Ant Colony Algorithms for the Dynamic Vehicle Routing Problem with Time Windows

Barry van Veen, Michael Emmerich, Zhiwei Yang,
Thomas Bäck, and Joost Kok

LIACS, Leiden University, Niels Bohrweg 1, 2333-CA Leiden, The Netherlands

Abstract. The Vehicle Routing Problem with Time Windows relates to frequently occuring real world problems in logistics. Much work has been done on solving static routing problems but solving the dynamic variants has not been given an equal amount of attention, while these are even more relevant to most companies in logistics and transportation. In this work an Ant Colony Optimization algorithm for solving the Dynamic Vehicle Routing Problem with Time Windows is proposed. Customers and time windows are inserted during the working day and need to be integrated in partially committed solutions. Results are presented on a benchmark that generalizes Solomon's classical benchmark with varying degrees of dynamicity and different variants, including pheromone preservation and the min-max ant system.

1 Introduction

With recent developments in mobile communication and positioning systems, it is now possible for companies in transportation to view and change their planning during the day. This leads to a new group of dynamic routing problems for which algorithms have to be designed. In this paper we will extend the ant algorithm described in [7] for the standard vehicle routing problem with time windows (VRPTW) to dynamic problems (DVRPTW). As described by Psaraftis [12], a problem is dynamic when some part of the input is revealed to the solver during optimization.This means that we can not build a fixed solution and we have to adjust the solution while the problem changes. In the DVRPTW, the task is to schedule a fleet of vehicles in a working day and new orders (clients that need to be visited) are introduced during the day. This specific problem has not been solved with ant colony algorithms, yet, although it is very relevant in practice. Some features of the bio-mimetic ant-colony optimization algorithm [3] seem to well support dynamic adaptations of delivery routes, as results for the related TSP problem indicate [5]. To test our new approach a set of benchmark problems is introduced as a generalization of a common VRPTW benchmark. To make our results reproducible, the benchmark as well as the C code of the developed algorithms will be made public for other users.

In Section 2 of this paper we describe the static and dynamic VRPTW problem and existing solvers for both problems from literature. Section 3 provides a detailed description of the novel Ant-based DVRPTW solver. Section 4 deals with the benchmark and performance studies for different variations of the algorithms. Finally, Section 5 concludes the work with a summarizing discussion.

2 Problem Description and Related Work

An important part of the ant algorithm is the way in which a tour is represented. Each tour should visit the depot more than once, which represents the start of a new vehicle. To accomplish this the depot node is copied a number of times and all copies can be visited individually. Hence we get n customer nodes with a demand and time window and up to n depot nodes(duplicates). This is used in the problem definition:

Problem 1. Capacitated Vehicle Routing Problem with Time Windows(VRPTW). Let $V = \{v_1, \ldots, v_{2n}\}$ define a set of n customer nodes and up to n depot nodes (duplicates). Let Q denote the maximal capacity of each vehicle, and q_i denote the demand and $[e_i, l_i]$ the time window, and s_i the service time at node v_i, if any. A solution is a tuple $(\pi_1, ..., \pi_x)$ with a maximal length of $2n$ and of which n nodes refer to customers and up to n nodes are identical with the depot. Each cycle that returns to the depot is a single tour (vehicle). In addition, $d_{i,j}$ is the traveling distance from node i to node j. The goal is to minimize the traveling distance

$$\sum_{i=1}^{x-1}(d_{\pi_i,\pi_{i+1}}) + d_{\pi_x,\pi_1} \tag{1}$$

and minimize, with priority, the number of vehicles needed.

To be able to solve a **dynamic problem** we first have to simulate a form of dynamicity. Kilby, Prosser and Shaw [10] have described a method to do this, which is also used by Montemanni et al. [11]. The notion of a working day of T_{wd} seconds is introduced, which will be simulated by the algorithm. Not all nodes are available to the algorithm at the beginning. A subset of all nodes are given an *available time* at which they will become available. This percentage determines the degree of dynamicity of the problem. At the beginning of the day a tentative tour is created with a-priori available nodes. The working day is divided into n_{ts} time slices of length T_{wd}/n_{ts}, also notated with t_{ts}. At each time slice the solution is updated. This allows us to split up the dynamic problem into n_{ts} static problems, which can be solved consecutively. A different approach would be to restart the algorithm every time a node becomes available. This could have a very disruptive effect on the algorithm because it could be stopped before a good solution is found. Note, that the goal is similar than stated in Problem 1, except that some customers and their time windows are unknown a priori and parts of solutions might already have been committed.

In general VRP and VRPTW are considered to be intractable problems, because they generalize the NP complete traveling salesperson problem. Heuristic algorithms for static VRP problems include deterministic [2] and bio-inspired ant-based methods [1,3].

Ant-based methods were first proposed with the Ant System method [3] and simulate a population of ants which use pheromones to communicate with each other and collectively are able to solve complex path-finding problems – a phenomenon called *stigmergy*. For the VRPTW, an ant-based method was proposed by [7]. The paradigm of ant algorithms fits well to dynamic problems [9] including TSP [5] and special types of VRP, where vehicles do not have to return to the depot [11]. However, they have not been applied yet to DVRPTW. Existing work is restricted to tabu search [8], where, as opposed to MACS-VRPTW soft time windows are used.

3 Dynamic Ant Algorithm

The plan is to extend the state-of-the-art ant algorithm for VRPTW to the dynamical case. To our best knowledge, [7] is the only ant algorithm for the VRPTW with a description that allows to reproduce results, and it shows a good performance on standard benchmark problems by Solomon `http://web.cba.neu.edu/ msolomon/heuristi.htm` Due to space limitations, we will directly describe the new dynamic version of this algorithm and indicate changes.

The **controller** is the central part that reads the benchmark data, initializes data structures, builds an initial solution and starts the colonies. The **nearest neighbor heuristic** [6] is used to find initial solutions for the entire algorithm and the ACS-VEI colony, but it was adjusted in two ways. First the constraints on time windows and capacity are checked to make sure no infeasible tours are created. Besides that a limit on the number of vehicles is passed to the function. Because of these limitations it is not always possible to return a tour that incorporates all nodes. In that case a tour with less nodes is returned.Only nodes that are available at $t = 0$ are considered. After initialization, a timer is started that keeps track of t, the used CPU time in seconds. At the start of each time slice the controller checks if any nodes became available during the last time slice. If so, these nodes are inserted using the **InsertMissingNodes** method to make T^* feasible again. Then all necessary nodes are committed. If v_i is the last committed node of a vehicle in the tentative solution, v_j is the next node and t_{ij} is travel time from node v_i to node v_j, the v_j is committed if $e_j - t_{ij} < t + t_{ts}$. When the necessary commitments have been made two ant colony systems (ACS) are started. If a new time slice starts, the colonies are stopped and the controller repeats its loop. A pseudo-code of the controller can be seen in Algorithm 1.

ACS contains two colonies, each one of the which tries to improve on a different objective of the problem. The ACS-VEI colony searches for a solution that uses less vehicles than T^*. The ACS-TIME colony searches for a solution with a smaller traveling distance than T^* while using at most as many vehicles. The two objectives have a fixed priority: a solution with less vehicles is always preferred over a solution with a smaller distance.

There are a few differences between the two colonies. ACS-VEI keeps track of the best solution found by the colony (T^{VEI}), which does not necessarily

Algorithm 1. Controller

```
1: Set time $t = 0$; Set available nodes $n$
2: $T^* \leftarrow$ NearestNeighbor($n$); $\tau_0 \leftarrow 1/(n \cdot$ length of $T^*)$;
3: Start measuring CPU time $t$
4: Start ACS-TIME(vehicles in $T^*$) in new thread
5: Start ACS-VEI(vehicles in $T^* - 1$) in new thread
6: repeat
7:     while colonies are active and time step is not over do
8:         Wait until a solution $T$ is found
9:         if vehicles in $T$ < vehicles in $T^*$ then
10:            Stop threads
11:        $T^* \leftarrow T$
12:    if time-step is over then
13:        if new nodes are available or new part of $T^*$ will be defined then
14:            Stop threads
15:            Update available nodes $n$
16:            Insert new nodes into $T^*$
17:            Commit to nodes in $T^*$
18:    if colonies have been stopped then
19:        Start ACS-TIME(vehicles in $T^*$) in new thread
20:        Start ACS-VEI(vehicles in $T^* - 1$) in new thread
21: until $t \geq T_{wd}$
22: return $T^*$
```

incorporate all nodes. As T^{VEI} also contributes to the pheromone trails it helps ACS-VEI to find a solution that covers all nodes with less vehicles. ACS-TIME does not work with infeasible solutions and does not have a colony-best solution. Unlike ACS-VEI, it performs a local search method called **Cross Exchange** [15] shown in Figure 1. The maximum number of vehicles that may be used is given as an argument to each colony. During the construction of a tour this number may not be exceeded. This may lead to infeasible solutions that do not incorporate all nodes. If a solution is not feasible it can never be send to the controller. Both colonies work on separate pheromone matrices and send their best solutions to the controller. Pseudo-codes for ACS-VEI and ACS-TIME can be found in Algorithm 2 and 3 respectively.

Algorithm 4 describes the **construction of a tour** by means of artificial ants. A tour starts at a randomly chosen depot copy and is then iteratively extended with available nodes. The set $\mathcal{N}_i^k$ contains all available nodes which have not been committed for ant k situated at node i. Committed parts of T^* have to be incorporated in every tour. Inaccessible nodes due to capacity or time window constraints are excluded from $\mathcal{N}_i^k$. In order to decide which node to chose, the probabilistic transition rules by Dorigo and Gambardella [4] are applied. For ant k positioned at node v_i, the probability $p_j^k(v_i)$ of choosing v_j as its next node is given by the following transition rule:

Algorithm 2. ACS-VEI(v)

```
1: Input: $v$ is the maximum number of vehicles to be used
2: Given: $\tau_0$ is the initial pheromone level
3:
4: Initialize pheromones to $\tau_0$
5: Initialize IN to 0
6: $T^{\text{VEI}} \leftarrow$ NearestNeighbor(v)
7:
8: repeat
9:     for each ant $k$ do
10:        $T^k \leftarrow$ ConstructTour($k$, IN)
11:        for each nodes $i \notin T^k$ do
12:            $\text{IN}_i = \text{IN}_i + 1$
13:        Local pheromone update on edges of $T^k$ using Equation 3
14:        $T^k \leftarrow$ InsertMissingNodes($k$)
15:
16:    Find ant $l$ with most visited nodes
17:    if nodes in $T^l >$ nodes in $T^{\text{VEI}}$ then
18:        $T^{\text{VEI}} \leftarrow T^l$
19:        Reset IN to 0
20:        if $T^{\text{VEI}}$ is feasible then
21:            return $T^{\text{VEI}}$ to controller
22:
23:    Global pheromone update with $T^*$ and Equation 4
24:    Global pheromone update with $T^{\text{VEI}}$ and Equation 4
25: until controller sends stop signal
```

Algorithm 3. ACS-TIME(v)

```
1: Input: $v$ is the maximum number of vehicles to be used
2: Given: $\tau_0$ is the initial pheromone level
3:
4: Initialize pheromones to $\tau_0$
5:
6: repeat
7:     for each ant $k$ do
8:         $T^k \leftarrow$ ConstructTour($k$, 0)
9:         Local pheromone update on edges of $T^k$ using Equation 3
10:        $T^k \leftarrow$ InsertMissingNodes($k$)
11:        if $T^k$ is a feasible tour then
12:            $T^k \leftarrow$ LocalSearch($k$)
13:
14:    Find feasible ant $l$ with smallest tour length
15:    if length of $T^l <$ length of $T^*$ then
16:        return $T^l$ to controller
17:
18:    Global pheromone update with $T^*$ and Equation 4
19: until controller sends stop signal
```

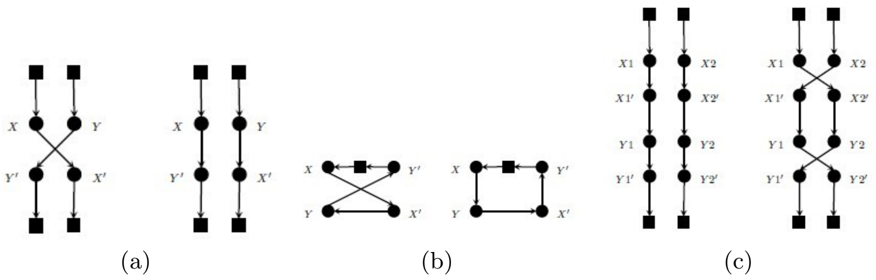

Fig. 1. Examples of 2-opt edge replacements. Squares represent depots, circles represent nodes. (a) demonstrates a move with edges from different tours. (b) is an example of a move within a single tour.(c) shows the process of cross exchange.

$$p_j^k(v_i) = \begin{cases} \underset{j \in N_i}{\arg\max}\{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta\} & \text{if } q \leq q_0 \text{ and } j \in N_i^k \\ \dfrac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{m \in N_i^k} [\tau_{im}]^\alpha \cdot [\eta_{im}]^\beta} & \text{if } q > q_0 \text{ and } j \in N_i^k \\ 0 & \text{if } j \notin N_i^k \end{cases} \tag{2}$$

with τ_{ij} being the pheromone level on edge (i, j), η_{ij} the heuristic desirability of edge (i, j), α the influence of τ on the probabilistic value, β the influence of η on the probabilistic value, N_i^k the set of nodes that can be visited by ant k positioned at node v_i, and $\tau_{ij}, \eta_{ij}, \alpha, \beta \geq 0$. Moreover q denotes a random number between 0 and 1 and $q_0 \in [0, 1]$ a threshold.

During the ConstructTour process of ACS-VEI, the IN array is used to give greater priority to nodes that are not included in previously generated tours. The array counts the successive number of times that node v_j was not incorporated in constructed solutions. This count is then used to increase the attractiveness η_{ij}. The IN array is only available to ACS-VEI and is reset when the colony is restarted or when it finds a solution that improves T^{VEI}. ACS-TIME does not use the IN array, which is equal to setting all values in the array to zero.

The local pheromone update rule from [4] is used to decrease pheromone levels on edges that are traversed by ants. Each time an ant has traversed an edge (i, j), it applies Equation 3. By decreasing pheromones on edges that are already traveled on, there is a bigger chance that other ants will use different edges. This increases exploration and should avoid too early stagnation of the search.

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0 \tag{3}$$

The global pheromone update rule is given in Equations 4. To increase exploitation, pheromones are only evaporated and deposited on edges that belong to the best solution found so far and $\Delta\tau_{ij}$ is multiplied by the pheromone decay parameter ρ.

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \sum_{k=1}^{m} \Delta\tau_{ij}^k \ , \quad \forall (i, j) \in T^* \text{ and } \Delta\tau_{ij}^k = 1/L^* \tag{4}$$

where T^* is the best tour found so far and L^* is the length of T^*.

Algorithm 4. ConstructTour(k, IN)

```
1: Input: $k$ is the ant we construct a tour for
2: Input: IN is an array containing the number of times nodes have not been incorporated in tours
3: Given: $\mathcal{N}_i^k$ is a set of nodes and depot duplicates that are reachable by ant $k$ in node $i$
4:
5: Current vehicle $x \leftarrow 0$
6: Select a random depot duplicate $i$
7: $T^k \leftarrow \langle i \rangle$                                    ▷ Add vehicle $i$ to tour $k$
8: current time$_k \leftarrow 0$
9: load$_k \leftarrow 0$
10: for each committed node $v_i$ of the $x^{th}$ vehicle of $T^*$ do
11:     $T^k \leftarrow \langle i \rangle$
12:     current time$_k \leftarrow$ delivery time$_i$ + service time$_i$
13:     load$_k \leftarrow$ load$_k + q_i$
14:
15: repeat
16:     for each $j \in \mathcal{N}_i^k$ do                    ▷ The part below is taken from [4]
17:         delivery time$_j \leftarrow \max($current time$_k + t_{ij}, e_j)$
18:         delta time$_{ij} \leftarrow$ delivery time$_j -$ current time$_k$
19:         distance$_{ij} \leftarrow$ delta time$_{ij} \times (l_j -$ current time$_k)$
20:         distance$_{ij} \leftarrow \max(1.0,$ (distance$_{ij} -$ IN$_j$))
21:         $\eta_{ij} \leftarrow 1.0/$ distance$_{ij}$
22:
23:     Pick node $j$ using Equation 2
24:     $T^k \leftarrow T^k + \langle j \rangle$
25:     current time$_k \leftarrow$ delivery time$_j +$ service time$_j$
26:     load$_k \leftarrow$ load$_k + q_j$
27:     if $j$ is a depot copy then
28:         current time$_k \leftarrow 0$
29:         load$_k \leftarrow 0$
30:         $x \leftarrow x + 1$
31:         for each committed node $v_i$ of the $x^{th}$ vehicle of $T^*$ do
32:             $T^k \leftarrow \langle i \rangle$
33:             current time$_k \leftarrow$ delivery time$_i$ + service time$_i$
34:             load$_k \leftarrow$ load$_k + q_i$
35:     $i \leftarrow j$
36: until $\mathcal{N}_i^k = \{\}$
37:
38: return $T^k$
```

4 Benchmark and Results

MACS-VRPTW and its extension MACS-DVRPTW was tested on the 56 benchmark problems by Solomon [13]. These problems are divided into six categories: C1, C2, R1, R2, RC1 and RC2. The C stands for problems with clustered nodes, the R problems have randomly placed nodes and RC problems have both. Problems of type 1 have a short scheduling horizon, only a few nodes can be serviced by a single vehicle. Problems of type 2 have a long scheduling horizon.

To simulate dynamics we modified the VRPTW problems by Solomon. A certain percentage of nodes is only revealed during the working day. A dynamicity of $X\%$ means that each node has a probability of $X\%$ to get a non-zero available time. Time intervals are assigned randomly within the available times using a method by Gendreau et al. [8]. Available time are generated on the interval $[0, \overline{e_i}]$, where $\overline{e_i} = \min(e_i, t_{i-1})$. Here, t_{i-1} is the departure time from v_i's predecessor in the best known solution. These best solutions are taken from the results of our MACS-VRPTW implementation (see table 1) – for the solutions we refer to the support material available on `http://natcomp.liacs.nl/index.php?page=code`. By generating available times on this interval, optimal solution can still be attained, enabling comparisons with MACS-VRPTW. For DVRPTW the Solomon problems in Table 1 were computed 10 runs of MACS-DVRPTW. Our implementation was executed on a Intel Core i5, 3.2GHz CPU with 4GB of RAM memory. The controller stops after 100 seconds of CPU time. We set the following default parameters according to literature: $m = 10$, $\alpha = 1$, $\beta = 1$, $q_0 = 0.9$, $\rho = 0.1$ (cf. [7]), $T_{wd} = 100$, and $n_{ts} = 50$ (cf. [11]). Four variants of the algorithms were

Table 1. Comparison of results reported for the original MACS-VRPTW [7] and our implementation for the Solomon benchmark

	C1		**C2**		**R1**		**R2**		**RC1**		**RC2**	
	Dist	**Vei**	**Dist**	**Vei**	**Dist**	**Vei**	**Dist**	**Vei**	**Dist**	**Vei**	**Dist**	**Vei**
Original	828.40	10.00	593.19	3.00	1214.80	12.55	971.97	3.05	1395.47	12.46	1191.87	3.38
Avg	828.67	10.00	591.00	3.00	1226.05	12.52	992.49	3.00	1381.20	12.25	1165.51	3.35
Best	828.37	10.00	589.85	3.00	1216.70	12.33	949.69	3.00	1362.58	12.00	1146.89	3.25

Table 2. Average results and Standard Deviations (Stdev) for 10 runs and 56 Problems of different MACS-DVRPTW variants and dynamicity levels (Dyn).

	Normal			**IIS**			**WPP**			**MMAS**		
Dyn	**Vei**	**Dist**	**Stdev**	**Vei**	**Dist**	**Stdev**	**Vei**	**Dist**	**Stdev**	**Vei**	**Dist**	**Stdev**
0%	7.39	1046.06	21.72	7.35	1035.86	20.14	7.35	1043.13	20.22	7.40	1050.06	22.29
10%	7.91	1095.10	28.95	7.93	1087.06	28.39	7.93	1087.98	26.11	7.95	1093.66	31.66
20%	8.37	1131.47	29.59	8.38	1131.41	31.13	8.39	1127.67	26.52	8.43	1133.99	36.00
30%	8.79	1180.36	34.84	8.78	1177.96	34.37	8.79	1175.14	35.32	8.88	1183.02	34.59
40%	9.03	1216.72	36.73	9.02	1212.11	37.12	9.04	1210.38	37.80	9.08	1212.48	39.64
50%	9.32	1241.32	38.09	9.36	1236.36	39.64	9.34	1235.90	38.52	9.34	1235.90	39.06

tested: (1) default settings as described above, (2) spending 20 CPU seconds before the starting of the working day to construct an improved initial solution (IIS), (3) with pheromone preservation (WPP)[11] ($\tau_{ij} = \tau_{ij}^{old}(1-\rho) + \rho\tau_0$), $\rho = 0.3$, and (4) min-max pheromone update [14]. For MMAS, we set $\rho = 0.8$. The values used are: $\tau_{max} = 1/(\rho T^*)$,$\tau_{min} = \tau_{max}/(2 \cdot \#\text{AvailableNodes})$, $\tau_0 = \tau_{max}$. These are updated every time a new improvement of T^* is found.

Average results for IIS and MMAS are almost identical to the original results. The reason for this seems to be that although the initial solution is greatly improved, it is more difficult to insert new nodes into the current best solution. Figure 2 shows results for different types of problems in more detail. WPP improves distance results for 10% dynamicity and MMAS for 50% dynamicity, both for the price of slightly more vehicles. Another finding is that for 10% dynamicity solution quality declines by up to 20% and for 50% by up to 50%.

10%	C1		C2		R1		R2		RC1		RC2	
	Dist	Vei	Dist	Vei	Dist	Vei	Dist	Vei	Dist	Vei	Dist	Vei
Static VRP	828.67	10	591	3	1226	12.52	992.49	3	1381.2	12.25	1165.51	3.35
DVRP,no wpp, no IIS	944.1	**10.85**	632.8	**3.67**	1283	**13.1**	1038.1	**3.52**	1450.76	**12.75**	1222.05	3.61
DVRP,0.3 wpp no IIS	947.04	10.87	629.2	3.67	1270	13.17	1023.4	3.55	**1438.17**	12.8	1219.73	3.56
DVRP,no wpp, IIS	**943.1**	10.88	**628.28**	3.68	**1268**	13.19	1022.65	3.54	1446.8	12.8	**1213.7**	**3.51**
DVRP, MMAS	954.55	10.87	632.31	3.68	1283	13.25	**1013.8**	3.54	1458.08	12.82	1219.99	3.57
Decline [%]	13.8089	8.5	6.308	22.3	3.409	4.633	2.14712	17.3	4.12467	4.082	4.13467	4.78

50%	C1		C2		R1		R2		RC1		RC2	
	Dist	Vei	Dist	Vei	Dist	Vei	Dist	Vei	Dist	Vei	Dist	Vei
Static VRP	828.67	10	591	3	1226	12.52	992.49	3	1381.2	12.25	1165.51	3.35
DVRP,no wpp, no IIS	1175.86	**12.31**	756.48	4.92	1367	14.33	1146.55	4.53	1581.72	14.26	1420.15	**5.6**
DVRP,0.3 wpp no IIS	**1166.81**	12.46	761.6	4.96	**1361**	**14.25**	1138.83	4.5	**1571.06**	**14.21**	1415.77	5.7
DVRP,no wpp, IIS	1167.09	12.48	751.26	4.91	1365	14.35	1145.02	**4.46**	1580.63	14.23	1409.61	5.73
DVRP,MMAS	1179.03	12.4	**740.36**	**4.87**	1378	14.42	**1111.33**	4.62	1586.22	14.37	**1386.35**	5.78
Decline [%]	40.8051	23.1	25.272	62.3	11.04	13.82	11.9739	48.7	13.746	16	18.9479	67.2

Fig. 2. Averaged results of 6 Solomon categories using different variants in 10% and 50% dynamicity. The yellow mark is for the best for each problem. Also the decline of solution quality for the dynamic problem as compared to the static problem is reported based on the best DVRP result.

5 Conclusion and Outlook

The MACS-DVRPTW is proposed as a first ant-based solver for the DVRPTW problem. Our results show that results with smaller than 20% of the original solution quality can be achieved for small dynamicity and 50% decline for 50% dynamic insertions. Different variants were tested and pheromone preservation and min-max improved distances, for the price of using on average slightly more vehicles. Future work will have to deepen the study of the conflict between these two objectives, and extend the algorithm with deletion of nodes and dynamically changing travelling times.

Support material (`C`-implementation of MACS-DVRPTW, benchmark, and best solutions) is available at `http://natcomp.liacs.nl/index.php?page=code`

Acknowledgement. The authors gratefully acknowledge financial support by Agentschap NL, The Netherlands within the project ' Deliver'. Zhiwei Yang acknowledges financial support from CSC.

References

1. Bell, J.E., McMullen, P.R.: Ant colony optimization techniques for the vehicle routing problem. Advanced Engineering Informatics 18(1), 41–48 (2004)
2. Clarke, G., Wright, J.W.: Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12(4), 568–581 (1964)
3. Colorni, A., Dorigo, M., Maniezzo, V.: Distributed optimization by ant colonies. In: Varela, F., Bourgine, P. (eds.) Proceedings of the First European Conference on Artificial Life, Paris, France, pp. 134–142. Elsevier (1991)
4. Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1(1), 53–66 (1997)
5. Eyckelhof, C.J., Snoek, M.: Ant systems for a dynamic TSP. In: Dorigo, M., Di Caro, G.A., Sampels, M. (eds.) ANTS 2002. LNCS, vol. 2463, pp. 88–99. Springer, Heidelberg (2002)
6. Flood, M.M.: The traveling-salesman problem. Operations Research 4(1), 61 (1956)
7. Gambardella, L.M., Taillard, É., Agazzi, G.: MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In: New Ideas in Optimization, ch. 5, pp. 63–76. McGraw-Hill (1999)
8. Gendreau, M., Guertin, F., Potvin, J.-Y., Taillard, É.: Parallel tabu search for real-time vehicle routing and dispatching. Transportation Science 33(4), 381 (1999)
9. Guntsch, M., Middendorf, M.: Applying population based ACO to dynamic optimization problems. In: Dorigo, M., Di Caro, G.A., Sampels, M. (eds.) ANTS 2002. LNCS, vol. 2463, pp. 111–122. Springer, Heidelberg (2002)
10. Kilby, P., Prosser, P., Shaw, P.: Dynamic VRPs: a study of scenarios. Technical Report APES-06-1998, University of Strathclyde (September 1998)
11. Montemanni, R., Gambardella, L.M., Rizzoli, A.E., Donati, A.V.: Ant colony system for a dynamic vehicle routing problem. Journal of Combinatorial Optimization 10, 327–343 (2005)
12. Psaraftis, H.N.: Dynamic vehicle routing: Status and prospects. Annals of Operations Research 61, 143–164 (1995)
13. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35(2), 254–265 (1987)
14. Stützle, T., Hoos, H.: Max-min ant system and local search for the traveling salesman problem. In: IEEE International Conference on Evolutionary Computation, pp. 309–314 (April 1997)
15. Taillard, É., Badeau, P., Gendreau, M., Guertin, F., Potvin, J.-Y.: A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science 31(2), 170 (1997)