.620	14	1213.624	156.28	0.00
R104	11	986.096	12	1000.900	163.44	1.50
R105	15	1360.783	15	1360.783	113.28	0.00
R106	13	1241.518	13	1240.468	177.16	-0.08
R107	11	1076.125	11	1074.243	198.50	-0.18
R108	10	948.573	11	952.325	209.09	0.40
R109	13	1151.839	12	1154.551	170.76	0.24
R110	12	1092.347	12	1072.415	138.63	-1.83
R111	12	1053.496	12	1053.802	149.84	0.03
R112	10	960.675	11	969.992	125.72	0.97
Mean R1	13.25	1183.380	13.42	1184.164	151.80	0.09
R201	9	1148.483	8	1150.917	207.80	0.21
R202	7	1049.737	7	1037.498	178.59	-1.17
R203	5	900.080	6	874.869	194.48	-2.80
R204	4	772.330	5	735.861	264.94	-4.72
R205	6	970.886	6	960.079	180.03	-1.11
R206	5	898.914	5	879.893	192.41	-2.12
R207	4	834.930	4	800.786	265.95	-4.09
R208	3	723.610	3	706.855	191.53	-2.32
R209	6	879.531	5	859.390	198.69	-2.29
R210	7	932.887	6	912.533	208.58	-2.18
R211	5	787.511	4	755.949	237.05	-4.01
Mean R2	5.55	899.900	5.36	879.512	210.91	-2.42
Mean R	9.57	1047.80	9.57	1038.46	180.07	-1.11

finds better results on problems with large time windows (classes R2 and RC2), with a saving exceeding 4% for two problems. This is remarkable, because these instances which tend to a VRP are the hardest ones for exact methods.

In particular, all solution values achieved by CGH except one are improved in class R2. Even when MA1 is outperformed by CGH, the worst deviation is 1.63% (problem RC108). Overall, MA1 is better than CGH, with an average gap of –0.45%. Although the number of vehicles is not considered in the optimization criterion, we can see that MA1 and CGH find in general the same values. MA1 uses sometimes one additional vehicle, but never more. Concerning running times, Alvarenga *et al.* do not specify the computer they use, but they indicate that CGH is stopped after one hour of execution. In comparison, MA1 is very fast since its duration is less than 3 min on average and does not exceed 4.5 min.

TABLE 4. Total distance per instance for classes RC1 and RC2.

	CGH		MA1		Time(s)	Gap(%)
	NV	TD	NV	TD		
RC101	16	1639.968	17	1658.991	138.14	1.16
RC102	14	1466.840	14	1480.363	120.23	0.92
RC103	11	1264.707	12	1276.050	181.11	0.90
RC104	10	1135.520	10	1139.887	199.58	0.38
RC105	16	1518.600	16	1524.220	120.56	0.37
RC106	13	1377.352	13	1388.095	139.09	0.78
RC107	12	1212.830	12	1212.833	156.41	0.00
RC108	11	1117.526	11	1135.734	163.45	1.63
Mean RC1	12.88	1341.668	13.13	1352.022	152.32	0.77
RC201	9	1274.537	10	1273.040	196.69	-0.12
RC202	8	1113.526	8	1099.542	233.11	-1.26
RC203	5	945.960	6	937.449	156.27	-0.90
RC204	4	799.670	4	791.399	197.45	-1.03
RC205	7	1161.810	8	1168.651	235.36	0.59
RC206	7	1059.886	7	1054.606	243.22	-0.50
RC207	7	976.396	6	966.372	138.11	-1.03
RC208	5	795.391	5	783.932	196.87	-1.44
Mean RC2	6.50	1015.897	6.75	1009.374	199.63	-0.71
Mean RC	9.69	1178.78	9.94	1180.70	175.98	0.03
Global mean	8.73	984.54	8.80	981.25	159.83	-0.45

3.4. RESULTS FOR SOLOMON'S OBJECTIVE – MA2

Since many metaheuristics have been published for this criterion, it is difficult to provide a comparison instance per instance. In their surveys, Bräysy *et al.* [5] and Bräysy and Gendreau [7] prefer to use two convenient indicators to rank published algorithms: the cumulative number of vehicles *CNV* and the cumulative total distance *CTD*, over the 56 instances.

The upper part of Table 5 gives these indicators for the best published evolutionary algorithms and MA2. Two rows give for each method the number of vehicles and the total length of the routes. To get a more compact table with smaller numbers, columns R1 to RC2 contain *average* values per instance. The *cumulative* values *CNV* and *CTD* are listed in the *global* column. The table shows that MA2 offers a good tradeoff between the number of vehicles and the total distance. MA2 does not supersede the two current best algorithms for the number of vehicles, Berger *et al.* [2] and Homberger and Gehring [14], but it does better than the algorithms of Wee Kit *et al.* [28] and Thangiah [27], for instance.

To better illustrate the conflicting nature of the two objectives, we added to Table 5 a lower part for the best algorithms designed to minimize the total distance, including MA1. The best and average results for 100 runs obtained by the HGA

TABLE 5. Cumulative indicators of the best evolutionary algorithms.

Solomon's objective	R1	R2	C1	C2	RC1	RC2	Global
Berger *et al.*	12.58	3.09	10.00	3.00	12.13	3.50	424
(1998)	1261.58	1030.01	834.61	594.25	1441.35	1281.25	60539
Berger *et al.*	11.92	2.73	10.00	3.00	11.50	3.25	405
(2003)	1221.10	975.43	828.48	589.93	1389.89	1159.37	57952
Gehring, Homberger	12.42	2.82	10.00	3.00	11.88	3.25	415
(1999)	1198	947	829	590	1356	1140	56942
Gehring, Homberger	12.00	2.73	10.00	3.00	11.50	3.25	406
(2001)	1217.57	961.29	828.63	590.33	1395.13	1139.37	57641
Homberger, Gehring	11.92	2.73	10.00	3.00	11.63	3.25	406
(1999)	1228.06	969.95	828.38	589.86	1392.57	1144.43	57876
Homberger, Gehring	11.92	2.73	10.00	3.00	11.50	3.25	405
(2005)	1212.73	955.03	828.38	589.86	1386.44	1123.17	57309
MA2	12.75	3.09	10.00	3.00	12.37	3.62	429
This paper	1188.01	920.86	828.38	589.86	1351.27	1087.18	56067
Mester	12.00	2.73	10.00	3.00	11.50	3.25	406
(2002)	1208	954	829	590	1387	1119	57219
Potvin, Bengio	12.58	3.00	10.00	3.00	12.13	3.38	422
(1996)	1296.83	1117.64	838.11	590.00	1446.25	1368.13	62634
Thangiah	12.75	3.18	10.00	3.00	12.50	3.38	429
(1995)	1300.25	1124.28	892.11	749.13	1474.13	1411.13	65074
Wee Kit *et al.*	12.58	3.18	10.00	3.00	12.75	3.75	432
(2001)	1203.32	951.17	833.32	593.00	1382.06	1132.79	57265
Total distance	R1	R2	C1	C2	RC1	RC2	Global
Alvarenga *et al.*	13.25	5.55	10.00	3.00	12.88	6.50	489
(2007)	1183.38	899.90	828.38	589.86	1341.67	1015.90	55134
Jung, Moon (2002)	13.25	5.36	10.00	3.00	13.00	6.25	486
best of 100 runs	1179.95	878.41	828.38	589.86	1343.64	1004.21	54779
Jung, Moon (2002)	13.73	5.61	10.00	3.00	13.27	6.43	498
mean of 100 runs	1190.69	885.99	828.40	589.93	1363.61	1014.06	55230
MA1 .	13.42	5.36	10.00	3.00	13.13	6.75	493
This paper	1184.16	879.51	828.38	589.86	1352.02	1009.37	54950
Tan *et al.*	13.17	5.00	10.11	3.25	13.50	5.00	478
(2001a)	1227	980	861	619	1427	1123	58605
Tan *et al.*	12.91	5.00	10.00	3.00	12.60	5.80	471
(2001b)	1205.0	929.6	841.96	611.2	1392.3	1081.1	56931

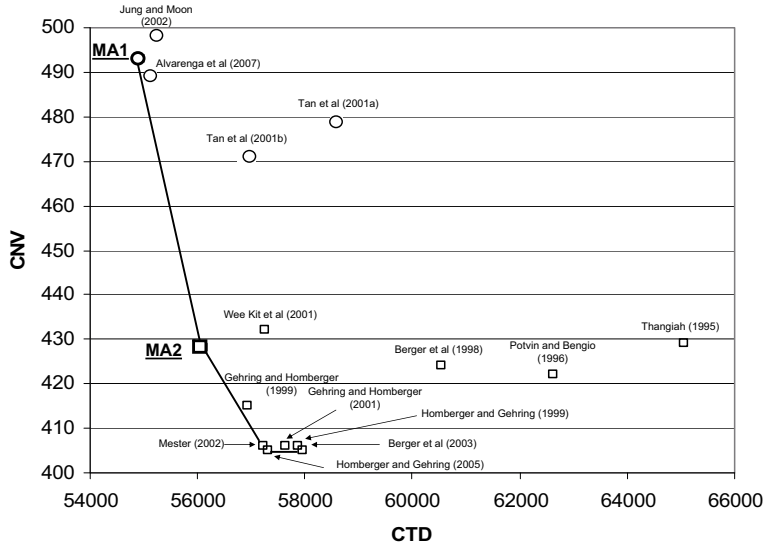

FIGURE 4. Results of evolutionary methods in objective space.

TABLE 6. Compared running times of MA1 and MA2.

Version	R1	R2	C1	C2	RC1	RC2	Global
MA1 total time	151.80	210.91	78.07	161.29	152.32	199.63	159.83
MA1 time to best	67.66	90.58	3.31	6.98	61.69	82.39	54.40
MA2 total time	225.57	392.62	85.65	255.52	154.84	379.61	252.07
MA2 time to best	136.14	208.93	2.44	14.41	79.14	152.73	105.79

of Jung and Moon [15] are also distinguished. We can see that MA2 becomes the best metaheuristic for the total distance, if only one run is allowed.

Figure 4 locates the performance of all methods in the objective space. Small squares indicate methods designed for Solomon's objective, while small circles concern algorithms for the total distance. The broken line connects non-dominated solutions (Pareto front). Clearly, MA1 and MA2 are non-dominated.

Table 6 compares the average running times in seconds of MA1 and MA2. It distinguishes between the total running time, until the programs stop, and the time required until the last improvement (*time to best*). MA2 is slower than MA1, due to the additional moves evaluated in the local search to reduce the number of vehicles. MA1 has a smaller ratio *Time to best/Total time*, indicating a faster convergence. Both algorithms are very fast on classes C1 and C2, for which one of the initial heuristics often finds an optimal or quasi-optimal solution.

4. Conclusion

This paper presents a memetic algorithm for the VRPTW. The first highlight of our method is its flexibility to deal with the two objective functions addressed in the literature. For the minimization of the total distance, MA1 outperforms the best metaheuristic published for this objective, the algorithm CGH of Alvarenga *et al.* [1]. It is also conceptually simpler and much faster. For the objective of Solomon in which priority is given to the number of vehicles, the MA takes place among the best algorithms, even if it does not become the best one.

The key-point behind the effectiveness and the simplicity of our method is the encoding of chromosomes: permutations of customers without trip delimiters. A feasible solution for the VRPTW is deduced optimally from each permutation, using the *Split* procedure. Since we have shown that there always exists a permutation for which *Split* gives an optimal VRPTW solution, the MA can explore a smaller solution space without any loss of information.

The results are less impressive for the number of vehicles but this can be explained. Let C be a chromosome and C_i one customer such that $i < n$ and $e(C_i) > l(C_{i+1})$, which means that the time window of C_{i+1} ends before the beginning of the window of C_i. *Split* is then obliged to cut C between indices i and $i+1$, since no feasible trip can contain these two customers. The problem comes from the OX crossover, which has no control on such pairs of customers: if there are numerous, the solution obtained by *Split* will be poor in terms of vehicles, even if it is optimal for the sequence defined by C and if the local search can save a few vehicles. Therefore, a new crossover able to reduce the number of vehicles (compared to the parents) would be required to improve the memetic algorithm with Solomon's objective.

References

[1] G.B. Alvarenga, G.R. Mateus and G. de Tomi, A genetic and set partitioning two-phase approch for the vehicle routing problem with time windows. *Comput. Oper. Res.* **34** (2007) 1561–1584.

[2] J. Berger, M. Barkaoui and O. Bräysy, A route-directed hybrid genetic approach for the vehicle routing problem with time windows. *Inf. Syst. Oper. Res.* **41** (2003) 179–194.

[3] J. Berger, M. Salois and R. Begin, A hybrid genetic algorithm for the vehicle routing problem with time windows. *Lecture Notes in Artificial Intelligence* **1418**. Springer, Berlin (1998) 114–127.

[4] J.L. Blanton and R.L. Wainwright, *Multiple vehicle routing with time and capacity constraints using genetic algorithms*, Proceedings of the Fifth International Conference on Genetic Algorithms. Morgan Kaufmann, San Francisco (1993) 452–459.

[5] O. Bräysy, W. Dullaert and M. Gendreau, Evolutionary algorithms for the vehicle routing problem with time windows. *J. Heuristics* **10** (2005) 587–611.

[6] O. Bräysy and M. Gendreau, Vehicle routing problem with time windows – Part I: route construction and local search algorithms. *Transportation Science* **39** (2005) 104–118.

[7] O. Bräysy and M. Gendreau, Vehicle routing problem with time windows – Part II: metaheuristics. *Transportation Science* **39** (2005) 119–139.

[8] G. Clarke and J.W. Wright, Scheduling of vehicles from a central depot to a number of delivery points. *Oper. Res.* **12** (1964) 568–581.

[9] H. Gehring and J. Homberger, *A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows*, Proceedings of EUROGEN 99, University of Jyväskylä, Finland (1999) 57–64.
[10] H. Gehring and J. Homberger, Parallelization of a two-phase metaheuristic for routing problems with time windows. *Asia-Pacific J. Oper. Res.* **18** (2001) 35–47.
[11] B.E. Gillett and L.R. Miller, A heuristic algorithm for the vehicle dispatch problem. *Oper. Res.* **22** (1974) 340-349.
[12] J.H. Holland, *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor (1975).
[13] J. Homberger and H. Gehring, Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR* **37** (1999) 297–318.
[14] J. Homberger and H. Gehring, A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *Eur. J. Oper. Res.* **162** (2005) 220–238.
[15] S. Jung and B.R. Moon, *A hybrid genetic algorithm for the vehicle routing problem with time windows*, Proceedings of Genetic and Evolutionary Computation Conference. Morgan Kaufmann, San Francisco (2002) 1309–1316.
[16] G.A.P. Kindervater and M.W.P. Savelsbergh, Vehicle routing: handling edge exchanges. edited by E.H.L. Aarts and J.K. Lenstra, *Local search in combinatorial optimization*. Wiley, Chichester (1997) 311–336.
[17] D. Mester, *An evolutionary strategies algorithm for large scale vehicle routing problem with capacitate and time windows restrictions*, Working paper, Institute of Evolution, University of Haifa, Israel (2002).
[18] P. Moscato, Memetic algorithms: a short introduction, edited by D. Corne, M. Dorigo and F. Glover, *New Ideas in Optimization*. McGraw-Hill, New York (1999) 219–234.
[19] J.Y. Potvin and S. Bengio, The vehicle routing with time windows – Part II: genetic search. *INFORMS J. Comput.* **8** (1996) 165–172.
[20] C. Prins, A simple and effective evolutionary algorithm for the vehicle routing problem, *Comput. Oper. Res.* **31** (2004) 1985–2002.
[21] Y. Rochat and E.D. Taillard, Probabilistic diversification and intensification in local search for vehicle routing. *J. Heuristics* **1** (1995) 147–167.
[22] M.M. Solomon, Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.* **35** (1987) 254-265.
[23] E. Taillard, P. Badeau, M. Gendreau, F. Guertin and J.Y. Potvin, Tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science* **31** (1997) 170–186.
[24] K.C. Tan, L.H. Lee and K. Ou, Hybrid genetic algorithms in solving ehicle routing problems with time window constraints. *Asia-Pacific J. Oper. Res.* **18** (2001) 121–130.
[25] K.C. Tan, L.H. Lee and K. Ou, *A messy genetic algorithm for the vehicle routing problem with time window constraints*, Proceedings of the 2001 Congress on Evolutionary Computation, IEEE, Piscataway (2001) 679–686.
[26] K.C. Tan, L.H. Lee, Q.L. Zhu and K. Ou, Heuristic methods for the vehicle routing problem with time windows. *Artificial Intelligence in Engineering* **15** (2001) 281–295.
[27] S. Thangiah, Vehicle routing with time windows using genetic algorithms. edited by L. Chambers, *Application handbook of genetic algorithms: new frontiers*, Vol II. CRC Press, Boca Raton (1995) 253–277.
[28] H. Wee Kit, J. Chin and A. Lim, A hybrid search algorithm for the vehicle routing problem with time windows. *Int. J. Art. Intell. Tools* **10** (2001) 431–449.