# A REVISED MULTIPLE ANT COLONY SYSTEM FOR VEHICLE ROUTING PROBLEMS WITH TIME WINDOWS

by

DUYGU TAŞKIRAN

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University
August 2006

**A REVISED MULTIPLE ANT COLONY SYSTEM FOR VEHICLE ROUTING PROBLEMS WITH TIME WINDOWS**

APPROVED BY:

Assistant Prof. Dr. Bülent Çatay …………………………
(Thesis Supervisor )

Assistant Prof. Dr. Hüsnü Yenigün ………………..……

Assistant Prof. Dr. Kemal Kılıç …..……………………

Assistant Prof. Dr. Kerem Bülbül…..………………….

Associate Prof. Dr. Erhan Budak…..…………………..

DATE OF APPROVAL:

A REVISED MULTIPLE ANT COLONY SYSTEM FOR
VEHICLE ROUTING PROBLEMS WITH TIME WINDOWS

Duygu TAŞKIRAN

IE, MS Thesis, 2006

Thesis Supervisor: Assistant Prof. Dr. Bülent ÇATAY

Keywords: Ant colony system, vehicle routing problem with time windows

## ABSTRACT

In this thesis, a Revised Multiple Ant Colony System (RMACS) approach is applied to the Vehicle Routing Problem with Time Windows (VRPTW). Our primary objective is to minimize the number of vehicles and the secondary objective is to minimize the total travel distance. Two artificial ant colonies, where one minimizes the number of vehicles and the other the total travel time, cooperate with each other through pheromone update to optimize the corresponding objectives. The developed approach is coded in C++ and tested on the well-known 56 benchmark instances of Solomon (1987). These instances are composed of six different problem types, each containing 8-12 100-node problems. Although the best solutions could not be improved, in many instances the number of the vehicles is the same with the best results or 1-2 near to them. However, the travel distance %30 far from the best benchmark solutions in some of the problem instances.

ZAMAN KISITLI ARAÇ ROTALAMA PROBLEMİNE FARKLI BİR KARINCA KOLONİSİ SİSTEMİ YAKLAŞIMI

Duygu TAŞKIRAN

IE, Yüksek Lisans Tezi, 2006

Tez Danışmanı: Yrd. Doç. Dr. Bülent ÇATAY

Anahtar Kelimeler: Karınca kolonisi sistemi, zaman kısıtlı araç rotalama problemi

## ÖZET

Bu çalışma, Zaman Kısıtlı Araç Rotalama Problemini Karınca Kolonisi optimizasyonuna dayalı bir yaklaşımla çözmeyi amaçlamaktadır. Problemdeki birinci amacımız araç sayısını, ikinci amacımız ise toplam katedilen yolu minimize etmektir. Bu minimizasyon problemini çözmek üzere biri araç sayısını, diğeri ise toplam katedilen yolu minimize etmeye odaklı iki karınca kolonisi feromen seviyeleri vasıtasıyla haberleşerek bir yardımlaşma anlayışı içerisinde çalışırlar. Algoritma C++ programında kodlanmış olup, Solomon'un (1987) 56 problem örneği üzerinde test edilmiştir. Herbiri 8-12 100 noktalı problem içeren bu problem örnekleri 6 değişik problem setine karşılık gelmektedir. Bu çalışma sonucunda araç sayısında literatürdeki en iyi sonuçlara karşın bir geliştirme sağlanamamış olmasına karşın, en iyi sonuçlara maksimum 2 araç sayısı uzaklıkta sonuçlar bulunmuştur. Fakat katedilen yol miktarı bazı problem örneklerinde literatürdeki en iyi sonuçlardan %30 daha uzak sonuçlar vermektedir.

## ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Assistant Prof Dr. Bülent Çatay for his encouragement, motivation and considerable time he spent from beginning to end of my thesis.

I thank to graduate committee members of my thesis, Assistant Prof. Dr. Kemal Kılıç, Assistant Prof. Dr. Kerem Bülbül, Assistant Prof. Dr. Erhan Budak and Assistant Prof. Dr. Hüsnü Yenigün for their worthwhile suggestions and remarks.

# TABLE OF CONTENTS

**Page**

Abstract ........ IV
Özet ........ V
1. Introduction ........ 1
2. Literature Review ........ 3
3. Ant Colony System ........ 7
4. RMACS for VRPTW ........ 10
    4.1 ACS-TIME and ACS-VEI Colonies ........ 12
5. Solution Constructive Procedure ........ 15
6. Numerical Results ........ 17
7. Conclusions ........ 20
Bibliography ........ 21
Appendix A ........ 23
    Description of the RMACS ........ 23
Appendix B ........ 27
    Detailed solutions comparison (10 ants case) ........ 27
Appendix C ........ 28
    Average of the 5 runs of RMACS vs best known (30 ants case) ........ 28

## LIST OF FIGURES

**Page**

3.1. The ACO heuristics developed by Dorigo Caro (1999) .......... 9
4.1. The MACS-VRPTW procedure .......... 10
4.2. The MACS-VRPTW algorithm .......... 11
4.3 The ACS-TIME procedure .......... 12
4.4. The ACS-VEI procedure .......... 13
4.5 The new_active_ant procedure used by ACS-VEI and ACS-TIME .......... 14

## LIST OF TABLES

**Page**

6.1. Detailed solutions comparison (30 ants case) .......... 18
6.2. Average of the best solutions computed by RMACS and MACS .......... 19

# 1. INTRODUCTION

The Ant System, introduced by Colorni et al. (1991), and Dorigo et al. (1992) with an application on the Traveling Salesman Problem (TSP), is a recent metaheuristic for hard combinatorial optimization problems. Many Ant System algorithms, proven to be very efficient, have been proposed to solve different types of combinatorial optimization problems such as symmetric and asymmetric traveling salesman problems (TSP/ATSP, Dorigo and Gambardella, 1997, Stützle, 1998, Stützle and Dorigo, 1999), the sequential ordering problem (SOP, Gambardella and Dorigo, 1997), the quadratic assignment problem (QAP, Gambardella, Taillard and Dorigo, 1999, Taillard and Gambardella, 1997), the bi-quadratic assignment problem and the p-median problem (Taillard, 1998).

The idea of imitating the behaviour of real ant colonies for solving hard combinatorial optimization problems led to the development of the ant colony algorithms. Real ants communicate with each other via an aromatic essence called 'pheromone' in their search of food, where the quantity of pheromone depends on the quality of the food source. This will consequently make all ants choose the paths leading to rich and nearby food sources as the pheromone trails on these paths will grow faster.

In the Ant System the artificial ants search the solution space to solve combinatorial optimization problems instead of real ants searching their environment to find rich and nearby food sources. These artificial ants cooperate with each other by building solutions in parallel using an indirect form of communication, the pheromone updates. They construct solutions iteratively by adding a new node to the existing partial solution using both the information gained from past and a greedy heuristic called visibility. In this system, the objective function matches with the quality of the food source and an adaptive memory matches with the pheromone trails.

This paper presents a Revised Multiple Ant Colony System (RMACS) application to the Vehicle Routing Problems with Time Windows which is based on Multiple Ant Colony System (MACS) (Gambardella, Taillard, and Agazzi, 1999) approach inspired by the foraging behavior of real colonies of ants.

VRPTW is defined as the problem of minimizing time and costs in case a fleet of vehicles has to distribute goods from a depot to a set of customers. The problem studied in this paper is a hierarchical multi-objective problem; the first objective is to minimize the number of tours (or vehicles) and the second is to minimize the total travel time where the objective of minimization of the number of tours takes precedence over the minimization of the total travel time. The objectives of the VRPTW can be antagonistic in case the problem constraints are very tight. The idea to adapt ACS for these multiple objectives is to define two ACS colonies, each dedicated to the optimization of a different objective function, and cooperates by exchanging information through pheromone updating.

## 2. LITERATURE REVIEW

Capacitated Vehicle Routing Problem (CVRP) is the most basic version of the vehicle routing problems. In the CVRP, $n$ customers, each asking for a quantity $q_i$ of goods, must be served from a unique depot with the limited number of vehicles (v) with capacity $Q$, and with the objective of achieving the minimum total travel time.

From a graph theoretical point of view the CVRP may be stated as follows: Let $G = (C,L)$ be a complete graph with node set $C = (c_o, c_1, c_2,..., c_n)$ and arc set $L = (c_i, c_j)$: $c_i$, $c_j \in C$, $i \neq j$, where $c_o$ is the depot and the other nodes are the customers to be served. Each node is associated with a fixed quantity $q_i$ of goods to be delivered where $q_o = 0$ for the depot, and $t_{ij}$ represents the travel time between $c_i$ and $c_j$ for each arc $(c_i, c_j)$. A solution to the CVRP is a set of tours where each customer is visited exactly once, and each tour starts and ends at the depot. The vehicle has to periodically return to the depot for reloading since the vehicle capacity is limited.

VRPTW is an important extension of the CVRP. In addition to the CVRP characteristics, this problem includes a time window $[b_i, e_i]$ both for the depot and for each customer $c_i$ ($i$ = 0,..., $n$). So the additional constraints to CVRP are that the service beginning time at each node $c_i$ ($i$ = 1,..., $n$) must be greater than or equal to $b_i$, and the arrival time at each node $c_i$ must be lower than or equal to $ei$. Whenever the vehicle reaches the customer before $b_i$, it has to wait until $b_i$ to start the service.

A number of exact and heuristic methods have been proposed for the VRPTW. When the solution space is restricted by narrow time windows so that less combinations of customers are possible to define feasible tours, exact methods are proven to be more efficient.

Dynamic Programming, Lagrangean Relaxation Based Methods and Column Generation principles are used in solving the VRPTW in the context of exact algorithms. Kolen *et al.* (1987) used branch and bound; Jörnsten *et al.* (1986), Madsen *et al.* (1988) and Halse (1992) proposed variable splitting followed by

Lagrangean decomposition, Fisher *et al.* (1997) adopted K-*tree* approach followed by Lagrangean Relaxation, and Desrochers *et al.* (1992) utilized the column generation approach for solving the VRPTW for the first time.

The method of Kohl *et al*. (1997), which was proven to be one of the most efficient methods among the exact methods, succeeded in solving a number of 100 customer instances by relaxing the constraints that ensure that each customer must be visited exactly once and adding a penalty term to the objective function. The model is decomposed into one sub-problem for each vehicle which is a shortest path problem with time window and capacity constraints.

The studies on the heuristic methods for solving the VRPTW are much more than the exact methods since the problem is NP-hard. These algorithms can be grouped as construction algorithms, improvement algorithms, and metaheuristics. Baker and Schaffer (1986) are the first ones proposing the first sequential construction algorithm which is based on the savings heuristic. Solomon (1987) proposed Time Oriented Nearest Neighbourhood Heuristic, Time Oriented Sweep Heuristic (1987), and Giant Tour Heuristics (1987). Antes and Derigs (1995) also proposed a construction algorithm based on Solomon's heuristic.

In the improvement algorithms, generally an exchange intra or inter route neighbourhood is searched to find a better solution. Croes (1958) introduced $k$ opt approach for single vehicle routes. Christofides and Beasley (1984) proposed the $k$ - node interchange for the first time to take time windows into account. Potvin and Rousseau (1995) presented two variants of 2-Opt and Or-Opt, and Schulze and Fahle (1999) proposed shift-sequence algorithm.

Metaheuristic algorithms such as simulated annealing (SA), tabu search (TS), genetic algorithm (GA), and ant colony algorithm (ACO) have been used to solve the VRPTW in order to escape local optima and enlarge the search.

Chiang and Russell (1996) proposed three different SA methods. Tan *et al.* (2001) proposed an SA heuristic, defining a new cooling schedule. Finally, Li and

Lim (2003) proposed an algorithm that finds an initial solution using Solomon's insertion heuristic and then starts local search from initial solution using tabu-embedded simulated annealing approach.

Garcia *et al.* (1994) applied TS to solve VRPTW for the first time, by generating an initial solution using Solomon's insertion heuristic and searching the neighborhood using 2-opt and Or-opt. Garcia *et al.* (1994) also parallelized the TS using partitioning strategy. Thangiah et al. (1994) proposed TS combining TS with SA to accept or reject a solution. Potvin *et al.* (1995) proposed an approach similar to Garcia *et al.* (1994) based on the local search method of Potvin and Rousseau (1995). Gendreau, Hertz and Laporte (1994) used complex iteration schemes that involve a partial re-optimization of the target route to solve the VRPTW.

Badeau *et al.* (1997) performed TS by generating a series of initial solutions, decomposing them into groups of routes and penalizing exchanges that are frequently performed. De Backer and Furnon (1997) used the savings heuristic to generate the initial solution and searched the neighbourhood using 2-opt and Or-opt. Schulze and Fahle (1999) proposed a parallel TS heuristic where initial solutions are generated using the savings heuristic and the neighborhood is searched using route elimination and Or-opt.

Thangiah *et al.* (1991) applied the GA to VRPTW for the first time, where GA is proposed to find good clusters of customer. Thangiah *et al.* (1995) generated initial population by clustering the customers randomly into groups and applying the cheapest insertion heuristic for each group. Afterwards, 2-point crossover is used. Potvin and Bengio (1996) performed GA on chromosomes of feasible solutions. Parents are randomly selected and two types of crossover are applied to these parents. The reduction of routes is obtained by two mutation operators, and the routes are further improved by applying Or-Opt. Homberger and Gehring (1999), making a difference by the role of mutation in their algorithm, generated initial population using a modified savings heuristic and a precedence relationship among the genes in a chromosome. Tan *et al.* (2001), differing by the way of determining the customers in different routes, proposed a GA approach in which the genetic operators are applied directly to solutions, represented as integer strings.

Rochat and Taillard (1995) used a probabilistic local search method based on intensifying the solution, which is in some ways similar to the SA approach. Kilby *et al.* (1999) used a memory-based metaheuristic, Guided Local Search (GLS), in which the cost function is modified by adding a penalty term, and improving the solution by applying 2-opt exchanges. In Potvin and Robillard (1999), a competitive neural network is used to cluster the customers. A combination of a competitive neural network and a GA is described. A weight vector is defined for every vehicle and all weight vectors are placed randomly close to the depot initially. Then, customers are selected.

Braysy *et al.* (2000) described a two-step evolutionary algorithm based on the hybridization of a GA consisting of several local searches and route construction heuristics inspired form the studies of Solomon (1987). Tan *et al.* (2001) proposed an artificial intelligence heuristic which can be interpreted as the hybrid combination of SA and TS.

Bullnheimer *et al.* (1998) applied the AS to the VRP with one central depot and identical vehicles for the first time, and Bullnheimer *et al.* (1999) improved this initial algorithm by the random proportional rule and the phremone update structure (1999). Bell and McMullen (2003) differed from Bullnheimer (1999) in the approach of selecting the next customer and pheromone update structure.

Doerner *et al.* (2001) proposed the savings based ant system approach (SbAS) which differs from Bullnheimer *et al.* (1999) with the use of savings function in calculating the visibility.

Gambardella *et al.* (1999) presented Multiple Ant Colony System for Vehicle Routing Problem with Time Windows (MACS-VRPTW). This approach is the main inspiration of this study and it will be explained in detail in the next chapter.

## 3. ANT COLONY SYSTEM

The original ACS (Gambardella and Dorigo, 1996, Dorigo and Gambardella, 1997a, 1997b) was applied to the TSP. In ACS, a number of artificial ants search for good quality solutions to the discrete optimization problems. A solution is described in terms of paths through the states of the problem in accordance with the constraints of the problem. Each ant is assigned to an initial state based on problem criteria and it has to build a solution with a complete tour. Artificial ants find solutions in parallel processes using an incremental constructive mechanism, starting from the initial state and moving to feasible neighbour states. In this search, moves are made by applying a stochastic search policy and choosing the ways of exploitation and exploration probabilistically, guided by ants' memory, problem constraints, pheromone trail accumulated by all the ants from the beginning of the search process and problem-specific heuristic information named as visibility which measures the attractiveness of the next node to be selected.

The closeness $\eta_{ij}$ is defined as the inverse of the arc length in some of the ant colony system formulations; however it is possible to develop new formulations. The pheromone trail $\tau_{ij}$, which is simply the information collected by the ants while they are building solutions, is updated by using the pheromone update functions. Thus, it is dynamic throughout the problem's runtime. Therefore, the management of the pheromone trails gains big importance for constructing better solutions.

Pheromone trails are used for the exploration and exploitation mechanisms. When ant *k* is located at node *i*, it chooses the next node *j* probabilistically in the set of feasible nodes $N_i^k$. In exploitation with probability $q_0$, a node with the highest $\tau_{ij}.[\eta_{ij}]^{\beta,}$ $j \in N_i^k$ is chosen, and in exploration with probability $(1-q_0)$, the node *j* is chosen with the probability function below:

$$p_{ij}^k = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha}\,[\eta_{ij}]^{\beta}}{\sum_{k \in allowed_k} [\tau_{ik}]^{\alpha}\,[\eta_{ik}]^{\beta}} & \textbf{if } \mathbf{j \in allowed}_k \\ 0 & \textbf{otherwise} \end{cases}$$

The amount of the pheromone deposited depends on the goodness of the solution. The pheromone evaporation mechanism prevents the ants to stick to the same part of the search space whereas extra pheromone is deposited on the arcs used by the shortest path by daemon action process. By the strong communication among the ants, it becomes possible to achieve high quality solutions.

In ACS, pheromone trail is updated both locally and globally. In local update, every time an ant moves from node *i* to node *j*, the phremone level on this arc is decreased in order to decrease the attractiveness of this arc, so giving more chance to other not visited nodes to diversify the solution. However, global update takes place after the completion of the solutions and it aims to intensify the search in the best solution neighbourhood. Either the ars on all/some of the constructed solutions or only the arcs on the best solution may be globally updated. Gambardella and Dorigo (1995), Gambardella and Dorigo (1996), Dorigo and Gambardella (1997) have shown that the update of the arcs in only the best solution works better than the update of arcs in all of the solutions.

In the local update, the amount of phremone on arc (*i,j*) is decreased according to the following formula:

$$\tau_{ij} = (1-\rho)\, \tau_{ij} + \rho\, \tau_0$$

$\tau_0$ is the initial value of the pheromone trails and it is taken as $\tau_0 = 1/(n.J_\omega^h)$ where $J_\omega^h$ is the length of the initial solution generated by the nearest neighbourhood heuristic and $n$ is the number of the nodes. Here, p is a parameter affecting the amount of pheromone evaporation.

In the global update, the amount of pheromone on arc (*i,j*) is updated according to the below formula:

$$\tau_{ij} = (1-\rho)\, \tau_{ij} + \rho / J_\omega^{gb}$$

$J_\omega^{gb}$ is the length of the shortest path generated since the beginning of the computation.

The solutions are improved by local search after each ant builds a complete solution. And the process starts from the beginning until a termination condition is met.

```
procedure ACO heuristics()
    While (termination condition not met)
        schedule activities
            ants generation and activity();
            pheromone evaporation();
            daemon actions();
        end schedule activities
    end while
end procedure

procedure ants generation and activity()
    While (available resources)
        new active ant();
    end while
end procedure

procedure new active ant();
    initialize ant();
    M=update ant memory ();
    While (current memory ?complete solution )
        A=read local ant routing table();
        P=compute transition probabilities;
        next state =apply decision policy;
        move to next state(next state);
        if (local pheromone update)
            deposit pheromone on the visited arc();
            update ant routing table();
        end if
        M=update internal state();
    end while
    if (global pheromone update)
        foreach visited arc do
            deposit pheromone on the visited arc();
            update ant routing table();
        end foreach
    end if
    die();
end procedure
```

Figure 3.1 The ACO heuristics developed by Dorigo Caro (1999)

## 4. RMACS for VRPTW

Taking the ACS as a starting point, MACS-VRPTW has been proposed to solve a VRPTW where both the number of vehicles and the travel time have to be minimized, and the minimization of the number of vehicles takes precedence over the travel time minimization. This dual objective minimization is achieved by using two artificial ant colonies based on ACS. Figure 4.1 illustrates the basic principles of MACS-VRPTW.

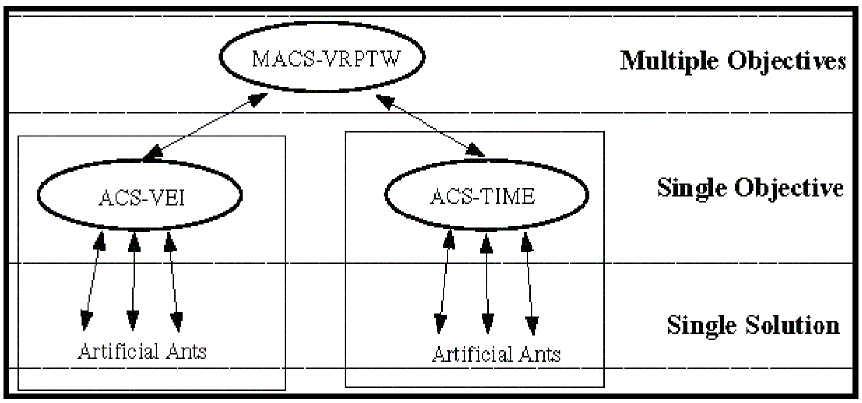


Figure 4.1 The MACS-VRPTW procedure

```
/* MACS-VRPTW: Multiple Ant Colony System for Vehicle Routing Problems with Time Windows */
Procedure MACS-VRPTW()
  1. /* Initialization */
     /* ω^gb is the best feasible solution: lowest number of vehicles and shortest travel time
      #active_vehicles(ω) computes the number of active vehicles in the feasible solution ω */
     ω^gb feasible initial solution with unlimited number of vehicles produced with a nearest neighbor heuristic
  2. /* Main loop */
  Repeat
       v ← #active_vehicles(ω^gb)
       Activate ACS-VEI(v - 1)
       Activate ACS-TIME(v)
       While ACS-VEI and ACS-TIME are active
            Wait an improved solution ω from ACS-VEI or ACS-TIME
            ω^gb ← ω
            if #active_vehicles(ω^gb) < v then
                 kill ACS-TIME and ACS-VEI
       End While
  until a stopping criterion is met
```

Figure 4.2 The MACS-VRPTW algorithm

The first colony, ACS-VEI, tries to diminish the number of vehicles used, while the second colony, ACS-TIME, optimizes the feasible solutions found by ACS-VEI. Although both colonies use independent pheromone trails, they collaborate by sharing the variable $\omega^{gb}$. The solution reached by the nearest neighbourhood heuristic at the start of the algorithm is saved in $\omega^{gb}$, then this solution is improved by the cooperative work of the two colonies.

When ACS-VEI is called, it works with one vehicle less than the number of vehicles used in $\omega^{gb}$ and tries to find a feasible solution. During its search, it finds infeasible solutions with the *new_active_ant* procedure, which will be explained later and it stores the solution with the highest number of visited customers in $\omega^{ACS\text{-}VEI}$. So in ACS-VEI the current best solution is generally the infeasible solution with the maximum number of visited customers. ACS-TIME is called then and it tries to optimize the total travel time by using as many vehicles as used in $\omega^{gb}$ while running the *new_active_ant* algorithm. Whenever an improved solution comes from either of the colonies, both $\omega^{gb}$ and the pheromone values are updated globally. Whenever the improved solution contains fewer vehicles than the vehicles used in $\omega^{gb}$, both ACS-TIME and ACS-VEI colonies are killed and the process continues with the two new colonies working with the reduced number of vehicles.

### 4.1 ACS-TIME and ACS-VEI Colonies

The working principles of ACS-VEI and ACS-TIME colonies are described in Figure 4.3 and Figure 4.4.

```
/* ACS-TIME: Travel time minimization. */
Procedure ACS-TIME(v)
/* Parameter v is the smallest number of vehicles for which a feasible solution has been computed */
1. /* Initialization */
   initialize pheromone and data structures using v
2. /* Cycle */
   Repeat
        for each ant k
        /* construct a solution ω^k */
          new_active_ant(k, local_search=TRUE, 0)
        end for each
          /* update the best solution if it is improved */
        If there exists a k : ω^k is feasible and J_ω^k < J_ω^gb then
               send ω^k to MACS-VRPTW
        /* perform global updating according to below equation */
        τ_ij = (1- ρ) τ_ij + ρ / J_ω^gb
   until a stopping criterion is met
```

Figure 4.3 The ACS-TIME procedure

```
/* ACS-VEI: Number of vehicles minimization. */

Procedure ACS-VEI(s)
/* Parameter s is set to v-1, that is, one vehicle less than the smallest number of vehicles for
which a feasible solution has been computed
#visited_customers(ω) computes the number of customers that have been visited in
solution */
1. /* Initialization */
  initialize pheromone and data structures using s
  ω^ACS-VEI : initial solution with s vehicles produced with a nearest
neighbor heuristic. /* ω^ACS-VEI is not necessarily feasible */
2. /* Cycle */
    Repeat
         for each ant k
             /* construct a solution ω^k */
             new_active_ant(k,local_search=FALSE,IN)
             for every customer j ∉ ω^k : INj ← INj + 1
         end for each
       /* update the best solution if it is improved */
       If for any k:
     #visited_customers(ω^k) > #visited_customers(ω^ACS-VEI) then
     ω^ACS-VEI ← ω^k
      for every j: INj ← 0 /* reset IN */
      if ω^ACS-VEI is feasible then
        send ω^ACS-VEI to MACS-VRPTW
/* perform global updating according to below equation using both ω^ACS-VEI and ω^gb */
τij = (1- ρ) τij + ρ / Jω^ACS-VEI   for every (i, j) ∈ ω^ACS-VEI

τij = (1- ρ) τij + ρ / Jω^gb   for every (i, j) ∈ ω^gb


    until a stopping criterion is met
```

Figure 4.4 The ACS-VEI procedure

$IN_j$ stores the number of time a node is not inserted in a solution and it makes possible to favor the nodes which are less frequently inserted in the solutions. ACS-VEI and ACS-TIME use the same *new_active_ant* constructive procedure that is presented in details in Figure 4.5.

```
/* new_active_ant: constructive procedure for ant k used by ACS-VEI and ACS-TIME */
Procedure new_active_ant(k, local_search, IN)
1. /* Initialization*/
        put ant k in a randomly selected depot i
       ω^k ← ι
       current_time_k ← 0 , loadk ← 0
2. /* This is the step in which ant k builds its tour. Tour is stored in ω^k */
   Loop
       /* Starting from node i compute the set N_i^k of feasible nodes (i.e., all the nodes j still to be visited and such that current_timek and loadk are compatible with time windows [bj,ej] and delivery quantity qj of customer j)
       for every j ∈ N_i^k compute the attractiveness ηij as follows: */
            delivery_timej ← max(current_timek + t_ij, bj)
            delta_timeij ←delivery_timej - current_timek
            distanceij ←delta_timeij *( ej - current_timek)
            distanceij ←max(1.0, (distanceij - INj))
            η_ij ← 1.0/ distanceij
      Choose probabilistically the next node j using η_ij in exploitation and exploration mechanisms
     ω^k ← ω^k + j
     current_timek ←delivery_timej
     loadk ←loadk + qj
   If j is a depot then current_timek ←0, loadk ←0
  τ_ij = (1- ρ) τ_ij + ρ τ_0
    /* Local pheromone updating */
   i ←j /* New node for ant k */
   Until
  N_i^k = {} /* no more feasible nodes are available */
3. /* In this step path ω^k is extended by tentatively inserting non visited customers */
   ω^k ← insertion_procedure(ω^k)
4. /* In this step feasible paths are optimized by a local search procedure.
     The parameter local_search is TRUE in ACS-TIME and it is FALSE in ACS-VEI*/
     if local_search = TRUE and ω^k is feasible then
    ω^k ← local_search_procedure(ω^k)
```

Figure 4.5 The new_active_ant procedure used by ACS-VEI and ACS-TIME

At the end of the constructive phase, some nodes may not have been visited making the solution incomplete, afterwards the solution is tentatively completed by performing further insertions. Lastly, ACS-TIME implements a local search procedure to decrease the total travel time.

## 5. SOLUTION CONSTRUCTIVE PROCEDURE

The general methodology of the MACS-VRPTW is applied in our algorithm. However, our algorithm differs in some aspects. First of all, a nearest list array is defined at the beginning of the problem and that list is used during the implementation of the whole code. In the nearest neighbourhood heuristic, which is used to find an initial feasible solution, the first point in the nearest list array is selected to be visited next, among the feasible nodes if its reachtime is between the ready time and due date otherwise a point with the minimum due date is selected to be visited.

After an initial solution is found, the MACS procedure takes place, which calls ACS-VEI and ACS-TIME followingly. In the ACS-VEI, the solution is computed for *v-1* vehicles, where *v* is the number of vehicles in global feasible solution. Here, we take out the vehicle with the maximum capacity available and apply insertion for the nodes not visited before starting the *new_active_ant* algorithm. The insertion algorithm attempts to place an unvisited point to the first suitable place on the nearest list array, which matches with the time constraints of the nodes on the route and the vehicle capacity constraint.

Another difference of our algorithm lies in the calculation of the attractiveness function. In the *new_active_ant algorithm,* the vehicles search for the customers at which they will not wait or they will wait at minimum. Although, this is a reasonable logic, in many of the problem instances, the vehicles have to return to the depot with available capacity, but no feasible point to visit remained. It can be observed in the problem instances where the number of vehicles is large and small number of customers are visited in each route. At these cases, the insertion algorithms do not work either, so improving the solution becomes very difficult.

In order to find a solution for these cases, we defined two more rules on finding the attractiveness of the customers. The first rule is the remaining capacity rule, in which if a vehicle's remaining load is equal to a feasible customer's demand, then this customer's attractiveness becomes 1. This rule slightly decreases the remaining loads on the vehicles when they are returning to the depot.

Second and the more important rule is the accessibility rule. We define another constant, accessibility and set its value to 0.98. For every turn, if rule 1 explained above is not applicable, a random number between 0 – 1 is generated. If this number is less than the accessibility constant, normal attractiveness finding procedure (explained in the detailed description of the algorithm in Appendix A) is applied. Otherwise, the distance is initialized as the time between the delivery time and due date. We made an insertion point in that route, searching a nearest point with a (minimum euclidean distance between two points + serviceTime) delay. So our insertion procedure tries to insert the points just before or after the points which are very close. To insert a point to a completed route, we have to take the service time into account.

In Appendix A the detailed description of the algorithm is attached.

## 6. NUMERICAL RESULTS

Our algorithm has been tested on a classical set of 56 benchmark problems of Solomon (1987) which consists of six different problem types: C1, C2, R1, R2, RC1, RC2. Each data set contains eight to twelve 100-node problems. C type problems have clustered customers whose time windows were generated based on a known solution. R type problems have customers location generated uniformly randomly over a square. RC type problems have a combination of randomly placed and clustered customers. Type 1 problems have narrow time windows and small vehicle capacity, whereas type 2 problems have large time windows and large vehicle capacity. Therefore, the solutions of type 2 problems have very few routes and significantly more customers per route.

The algorithm coded in C++ run 5 times for each problem data set and the average of the solutions of 5 runs are listed in Appendix C. By applying several runs to different problems, the following parameters are selected to be used in the experiments: $m$=30 ants, $q_0$=0.9, $\beta$=2 and $\rho$=0.1.

Table 6.1 Detailed solutions comparison (30 ants case)

| | RMACS Best | | Best Known | | | RMACS Best | | Best Known | |
|---|---|---|---|---|---|---|---|---|---|
| | TD | NV | TD | NV | | TD | NV | TD | NV |
| c101 | 852,95 | 10 | 828,94 | 10 | c201 | 591,56 | 3 | 591,56 | 3 |
| c102 | 1024,76 | 10 | 828,94 | 10 | c202 | 768,34 | 3 | 591,56 | 3 |
| c103 | 1022,12 | 10 | 828,06 | 10 | c203 | 743,32 | 4 | 591,17 | 3 |
| c104 | 1069,51 | 10 | 824,78 | 10 | c204 | 802,65 | 3 | 590,6 | 3 |
| c105 | 852,95 | 10 | 828,94 | 10 | c205 | 612,93 | 3 | 588,88 | 3 |
| c106 | 945,98 | 10 | 828,94 | 10 | c206 | 643,23 | 3 | 588,49 | 3 |
| c107 | 858,82 | 10 | 828,94 | 10 | c207 | 644,84 | 3 | 588,29 | 3 |
| c108 | 968,66 | 10 | 828,94 | 10 | c208 | 623,57 | 3 | 588,32 | 3 |
| c109 | 1052,74 | 10 | 828,94 | 10 | | | | | |
| **Average** | **949,47** | **10,00** | **828,31** | **10,00** | **Average** | **678,80** | **3,13** | **589,86** | **3,00** |
| r101 | 1994,48 | 20 | 1645,79 | 19 | r201 | 1643,43 | 4 | 1252,37 | 4 |
| r102 | 1774,27 | 18 | 1486,12 | 17 | r202 | 1535,68 | 4 | 1191,7 | 3 |
| r103 | 1496,77 | 14 | 1292,68 | 13 | r203 | 1228,52 | 3 | 939,54 | 3 |
| r104 | 1216,70 | 11 | 1007,24 | 9 | r204 | 1033,20 | 3 | 825,52 | 2 |
| r105 | 1690,22 | 15 | 1377,11 | 14 | r205 | 1235,67 | 3 | 994,42 | 3 |
| r106 | 1519,77 | 14 | 1251,98 | 12 | r206 | 1162,32 | 3 | 906,14 | 3 |
| r107 | 1385,89 | 12 | 1104,66 | 10 | r207 | 1120,92 | 3 | 893,33 | 2 |
| r108 | 1191,65 | 10 | 960,88 | 9 | r208 | 923,64 | 3 | 726,75 | 2 |
| r109 | 1479,67 | 12 | 1194,73 | 11 | r209 | 1186,41 | 3 | 909,16 | 3 |
| r110 | 1425,40 | 12 | 1118,59 | 10 | r210 | 1147,54 | 3 | 939,34 | 3 |
| r111 | 1434,20 | 12 | 1096,72 | 10 | r211 | 1148,94 | 3 | 892,71 | 2 |
| r112 | 1165,11 | 10 | 982,14 | 9 | | | | | |
| **Average** | **1481,18** | **13,33** | **1209,89** | **11,92** | **Average** | **1215,12** | **3,18** | **951,91** | **2,73** |
| rc101 | 1972,47 | 15 | 1696,94 | 14 | rc201 | 1766,41 | 4 | 1406,91 | 4 |
| rc102 | 1730,73 | 14 | 1554,75 | 12 | rc202 | 1706,52 | 4 | 1367,09 | 3 |
| rc103 | 1623,52 | 12 | 1261,67 | 11 | rc203 | 1374,14 | 3 | 1049,62 | 3 |
| rc104 | 1418,41 | 11 | 1135,48 | 10 | rc204 | 987,50 | 3 | 798,41 | 3 |
| rc105 | 1890,82 | 15 | 1629,44 | 13 | rc205 | 1677,18 | 4 | 1297,19 | 4 |
| rc106 | 1692,36 | 13 | 1424,73 | 11 | rc206 | 1479,02 | 4 | 1146,32 | 3 |
| rc107 | 1567,16 | 12 | 1230,48 | 11 | rc207 | 1380,51 | 3 | 1061,14 | 3 |
| rc108 | 1380,73 | 11 | 1139,82 | 10 | rc208 | 1045,72 | 3 | 828,14 | 3 |
| **Average** | **1659,53** | **12,88** | **1384,16** | **11,50** | | **1427,13** | **3,50** | **1119,35** | **3,25** |

The results achieved by setting the parameters to $m$=10 ants, $q_0$=0.9, β=1 and ρ=0.1 are listed in Appendix B. Although increasing the number of ants from 10 to 30 increased the computational time from 15 minutes to approximately 40 minutes for each problem instance, the travel distance improved a lot (additionally in R102 the number of vehicles decreased to 18 from 20). Increasing the number of ants or the number of runs furthermore do not improve the solutions almost at all, however the computational time increases exponentially. Therefore, the parameters are chosen as: $m$=30 ants, $q_0$=0.9, β=2 and ρ=0.1.

In Table 6.1, we observe that the RMACS for VRPTW provides competitive results for C1 and C2 type problems, since it gives the same number of vehicles (except c203) with the best benchmark solutions. The approach also gives at most 2 more vehicles compared with the best benchmarks in the other problem sets.

However, the total travel time is in some instances %30 larger than the best benchmarks. Note also that the RMACS for VRPTW gets these results in approximately 40 minutes of computational time for each problem instance.

Table 6.2 Average of the best solutions computed by RMACS and MACS

| | R1 | | C1 | | RC1 | |
|---|---|---|---|---|---|---|
| | VEI | DIST | VEI | DIST | VEI | DIST |
| RMACS | 13,3 | 1481,2 | 10,0 | 949,5 | 12,9 | 1659,5 |
| MACS | 12,0 | 1217,1 | 10,0 | 828,4 | 11,6 | 1382,4 |
| | R2 | | C2 | | RC2 | |
| | VEI | DIST | VEI | DIST | VEI | DIST |
| RMACS | 3,2 | 1215,1 | 3,1 | 678,8 | 3,5 | 1427,1 |
| MACS | 2,7 | 967,8 | 3,0 | 589,9 | 3,3 | 1129,2 |

## 7. CONCLUSIONS

A RMACS approach for VRPTW is proposed is this study. The problem has two objectives: the minimization of the number of vehicles which is the primary objective, and the minimization of the total travel time. Two artificial ant colonies, one minimizing the number of vehicles and the other the total travel time, cooperate with each other through pheromone update to optimize these objectives.

The algorithm differs from the MACS of Gambardella (1999) by the usage of the nearest list array, application of the insertion algorithm at the beginning of the ACS-VEI, and the calculation of the attractiveness function. The change in the attractiveness function makes it possible to make insertions after the ACS-VEI is completed.

The algorithm is tested on the well-known problems of Solomon (1987) and the results are compared with the best benchmarks and the MACS of Gambardella (1999). The RMACS algorithm finds the same number of vehicles as the best solutions or 1-2 near to them, and the travel distance is in some of the problem instances %30 larger than the best solutions.

Future work may focus on the attractiveness function and the pheromone update structure. Since both functions have a significant importance on the results, the improvements to these functions may improve he results considerably. Computational time is not the main concern of this study; however the algorithm may be run on parallel computers to improve the computational time. The RMACS algorithm may also be applied to other types of VRPs with modifications.

**Bibliography**


1. Gambardella, L.M., Taillard, E., Agazzi, G., "MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows," Technical Report, IDSIA, Lugano, Switzerland, 1999.

2. Bullnheimer, B., Hartl, R.F., Strauss, C., "Applying ant system to the vehicle routing problem," Presented at the *2nd International Conference on Metaheuristics*, Sophia, France, July 21-24, 1997.

3. Chiang, W., Russell, R. "Simulated annealing metaheuristics for the vehicle routing problem with time windows," *Annals of Operations Research*, (63), pp.3-27,1996.

4. Clarke, G., Wright W., "Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research* , (12), pp.568-581, 1964

5. Dorigo, M., Maniezzo, V. Colorni, A., "The ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, vol.26, pp.29-41, 1996.

6. Gambardella, L.M., Dorigo, M., "HAS-SOP: Hybrid Ant System for the Sequential Ordering Problem, Technical Report IDSIA 11-97, IDSIA, Lugano, Switzerland, 1997.

7. Bullnheimer, B., Hartl, R.F., Strauss, C., "A new ranked based version of the ant system," Working Paper, Vienna University of Economics and Business Administration, Austria, 1997.

8. Kohl, N., "Exact methods for time constrained routing and scheduling problems," Phd. Thesis, Department of Mathematical Modeling, Technical University of Denmark, 1995.

9. N., Madsen O., "An optimization algorithm for the vehicle routing problem with time windows based on Lagrangean Relaxation," *Operations Research*, (45), pp.395-406, 1997.

10. Kolen, A., Rinnooy A., Trienekens, H., "Vehicle routing with time windows," *Operations Research* , (35), pp.266-273, 1987

11. Larsen, J., "Parallelization of the vehicle routing problem with time Windows," PhD. Thesis, Technical University of Denmark, Lyngby, 1999.

12. Montemanni, R., Gambardella, L.M., Rizzoli, A.E., Donati, A.V., "A new algorithm for a Dynamic Vehicle Routing Problem based on Ant Colony System," *ODYSSEUS 2003: Second International Workshop on Freight Transportation and Logistic* s, Palermo, Italy, 27- 30 May 2003.

13. Potvin, J., Kervahut, T., Garcia, B.L., Rousseau, J.M., "The vehicle routing problem with time windows; part I: tabu search," *INFORMS Journal on Computing,* (8), pp.158-164, 1995.

14. Potvin,J., Bengio, S., "The vehicle routing problem with time windows-part II: genetic search," *INFORMS Journal on Computing* , (8), pp.165-172, 1996.

15. Rochat, Y., Taillard, E.D., "Probabilistic Diversification and intensification in local search for vehicle routing," *Journal of Heuristics*, (1), pp.147-167, 1995.

16. Savelsbergh, M.W.P., "Local search for routing problems with time windows," Annals of *Operations Research*, (4), pp.285-305, 1985.

17. Stützle, T., Hoos, H H.., "Improving the Ant System: A Detailed Report on the MAX-MIN Ant System," Technical Report AIDA-96-12 - Revised version, Darmstadt University of Technology, Computer Science Department, Intellectics Group., 1996.

18. Stützle , T., Hoos, H.H., "MAX-MIN ant s ystem and local search for combinatorial optimization problems ," *Meta -Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, pp. 313-329, Boston, 1999.

19. http://web.cba.neu.edu/~msolomon/problems.htm, July, 2004.

20. Deitel, H.M., Deitel, P.J., *C++, How to Program*, Prentice Hall, New Jersey, 2001.

21. Gendreau, Hertz, Laporte, "A tabu search heuristic for the vehicle routing problem", *Management Science*, (40), pp.1276-1290, 1994.

## Appendix A

## Description of the RMACS

### Main Procedure

For every problem in a specific problem set

**Step 1:** While there exist an unvisited point, create a vehicle and load it with the maximum capacity, and set that vehicle to be in depot

**Step 2:** For every unvisited point (from the nearest to farthest)
Calculate the reaching time from the depot to this point
Check if this reaching time is bw that point's ready time and due date
If it is in that interval, send this vehicle to that point
else
Send the vehicle to the unvisited point with the minimum due date
Update that vehicle's condition (load and route conditions)
If there is/are unvisited point(s), firstly search for the points which are unvisited and when the vehicle gets there, current time will be in ready time due date interval
If there exist such a point/s then, go to the nearest one of them
else
Send the vehicle to the point at which our vehicle will wait minimumly
Update that vehicle's condition(load and route conditions)
If there is not enough capacity or there does not exist a feasible point
Send that vehicle to depot and check its total route
After there isn't any point left set this solution as global&best solution

For 10 times, call MACS_VRPTW function

**Step 3: MACS_VRPTW Function**

For 5 times
Calculate the toZero value
Call ACS_VEI function
Call ACS_TIME function
If the global solution found is improved, update it

**Step 4: ACS_TIME Function**

Calculate the toZero value
Initialize the pheromone levels according to that toZero value
For 100 times
For the number of ant times
Reset all points ( Make all points unvisited )

Call NEW_ACTIVE_ANT function and get a solution
Calculate the visited points in this solution
If the global solution is improved, update it
After the ants find their solutions, make the global pheromone update

**ACS_VEI Function**

**Step 5:** Calculate the toZero value
Calculate the load of the vehicles at the time they are returning to the depot in global solution
Find the vehicle least used and exclude it from global solution
Store this new solution as oneVehicleLessSolution.
Mark the visited customers in that oneVehicleLessSolution
Count the number of visited customers

**Step 6:** For the unvisited customers, find the nearest point's location in the route and try to insert the unvisited point near that found point
else
Try to insert that unvisited node to the start and end points of all routes
If there is an successfull insertion, start the loop from the beginning
After insertion procedure is finished, recalculate the unvisited points
If there is not an unvisited point, update the global solution
Re-create a oneVehicleLessSolution with the same procedure
Calculate the toZero value according of the oneVehicleLessSolution
Initialize all pheromone levels with this toZero value.

**Step 7:** For 100 times
For number of ant times
Reset all points ( Make all points unvisited )
Call NEW_ACTIVE_ANT function and get a solution
Calculate the visited points in this solution
Increment the insertion of the unvisited points

**Step 8:** For every solution coming from NEW_ACTIVE_ANT
Compare it with the oneVehicleLessSolution
If it is improved, update the solution
Reset insertion values of all the points to zero
If this solution visits all customers, update global solution
If it is better just for travel distance and feasible, update the minimumTravelDistanceSolution ignoring the vehicle number
Update pheromones (with oneVehicleLessSolution&global solution)

**NEW_ACTIVE_ANT Function**

**Step 9:** Reset all points to unvisited state
While there are unvisited nodes, create a vehicle, load it to capacity
Make this vehicle to be in depot

**Step 10:** While there available points such that this vehicle can go
Find attractiveness' of all points according to these rules :
If demand of a point is equal to remaining load on that vehicle
Than attractiveness of this point is 1
else
Create a random number between 0 – 1
If this number is less than the availability constant
delivery_time=max(current_time + travel_time, ready_time)
delta_time = delivery_time - current_time
distance = delta_time * ( due_date - current_time )
distance = max(1.0, (distance - Insertion))
attractiveness = 1.0/ distance
else
deliveryTime = Max( reachTime, readyTime)
distance = dueDate - deliveryTime
distance=Max((mindistance bw two points+serviceTime ),distance )
attractiveness=(distance bw two points + serviceTime ) / distance

**Step 11:** Find the probabilities of all points from the previous point
Create a random number between 0 - 1
If the number is less than 0.9, make the next point as more attracted
else
Create a random number between 0 – 1
Make the next point the one having the nearest random probability
Make vehicle to go to that point and the state of that point as visited
Update that vehicle's conditions ( load, route )

**Step 12:** At the end of a vehicle's route, check the feasibility of the route
After a solution is completed, find the used vehicle number

**Step 13:** If this vehicle number is more than oneVehicleLessSolution
Exclude these routes from that solution
Re-Calculate unvisited point number
If it is not zero
For the unvisited customers from the nearest to farthest
Search all points in oneVehicleLessSolution's vehicle routes
If the nearest point is not depot
Try to insert to the before and after the nearest point
else
Try to insert that unvisited to the start and end points of routes
If there is a successfull insertion, start the loop from the beginning
Recalculate the unvisited points

**Step 14:** If it is zero (a feasible solution) and the NEW_ACTIVE_ANT function is called from ACS_TIME function
Start 3 - opt local search
For 100 times
Create three random numbers acting as vehicle ID
Create a random number acting as a point's order in a route
Interchange those three points checking the validity
If these routes are valid and travel distance is smaller
Update the global solution

# Appendix B

## Detailed solutions comparison (10 ants case)

| | RMACS Best | | Best Known | | | RMACS Best | | Best Known | |
|---|---|---|---|---|---|---|---|---|---|
| | TD | NV | TD | NV | | TD | NV | TD | NV |
| c101 | 852,95 | 10 | 828,94 | 10 | c201 | 591,56 | 3 | 591,56 | 3 |
| c102 | 1300,07 | 10 | 828,94 | 10 | c202 | 908,34 | 3 | 591,56 | 3 |
| c103 | 1282,12 | 10 | 828,06 | 10 | c203 | 1171,91 | 4 | 591,17 | 3 |
| c104 | 1221,69 | 10 | 824,78 | 10 | c204 | 986,35 | 3 | 590,6 | 3 |
| c105 | 934,36 | 10 | 828,94 | 10 | c205 | 621,11 | 3 | 588,88 | 3 |
| c106 | 954,76 | 10 | 828,94 | 10 | c206 | 662,59 | 3 | 588,49 | 3 |
| c107 | 858,82 | 10 | 828,94 | 10 | c207 | 663,19 | 3 | 588,29 | 3 |
| c108 | 968,66 | 10 | 828,94 | 10 | c208 | 644,95 | 3 | 588,32 | 3 |
| c109 | 1054,06 | 10 | 828,94 | 10 | | | | | |
| **Average** | **1046,68** | **10,00** | **828,31** | **10,00** | **Average** | **781,25** | **3,13** | **589,86** | **3,00** |
| r101 | 1994,48 | 20 | 1645,79 | 19 | r201 | 1932,91 | 4 | 1252,37 | 4 |
| r102 | 1811,49 | 20 | 1486,12 | 17 | r202 | 1635,68 | 4 | 1191,7 | 3 |
| r103 | 1496,77 | 14 | 1292,68 | 13 | r203 | 1532,01 | 3 | 939,54 | 3 |
| r104 | 1222,95 | 11 | 1007,24 | 9 | r204 | 1133,20 | 3 | 825,52 | 2 |
| r105 | 1697,43 | 15 | 1377,11 | 14 | r205 | 1535,20 | 3 | 994,42 | 3 |
| r106 | 1542,18 | 14 | 1251,98 | 12 | r206 | 1362,32 | 3 | 906,14 | 3 |
| r107 | 1385,89 | 12 | 1104,66 | 10 | r207 | 1300,92 | 3 | 893,33 | 2 |
| r108 | 1191,65 | 10 | 960,88 | 9 | r208 | 1104,87 | 3 | 726,75 | 2 |
| r109 | 1534,04 | 12 | 1194,73 | 11 | r209 | 1426,41 | 3 | 909,16 | 3 |
| r110 | 1434,27 | 12 | 1118,59 | 10 | r210 | 1585,42 | 3 | 939,34 | 3 |
| r111 | 1435,07 | 12 | 1096,72 | 10 | r211 | 1231,99 | 3 | 892,71 | 2 |
| r112 | 1165,11 | 10 | 982,14 | 9 | | | | | |
| **Average** | **1492,61** | **13,50** | **1209,89** | **11,92** | **Average** | **1434,63** | **3,18** | **951,91** | **2,73** |
| rc101 | 1972,47 | 15 | 1696,94 | 14 | rc201 | 2066,41 | 4 | 1406,91 | 4 |
| rc102 | 1730,73 | 14 | 1554,75 | 12 | rc202 | 1906,52 | 4 | 1367,09 | 3 |
| rc103 | 1623,52 | 12 | 1261,67 | 11 | rc203 | 1588,31 | 3 | 1049,62 | 3 |
| rc104 | 1418,41 | 11 | 1135,48 | 10 | rc204 | 1183,02 | 3 | 798,41 | 3 |
| rc105 | 1890,82 | 15 | 1629,44 | 13 | rc205 | 2067,18 | 4 | 1297,19 | 4 |
| rc106 | 1692,36 | 13 | 1424,73 | 11 | rc206 | 1679,02 | 4 | 1146,32 | 3 |
| rc107 | 1567,16 | 12 | 1230,48 | 11 | rc207 | 1655,00 | 3 | 1061,14 | 3 |
| rc108 | 1380,73 | 11 | 1139,82 | 10 | rc208 | 1321,99 | 3 | 828,14 | 3 |
| **Average** | **1659,53** | **12,88** | **1384,16** | **11,50** | | **1683,43** | **3,50** | **1119,35** | **3,25** |

# Appendix C

## Average of the 5 runs of RMACS vs best known (30 ants case)

| | RMACS Averages | | Best Known | | | RMACS Averages | | Best Known | |
|---|---|---|---|---|---|---|---|---|---|
| | TD | NV | TD | NV | | TD | NV | TD | NV |
| c101 | 852,95 | 10 | 828,94 | 10 | c201 | 591,56 | 3 | 591,56 | 3 |
| c102 | 1154,32 | 10 | 828,94 | 10 | c202 | 786,43 | 3 | 591,56 | 3 |
| c103 | 1033,45 | 10 | 828,06 | 10 | c203 | 755,98 | 4 | 591,17 | 3 |
| c104 | 1099,72 | 10 | 824,78 | 10 | c204 | 802,65 | 3 | 590,6 | 3 |
| c105 | 852,95 | 10 | 828,94 | 10 | c205 | 612,93 | 3 | 588,88 | 3 |
| c106 | 1021,09 | 10 | 828,94 | 10 | c206 | 643,23 | 3 | 588,49 | 3 |
| c107 | 858,82 | 10 | 828,94 | 10 | c207 | 655,78 | 3 | 588,29 | 3 |
| c108 | 998,12 | 10 | 828,94 | 10 | c208 | 648,76 | 3 | 588,32 | 3 |
| c109 | 1087,32 | 10 | 828,94 | 10 | | | | | |
| **Average** | **995,42** | **10,00** | **828,38** | **10,00** | **Average** | **687,16** | **3,13** | **589,86** | **3,00** |
| r101 | 1999,98 | 20 | 1645,79 | 19 | r201 | 1689,72 | 4 | 1252,37 | 4 |
| r102 | 1823,32 | 18 | 1486,12 | 17 | r202 | 1610,92 | 4 | 1191,7 | 3 |
| r103 | 1522,65 | 14 | 1292,68 | 13 | r203 | 1278,13 | 3 | 939,54 | 3 |
| r104 | 1236,76 | 11 | 1007,24 | 9 | r204 | 1112,23 | 3 | 825,52 | 2 |
| r105 | 1698,24 | 15 | 1377,11 | 14 | r205 | 1301,34 | 3 | 994,42 | 3 |
| r106 | 1534,32 | 14 | 1251,98 | 12 | r206 | 1178,32 | 3 | 906,14 | 3 |
| r107 | 1389,11 | 12 | 1104,66 | 10 | r207 | 1160,97 | 3 | 893,33 | 2 |
| r108 | 1205,45 | 10 | 960,88 | 9 | r208 | 946,54 | 3 | 726,75 | 2 |
| r109 | 1498,34 | 12 | 1194,73 | 11 | r209 | 1201,56 | 3 | 909,16 | 3 |
| r110 | 1430,12 | 12 | 1118,59 | 10 | r210 | 1238,17 | 3 | 939,34 | 3 |
| r111 | 1466,57 | 12 | 1096,72 | 10 | r211 | 1272,13 | 3 | 892,71 | 2 |
| r112 | 1198,23 | 10 | 982,14 | 9 | | | | | |
| **Average** | **1500,26** | **13,35** | **1209,89** | **11,92** | **Average** | **1271,82** | **3,18** | **951,91** | **2,73** |
| rc101 | 1986,32 | 15 | 1696,94 | 14 | rc201 | 1801,29 | 4 | 1406,91 | 4 |
| rc102 | 1745,72 | 14 | 1554,75 | 12 | rc202 | 1765,91 | 4 | 1367,09 | 3 |
| rc103 | 1640,52 | 12 | 1261,67 | 11 | rc203 | 1402,74 | 3 | 1049,62 | 3 |
| rc104 | 1446,48 | 11 | 1135,48 | 10 | rc204 | 1000,18 | 3 | 798,41 | 3 |
| rc105 | 1902,23 | 15 | 1629,44 | 13 | rc205 | 1698,10 | 4 | 1297,19 | 4 |
| rc106 | 1702,43 | 13 | 1424,73 | 11 | rc206 | 1498,24 | 4 | 1146,32 | 3 |
| rc107 | 1587,35 | 12 | 1230,48 | 11 | rc207 | 1397,42 | 3 | 1061,14 | 3 |
| rc108 | 1400,12 | 11 | 1139,82 | 10 | rc208 | 1075,46 | 3 | 828,14 | 3 |
| **Average** | **1676,40** | **12,88** | **1384,16** | **11,50** | | **1454,92** | **3,50** | **1119,35** | **3,25** |