4

Ant-based Solver for Dynamic Vehicle Routing Problem with Time Windows

4.1 · Introduction

The vehicle routing problem (VRP) is a combinatorial optimization problem which has been studied for a long time in the literature, such as [Bianchi et al., 2009, Marinakis et al., 2010, Xiao et al., 2012, Pillac et al., 2013] and [Yang et al., 2015a]. The aim of this problem is to deliver orders from a depot to customers using a fleet of vehicles. Here we look at a practically important variant of this problem where new events (demands, orders) are dynamically introduced during operation time and cars have to serve customers at times within given time windows. So far the problems of dynamical events and time windows have only been looked at in isolation, but in this chapter we will propose and analyze an algorithm that can deal with dynamicity and time windows.

Since the VRP problem already in its most basic variant is NP hard, it seems unlikely that efficient exact solvers for larger instances can be built and one has to rely on heuristics and meta-heuristics for finding good solutions. Among these heuristic methods, problem specific heuristics, including savings heuristic, local search meta-heuristics, and approaches from natural computing such as ant colony optimization are common. Yet, the most powerful solvers today combine several of these methods and could be termed *hybrid solvers*.

In this chapter a hybrid solver is developed. In the global search architecture it uses an ant colony optimization system, whereas in its initialization and search operators

it uses problem specific construction and local search methods. More specifically, the Multi Ant Colony System (MACS) is introduced to solve the real-world dynamic vehicle routing problem. MACS was first proposed by [Gambardella et al., 1999a] who used two ant colonies to search the best solution for the vehicle routing problem in order to improve the performance of ant colonies. In this algorithm, the first colony minimizes the number of vehicles while the second one minimizes the travel cost. [van Veen et al., 2013] generate a dynamic vehicle routing problem with time windows (DVRPTW) benchmark based on the static Solomon benchmark and adjust the MACS to this dynamic problem. This chapter extends upon this chapter by providing a more in-depth discussion and motivation of the approach and benchmark designs. More importantly, we add results from a real-world pilot study provided by a Dutch mobile surveillance company.

This chapter is organized as follows: Related work is summarized in Section 4.2. Section 4.3 describes the MACS algorithm and how it is adapted to the dynamical vehicle routing problem with time windows. Section 4.4 introduces a benchmark for this problem class and describes the performance of the algorithm on the benchmark and also includes results on static benchmarks for validation. Finally, Section 4.5 summarizes the work of this chapter and suggests directions for relevant future research.

4.2 · Related Work

In general VRP and VRPTW are NP hard problems and they generalize the NP complete traveling salesman problem. Therefore heuristic algorithms are widely used in order to solve the vehicle routing problem. Classical examples are the *nearest neighbor heuristic* by [Flood, 1956] and the *savings algorithm* that was developed by [Clarke and Wright, 1964] based on the savings concept which repeatedly combines two customers on the same route. Early advances were achieved by [Shaw, 1998] using large neighborhood search.

Nowadays, the use of metaheuristics becomes more and more popular. [Semet and Taillard, 1993] presented a tabu search for finding a good solution for the vehicle routing problem. [Baker and Ayechew, 2003] combined the genetic algorithm and neighborhood search methods which can give good results for this problem. [Gambardella et al., 1999a] introduced Ant Colony Optimization which uses artificial ant colonies to construct a shortest route.

In contrast to a large multitude of available static VRP solvers, there are only a

few algorithms which can tackle dynamic VRPs. In principle, most of the algorithms described above can be adapted to solve the dynamic VRPs. But in order to deal efficiently with the dynamics of this problem, the algorithm should also have some mechanisms that promote reusing learned features of the problem from previous solutions. As indicated in [Eyckelhof and Snoek, 2002], some bio-mimetic ant-colony optimization algorithm seems to support dynamic adaptations of delivery routes well. For instance, in Ant Colony Optimization virtual pheromone trails are created to indicate good directions if solutions only need to be changed partially. Moreover, ACO can be easily combined with local search heuristics and route construction algorithms. The flexibility of ACO and its good performance in static vehicle routing problem make it an attractive paradigm for the dynamic vehicle routing problem.

Ant-based methods 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 problem, an ant-based method was proposed by [Gambardella et al., 1999a]. They showed that good results can be achieved by running one ant colony for optimizing the number of vehicles and one ant colony for minimizing route cost and term their method multiple ant colony system (MACS). The paradigm of ant algorithms fits well to dynamic problems in [Guntsch and Middendorf, 2002] including TSP in [Eyckelhof and Snoek, 2002] and special types of VRP problem, where vehicles do not have to return to the depot which can be seen in [Montemanni et al., 2005]. In this chapter we will extend multi ant colony optimization to problems *with time windows* and we will call our new method MACS-DVRPTW.

There exist some previous studies on using meta-heuristics other than ant colony algorithms on DVRPTW. [Gendreau et al., 1999] propose to use tabu search, but, as opposed to standard benchmarks for MACS-VRPTW, developed their approach for problems with time windows. Lund et al. defined a metric called the *degree of dynamism* to measure the dynamicity which is the ratio between the number of dynamic orders and the total orders [Lund et al., 1996]. However, there is no difference no matter the dynamic orders are given early or late during the execution. Larsen et al. extended the degree of dynamism to a metric called the *effective degree of dynamism* which denotes an average of how late the dynamic orders are released compared to the latest arrival time the dynamic orders could be released [Larsen, 2000]. However, the latest arrival times of dynamic orders are usually not available in real-world problems. In this

chapter, we use the *degree of dynamism* as the measurement of the dynamicity for the dynamic vehicle routing problem with time windows. Moreover, the available times are defined based on the service time of the best known solution of the static vehicle routing problem with time windows and the earliest arrival time of the dynamic orders in Section 4.4. By doing so, we can guarantee that the best known solution could always be obtained in the dynamic version of the vehicle routing problem with time windows.

4.3 · Algorithm

In order to solve this problem, it is natural to extend the state-of-the-art ant algorithm for VRPTW to the dynamical case. To our best knowledge, the multi-colony approach described in [Gambardella et al., 1999a] is the best ant algorithm for the VRPTW with a description that allows to reproduce results, and it shows a good performance on the standard benchmark problems by Solomon. Here we will directly describe our new dynamic version of this algorithm and indicate changes.

The central part of the algorithm is the *controller*. It reads the benchmark data, initializes the data structures, builds an initial solution and starts the ACS-TIME colony and ACS-VEI colony. The ACS-TIME colony tries to minimize traveling cost given a fixed number of vehicles, the ACS-VEI colony seeks to minimize the number of vehicles. Priority of the algorithm is on reducing the number of vehicles. Given solutions with the same number of vehicles, those solutions are preferred that use less time. The ACS-VEI colony restarts the ACS-TIME colony whenever a solution is found that can serve the demand with a smaller number of vehicles.

The *nearest neighbor heuristic* in [Flood, 1956] is used to find initial solutions of vehicle routing problems. However, for the VRPs with time windows it is difficult to get a feasible solution by using this method. Hence we adjusted the algorithms to make them fit to the new problem. One adjustment is checking the time windows constraints before adding the nearest neighbor customer in the tour. By doing so, the infeasible solutions with time windows constraint violations could be avoided. Then, there is a limitation of the number of vehicles which is logical, since in reality companies can only hire a certain number of vehicles. Therefore, a more appropriate algorithm is needed to generate the initial solutions. Moreover, because the constraints of time windows and the number of vehicles cannot be violated, it is not always possible to return a tour that consists of all customers. Then, the tour which incorporates more customers will be adopted.

The new *Ranking Time Windows Based Nearest Neighbor Algorithm* is proposed to generate an initial solution for the DVRPTW. By adding the sorted earliest arrival time of the orders to the m tours one by one, this algorithm can take the time windows and vehicles number constraints in advance. This way there is a higher chance to get a feasible solution with better fitness value. Algorithm 2 describes the initialization. It proceeds as follows: Firstly, the list of customers is sorted by increasing values of earliest arrival times. Then, m tours are created, each of which corresponds to one vehicle. Then, for each customer node the algorithm finds the tour with smallest distance among all those tours to which the node can be inserted without violating constraints. Following this procedure, the nodes are iteratively added in the node list. Finally, the solution is returned.

Algorithm 2 Initialization algorithm

Let V denote the set of n customers. Sort them by increasing values of the earliest arrival times e_i. If the nodes have the same value of e_i, arrange them by increasing values of the latest arrival times l_i.
Let T denote the list of tours, where m is the number of vehicles. Initially, each tour t_j has only a single node which is the depot.
$i \leftarrow 0$
while i is smaller than n **do**
    $TabuList \leftarrow \emptyset$;
    while node v_i is not added to a tour **do**
        for $j \in \{1, \ldots, m\} \setminus TabuList$ **do**
            Calculate the distances d_{ij} between v_i and node $t_{j,end}$,
              where $t_{j,end}$ denotes the last node of tour t_j.
        end for
        Find the index (= $minIndex$) of the tour that has the shortest distance to v_i:
          $minIndex := \arg\min_{j \in \{1,\ldots,m\} \setminus TabuList} \{distance(v_i, t_{j,end})\}$.
        if node v_i can be added to tour $minIndex$ **then**
            Add node v_i to the end of tour $minIndex$.
        else
            $TabuList \leftarrow TabuList \cup \{j\}$.
        end if
    end while
    $i \leftarrow i + 1$.
end while
return T

After initialization, a timer is started that keeps track of the current time ct, the used CPU time in seconds. Then the algorithm will run in online mode during the working day T_{wd}. Let T^* denote the currently optimal solution. Then, at the start of each time slice the controller checks if any new customer node(s) became available during the last time slice. If so, the new node(s) is inserted using the *InsertMissingNodes* method, in order to update T^*. Thereafter, some of the nodes are changed to the status *committed*. The position of committed nodes in the tour cannot be changed anymore. If v_i is the last committed node of a vehicle in the tentative solution, v_j is the next node and tt_{ij} is travel time from node v_i to node v_j, then v_j is committed if $e_j - tt_{ij} < ct + t_{ts}$. Here, t_{ts} is the length of the time slice. When the necessary commitments have been made the two ant colony systems are started. If a new time slice starts, the colonies are stopped and the controller repeats its loop.

The pseudo-code of the controller can be seen in Algorithm 3. ACS contains two colonies, each one of which tries to improve 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 cost than the cost of T^* using at most as many vehicles as the best solution so far, i.e. T^*. A solution with less vehicles has a higher priority than a solution with a smaller traveling cost. Once a feasible solution is found by the ACS-VEI, the controller restarts.

There are a few differences between the two colonies. The ACS-VEI keeps track of the best solution found by the colony (T^{VEI}), which does not necessarily incorporate all nodes. As the T^{VEI} also contributes to the pheromone trails it helps the ACS-VEI to find a solution that covers all nodes with less vehicles. The ACS-VEI does not use local search methods. In contrast, the ACS-TIME does not work with infeasible solutions and it performs a local search method called *Cross Exchange* in [Taillard et al., 1997] which is shown in Figure 4.1.

A constraint on the maximum number of vehicles that can 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 sent to the controller. Both colonies work on separate pheromone matrices and send their best solutions to the controller. Pseudo-codes for the ACS-VEI and the ACS-TIME can be found in Algorithm 4 and 5, respectively.

Algorithm 6 describes the *construction of a tour* by means of artificial ants. First

Algorithm 3 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 ct
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: **end if**
12: $T^* \leftarrow T$
13: **end while**
14: **if** time-step is over **then**
15: **if** new nodes are available or new part of T^* will be defined **then**
16: Stop threads
17: Update available nodes
18: Insert new nodes into T^*
19: Commit necessary nodes in T^*
20: **end if**
21: **end if**
22: **if** colonies have been stopped **then**
23: Start ACS-TIME(vehicles in T^*) in new thread
24: Start ACS-VEI(vehicles in $T^* - 1$) in new thread
25: **end if**
26: **until** $ct \geq T_{wd}$
27: **return** T^*

the number of ant k and the *IN* array are input. Meanwhile, the neighborhood set $\mathcal{N}_i^k$ is given. There are many ways to define the topology structure of neighborhood nodes. In this chapter, the neighborhood nodes are defined as all available nodes that have not been committed and visited yet. The neighborhood nodes set $\mathcal{N}_i^k$ contains all available nodes which have not been committed and visited for ant k situated at node v_i. Inaccessible nodes due to capacity or time window constraints are excluded from $\mathcal{N}_i^k$.

A tour starts at the depot (v_0). When constructing a new tour, the committed parts of T^* which cannot be changed any more have to be incorporated first (Line 9-13). Then,

Algorithm 4 ACS-VEI

Input: m is the number of vehicles to be used
Given: τ_0 is the initial pheromone level

Initialize pheromones to τ_0
Initialize IN_i to 0 for $i = 1, \ldots, N$
Comment: Here IN_i is a counter for how many times
  the customer node i has not been added to the solution.

$T^{\text{VEI}} \leftarrow$ NearestNeighbor(m)
repeat
    for all ants k **do**
        $T^k \leftarrow$ ConstructTour(k, IN)
        for all nodes $i \notin T^k$ **do**
            $IN_i = IN_i + 1$
        end for
        Local pheromone update on edges of T^k using Equation 4.2
        $T^k \leftarrow$ InsertMissingNodes(k)
    end for

    Find ant l with most visited nodes
    if number of nodes in $T^l >$ number of nodes in T^{VEI} **then**
        $T^{\text{VEI}} \leftarrow T^l$
        Reset IN to 0
        if T^{VEI} contains n nodes (meaning it is feasible) **then**
            return T^{VEI} to controller
        end if
    end if

    Global pheromone update with T^* and Equation 4.2
    Global pheromone update with T^{VEI} and Equation 4.2
until controller sends stop signal

Algorithm 5 ACS-TIME

Input: m is the number of vehicles to be used
Given: τ_0 is the initial pheromone level

Initialize pheromones to τ_0
Set IN_i to 0, for $i = 1, \ldots, N$

repeat
    for all ants k **do**
        $T^k \leftarrow$ ConstructTour(k,IN)
        Local pheromone update on edges of T^k using Equation 4.2
        $T^k \leftarrow$ InsertMissingNodes(k)
        if T^k is a feasible tour **then**
            $T^k \leftarrow$ LocalSearch(k)
        end if
    end for

    Find feasible ant l with smallest tour length
    if length of T^l < length of T^* **then**
        $T^* \leftarrow T^l$
        return T^* to controller
    end if

    Global pheromone update with T^* and Equation 4.2
until controller sends stop signal

Algorithm 6 ConstructTour(k, IN)

Input: k is the ant for which we construct a tour
IN is an array containing the number of times that nodes in Algorithm 4 have not been incorporated in tours
Given: $\mathcal{N}_i^k$ is a set of neighboring nodes including the depot duplicates that are reachable by ant k in node v_i

Current vehicle $x \leftarrow 0$
$T^k \leftarrow \langle v_0 \rangle$
current time$_k \leftarrow 0$
load$_k \leftarrow 0$
for all committed nodes v_i of the x^{th} vehicle of T^* **do**
    $T^k \leftarrow \langle v_i \rangle$
    current time$_k \leftarrow$ delivery time$_i$ + service time$_i$
    load$_k \leftarrow$ load$_k + q_i$
end for

repeat
    for all $v_j \in \mathcal{N}_i^k$ **do** ▷ See also [Dorigo and Gambardella, 1997]
        delivery time$_j \leftarrow$ max(current time$_k + t_{ij}, e_j$)
        delta time$_{ij} \leftarrow$ delivery time$_j -$ current time$_k$
        urgency$_{ij} \leftarrow$ delta time$_{ij} \times (l_j -$ current time$_k$)
        urgency$_{ij} \leftarrow$ max(1.0, (urgency$_{ij} - IN_j$))
        $\eta_{ij} \leftarrow$ 1.0/ urgency$_{ij}$
    end for

    Pick node v_j using Equation 4.1
    $T^k \leftarrow \langle v_j \rangle$
    current time$_k \leftarrow$ delivery time$_j$+ service time$_j$
    load$_k \leftarrow$ load$_k + q_j$
    if v_j is a depot **then**
        current time$_k \leftarrow 0$
        load$_k \leftarrow 0$
        $x \leftarrow x + 1$
        for all committed nodes v_i of the x^{th} vehicle of T^* **do**
            $T^k \leftarrow \langle v_i \rangle$
            current time$_k \leftarrow$ delivery time$_i$ + service time$_i$
            load$_k \leftarrow$ load$_k + q_i$
        end for
    end if
    $i \leftarrow j$
until $\mathcal{N}_i^k = \{\}$

return T^k

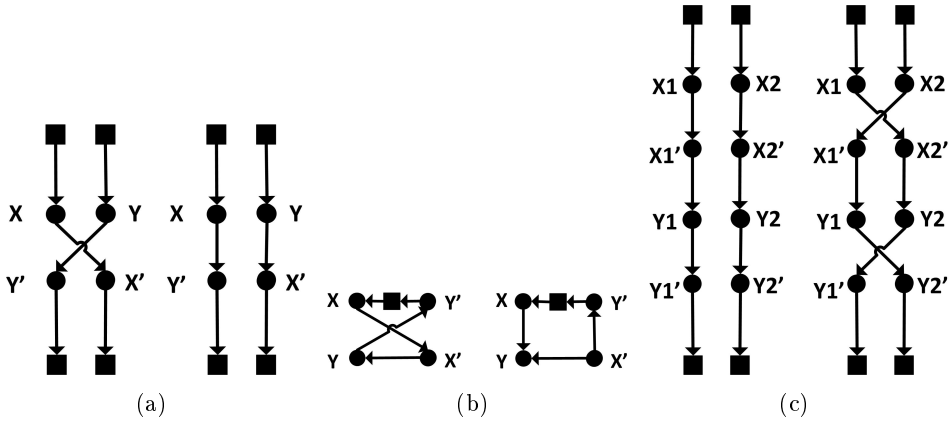

Figure 4.1 Examples of 2-opt edge replacements. Squares represent depots, circles represent nodes. (a) demonstrates a move with edges from different tours. (b) is an example of a move within a single tour.(c) shows the process of cross exchange.

for each iteration, the tour is extended with available neighborhood nodes based on the urgency of nodes and the *IN* array (Line 16-27). The *IN* array in the ACS-VEI is used to give a 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 the constructed solutions. This count and the urgency of the node are then used to increase the attractiveness η_{ij}. The *IN* array is only available to the ACS-VEI and is reset when the colony is restarted or when it finds a solution that improves T^{VEI}. The ACS-TIME does not use the *IN* array, which is equal to setting all values in the array to zero.

In order to decide which node to chose, the probabilistic transition rule by [Dorigo and Gambardella, 1997] 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:

$$p_j^k(v_i) = \begin{cases} \arg\max\limits_{j \in \mathcal{N}_i^k} \{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta\} & \text{if } q \leq q_0 \text{ and } j \in \mathcal{N}_i^k \\ \dfrac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{x \in \mathcal{N}_i^k} [\tau_{ix}]^\alpha \cdot [\eta_{ix}]^\beta} & \text{if } q > q_0 \text{ and } j \in \mathcal{N}_i^k \\ 0 & \text{if } j \notin \mathcal{N}_i^k \end{cases} \tag{4.1}$$

with τ_{ij} being the pheromone level on edge (i, j), η_{ij} denotes the heuristic desirability of

edge (i, j), α represents the influence of τ on the probabilistic value, β is the influence of η on the probabilistic value, $\mathcal{N}_i^k$ is 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 $q_0 \in [0, 1]$ denotes a threshold.

If the new added node is a depot, it means the current vehicle is not able to serve the new node. Hence, the new node has to be assigned to another vehicle (Line 28-37). For the new vehicle, all committed nodes have to be added first (Line 32- 36). This procedure continues until the neighborhood set of the last node in the tour is empty. The new generated tour T^k is then returned.

The local pheromone update rule from [Dorigo and Gambardella, 1997] 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 4.2:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0 \tag{4.2}$$

By decreasing pheromones on edges that are already visited, there is a bigger chance that other ants will use different edges. This increases the exploration and would avoid too early stagnation of the search.

The global pheromone update rule is given in Equation 4.3. To increase the 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.

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij}+\rho\cdot\sum_{k=1}^{\mu}\Delta\tau_{ij}^k\,, (v_i, v_j) \in T^*, i = 0, \ldots, N, j = 0, \ldots, N, \text{ and } \Delta\tau_{ij}^k = 1/L^* \tag{4.3}$$

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

[Gambardella et al., 1999a] has shown that the MACS is very efficient in solving *static* vehicle routing problems with time windows. Here we are going to test and benchmark the extended algorithm for *dynamic* vehicle routing problems with time windows.

4.4 · Benchmark on Simulated Data

The Solomon benchmark is a classical benmark for the static VRP in [Solomon, 1987]. It provides six categories of scalable VRPTW problems: C1, C2, R1, R2, RC1 and

Table 4.1 Comparison of results reported for the original MACS-VRPTW in [Gambardella et al., 1999a] and our implementation for the Solomon benchmark.

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

RC2. The C stands for problems with clustered nodes, the R problems have randomly placed nodes and RC problems have both. In problems of type 1, only a few nodes can be serviced by a single vehicle. But in problems of type 2, there is no restriction on the number of nodes that can be serviced by the same vehicle.

In order to make a dynamic problem benchmark we apply a method proposed by [Gendreau et al., 1999] for a VRP problem, to the more comprehensive benchmark by Solomon on the VRPTW. 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. The available time means the time when the order is revealed. It is generated on the interval $[0, \overline{e_i}]$, where $\overline{e_i} = \min(e_i, td_{i-1})$. Here, td_{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 a static MACS-VRPTW implementation (see Table 4.1) – for the detailed schedules we refer to the support material available on `http://natcomp.liacs.nl/index.php?page=code`. By generating available times on this interval, optimal solutions can still be attained, enabling comparisons with the MACS-VRPTW. Table 4.2 shows the average results and the standard deviation change with the dynamicity levels.

In the experiment 10 runs were executed on a Intel Core i5, 3.2GHz CPU with 4GB of RAM memory. The controller stopped after 100 seconds of CPU time. The

Table 4.2 Average results and Standard Deviations (Stdev) for 10 runs and 56 Problems of different MACS-DVRPTW variants and dynamicity levels (Dyn).

Dyn		0%	10%	20%	30%	40%	50%
Normal	Vei	7.39	7.91	8.37	8.79	9.03	9.32
	Dist	1046.06	1095.1	1131	1180.36	1217	1241.32
	Stdev	21.72	28.95	29.59	34.84	36.73	38.09
IIS	Vei	7.35	7.93	8.38	8.78	9.02	9.36
	Dist	1035.86	1087.06	1131	1177.96	1212	1236.36
	Stdev	20.14	28.39	31.13	34.37	37.12	39.64
WPP	Vei	7.35	7.93	8.39	8.79	9.04	9.34
	Dist	1043.13	1087.98	1128	1175.14	1210	1235.9
	Stdev	20.22	26.11	26.52	35.32	37.80	38.52
MMAS	Vei	7.40	7.95	8.43	8.88	9.08	9.34
	Dist	1050.06	1093.66	1134	1183.02	1212	1235.9
	Stdev	22.29	31.66	36.00	34.59	39.64	39.06

following default parameters were set according to the literature: $\mu = 10$, $\alpha = 1$, $\beta = 1$, $q_0 = 0.9$, $\rho = 0.1$ (cf. [Gambardella et al., 1999a]), $T_{wd} = 100$ seconds, and $n_{ts} = 50$ (cf. [Montemanni et al., 2005]).

To the best of our knowledge, there is no other algorithms which have been implemented to solve this problem. In this chapter, four variants of the algorithm are generated in order to improve the performance of the algorithm. Four variants of the algorithms were as follows: (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) in [Montemanni et al., 2005] ($\tau_{ij} = \tau_{ij}^{old}(1-\rho) + \rho\tau_0$), $\rho = 0.3$, and (4) min-max pheromone update in [Stützle and Hoos, 1997]. For the MMAS, we set $\rho = 0.8$. The values used were: $\tau_{max} = 1/(\rho T^*)$, $\tau_{min} = \tau_{max}/(2 \cdot \text{AvailableNodes})$, $\tau_0 = \tau_{max}$. They were updated every time a new improvement of T^* was found.

Average results for the IIS and the 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. Table 4.3 and Table 4.4 show results for different types of problems in more details. The WPP improves the distance for 10% dynamicity and the MMAS for 50% dynamicity, both for the price of slightly more vehicles. Another finding is that for 10% dynamicity the solution quality

declines by up to 20% and for 50% by up to 50%.

From a practical approach it can be stated that for a small dynamicity of 10% at most 1 additional vehicle is needed as compared to scheduling the same amount of static orders, and in many cases the same number of vehicles suffice. For 50% dynamicity the number of vehicles increases almost always by one vehicle and can in some cases even increase by two vehicles.

4.5 · Summary

The MACS-DVRPTW is proposed as the first ant-based solver for the DVRPTW problem. Our results show that the distance becomes bigger with the increasing dynamicity. Different variants of solvers were tested and the pheromone preservation as well as the min-max pheromone update is able to improve the travel distances. However, they used slightly more vehicles to obtain the improvements. 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.

Table 4.3 Average results of 6 Solomon categories using different variants in 10% dynamicity. The bold font is for the best results of each problem.

10%		Static	DVRP, Default	DVRP, 0.3 WPP	DVRP, IIS	DVRP, MMAS	Decline [%]
C1	Dist	828.67	944.10	947.04	**943.10**	954.55	13.81
	Vei	10.00	10.85	**10.87**	10.88	10.87	8.50
C2	Dist	591.00	632.80	629.20	**628.28**	632.31	6.31
	Vei	3.00	**3.67**	3.67	3.68	3.68	22.33
R1	Dist	1226.05	1282.79	1270.34	**1267.84**	1283.23	3.41
	Vei	12.52	**13.10**	13.17	13.19	13.25	4.63
R2	Dist	992.49	1038.10	1023.40	1022.65	**1013.80**	2.15
	Vei	3.00	**3.52**	3.55	3.54	3.54	17.33
RC1	Dist	1381.20	1450.76	**1438.17**	1446.80	1458.08	4.12
	Vei	12.25	**12.75**	12.80	12.80	12.82	4.08
RC2	Dist	1165.51	1222.05	1219.73	**1213.70**	1219.99	4.13
	Vei	3.35	3.61	3.56	**3.51**	3.57	4.78

Table 4.4 Average results of 6 Solomon categories using different variants in 50% dynamicity. The bold font is for the best results of each problem.

50%		Static	DVRP, Default	DVRP, 0.3 WPP	DVRP, IIS	DVRP, MMAS	Decline [%]
C1	Dist	828.67	1175.86	**1166.81**	1167.09	1179.03	40.81
	Vei	10.00	**12.31**	12.46	12.48	12.40	23.10
C2	Dist	591.00	756.48	761.60	751.26	**740.36**	25.27
	Vei	3.00	4.92	4.96	4.91	**4.87**	62.33
R1	Dist	1226.05	1367.20	**1361.35**	1364.57	1378.01	11.04
	Vei	12.52	14.33	**14.25**	14.35	14.42	13.82
R2	Dist	992.49	1146.55	1138.83	1145.02	**1111.33**	11.97
	Vei	3.00	4.53	4.50	**4.46**	4.62	48.67
RC1	Dist	1381.20	1581.72	**1571.06**	1580.63	1586.22	13.75
	Vei	12.25	14.26	**14.21**	14.23	14.37	16.00
RC2	Dist	1165.51	1420.15	1415.77	1409.61	**1386.35**	18.95
	Vei	3.35	**5.60**	5.70	5.73	5.78	67.16