Internal Report 2013–02 February 2013

Universiteit Leiden

Opleiding Informatica

Solving the Dynamic Vehicle Routing Problem with Time Windows using Ant Colony Optimization

Barry van Veen

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Solving the Dynamic Vehicle Routing Problem with Time Windows using Ant Colony Optimization

Barry van Veen
barryvanveen@gmail.com
LIACS, Leiden University

Supervisor
Dr. M.T.M. Emmerich
Second reader
Prof. Dr. T.H.W. Bäck

February 27, 2013

Contents

Abstract

Artificial routing problems like the Vehicle Routing Problem relate to many real-world problems. Many 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. Results are presented that show the performance on a new set of benchmark problems with varying degrees of dynamicity. Also, a method for applying two-stage planning to the Stochastic Vehicle Routing Problem is described.

Chapter 1

Introduction

Routing problems like the traveling salesman problem and the vehicle routing problem are among the most studied problems in computer science. They are as relevant to business as they have ever been and because of their complexity they still offer a good challenge. A multitude of algorithms has been developed to solve routing problems. They range from exact algorithms like the Nearest Neighbor algorithm [15] to constraint programming [31] to meta-heuristics like Ant Colony Optimization [16].

With recent developments in communication and positioning systems like GPS 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 that algorithms have to be designed for.

In this report we will deal specifically with an Ant Colony Optimization approach to solve the Dynamic Vehicle Routing Problem with Time Windows. This specific problem has not been solved with and ant colony optimization approach. To test our new approach a set of benchmark problems was created that is made public for others to use.

This master thesis is the final report for the Computer Science master program at Leiden University. The resulting report and software contribute to the DELIVER project that is a collaboration with the goal to design a framework for continuous logistic planning, as an improvement of traditional a priori planning[1].

Besides this Ant Colony Optimization approach we also formulated a two-stage stochastic programming approach to solve the Stochastic Vehicle Routing Problem.

The remainder of this work is structured as follows. First Chapter 2 will introduce the routing problems that will be discussed in the following chapters. Then Chapter 3 gives an extensive introduction into Ant Colony Optimization, its most common variants and extensions. Chapter 4 introduces the MACS-VRPTW algorithm that serves as a basis for the new MACS-DVRPTW algorithm that is described in Chapter 5. In Chapter 6 a two-stage planning approach is formulated. The report ends with some conclusions in Chapter 7 and directions for further work in Chapter 8.

Throughout this report many abbreviations will be introduced to describe different routing problems. Also, the chapters on Ant Colony Optimization and the MACS-VRPTW and MACS-DVRPTW algorithms use many symbols and letters in equations and pseudo-codes. Therefore a nomenclature is included at the end that lists all these abbreviations and symbols.

[1] More information about the DELIVER project can be found at `http://natcomp.liacs.nl/index.php?page=DELIVER`

Chapter 2

Problem description

Routing problems like the Traveling Salesman Problem (TSP) and Vehicle Routing Problem (VRP) were already early considered in the research field of computer science. Both problems have a multitude of variants that all resemble different real-world problems. In this section we will give an introduction into different routing problems. At the end of this chapter we will introduce the Dynamic Vehicle Routing Problem with Time Windows (DVRPTW), which is the main focus of this thesis.

2.1 Traveling Salesman Problem

An instance of the TSP can be formally described as a fully connected undirected graph $G = (V, E)$ where $V = \{v_1, ..., v_n\}$ is the set of nodes and $E = \{(v_i, v_j) \mid v_i, v_j \in V\}$ is the set of edges connecting the nodes, each one with an associated traveling distance d_{ij}. Given G and the associated distances, the task is to find a path of minimal length that visits each node exactly once and returns to the first node. The standard TSP is also called the symmetric TSP because $d_{ij} = d_{ji}$, hence the use of an undirected graph in the above definition.

A solution is an n-tuple $\{\pi_1, ..., \pi_n\}$ containing each node from V. The length of this solution can be computed with Equation 2.1. Note that each solution is a cycle in G, after visiting all nodes we return to the first node.

$$\sum_{i=1}^{n-1} (d_{\pi_i, \pi_{i+1}}) + d_{\pi_n, \pi_1} \tag{2.1}$$

The analogy of this problem is that of a salesman that has to visit n customers and has to find the shortest route. It was supposedly formulated as a mathematical problem for the first time by Menger in 1928 [25]. Since then it has been one of the most intensively studied problems in optimization. The problem is NP-complete [23]. It is conjectured that there is no polynomial-time algorithm to solve such problems.

TSP is the most basic of the routing problems that we consider here. All the following problems extend on this idea and formal description.

2.2 Vehicle Routing Problem

The Vehicle Routing Problem (VRP), also called Capacitated Vehicle Routing Problem (CVRP), is a generalization of the TSP. The used analogy is that of a company with a set of vehicles that delivers goods to customers. Each vehicle has a maximum capacity Q and each customer v_i has a demand for a certain amount, q_i, of goods. The problem is to deliver the correct amount of goods to all customers without exceeding the capacity of each vehicle. Basically a VRP instance consists of multiple TSP instances because each vehicle is associated with a separate TSP tour.

Each VRP problem instance has a special node v_0, called the depot, which is the start and end of each tour. When the depot is visited mid-tour this is equivalent to starting with a new vehicle.

The objective in VRP can be to minimize the traveling distance or the amount of vehicles. In many articles both objectives are minimized simultaneously, though most implementations prioritize the objectives instead of solving a truly multi-objective problem. In [16] for example, the first goal is to minimize the amount of vehicles and the second goal is to minimize the traveling distance. This means that a solution with less vehicles is always preferred over a solution with a smaller traveling distance.

2.3 Vehicle Routing Problem with Time Windows

In the Vehicle Routing Problem with Time Windows (VRPTW), also called the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW), each node v_i has a corresponding time window $[e_i, l_i]$ describing the earliest beginning of service e_i and the latest beginning of service l_i. Also, each service requires a service time s_i. The distance matrix T now resembles the traveling time between each pair of nodes. Just like in the standard VRP, each node has a certain demand and vehicles have a maximum capacity.

The depot has a wide time window that starts at $e_0 = 0$ so that all nodes can be serviced before returning to the depot before l_0. Also it usually has a service time $s_0 = 0$. Note that it is possible to arrive at node v_i before time e_i. In this case servicing can only start at e_i so time will be spend waiting. A solution is only valid if the service at all nodes, including the depot, can be started before the end of their time windows.

To avoid confusion we want to explicitly state that in this report the traveling distance and the traveling time of a route are equivalent. Intuitively one might expect that the traveling time would include waiting times and service times. Most articles consider the two to be equal so we conform to that.

2.4 Dynamic Vehicle Routing Problem with Time Windows

The Dynamic Vehicle Routing Problem with Time Windows (DVRPTW), is the last variant we will introduce here and the main focus of this research. Because capacity constraints are also enforced it can also be called the Dynamic Capacitated Vehicle Routing Problem with Time Windows (DCVRPTW). The dynamic vehicle routing problem, sometimes called *on-line vehicle routing problem* or *real-time vehicle routing problem*, can be dynamic in different ways.

As described by Psaraftis [29], 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, we have to adjust the solution while the problem changes. Some implementations can deal with changing traveling times, others deal with the addition or deletion of nodes. We focus on the latter, the addition of nodes to an initial problem.

The a priori problem only contains part of all nodes in a problem. During a simulated working day of length T_{wd} the other nodes become visible to the algorithm. During the working day a tentative schedule is kept. Static problems also keep track of the global best solution found during the algorithm. The difference is that a tentative solution will most probably be unfeasible after some time because new nodes have become available. Also, part of the tentative solution can not be changed anymore because nodes have already been serviced at time t of the simulation.

For a more elaborate description of the dynamic VRP, its applications and different solution methods we refer to Pillac et al. [28].

2.4.1 Relevance to this work

Summarizing, we solve the Dynamic Capacitated Vehicle Routing Problem with Time Windows. We therefor have to deal with:

- a maximal capacity Q of each vehicle;
- a time window $[e_i, l_i]$ on each node v_i;
- a service time s_i on each node v_i;
- the dynamic addition of nodes to the problem.

A solution is a tuple $\{\pi_1, ..., \pi_x\}$ with a maximal length of $2n - 2$, containing all n nodes from V and at most $n - 2$ copies of the depot node. The traveling distance of a solution can be computed with

$$\sum_{i=1}^{x-1}(d_{\pi_i,\pi_{i+1}}) + d_{\pi_x,\pi_1} \tag{2.2}$$

where x is the number of nodes in the solution.

How the dynamic problem can be simulated and solved is described in detail in Section 5.1.

Chapter 3

Ant Colony Optimization

Ant Colony Optimization (ACO) is a meta-heuristic that is based on the natural behavior of ants. It was introduced by Colorni, Dorigo and Maniezzo [7], [10], [13] and inspired by an experiment of Gross et al. [19]. The experiment dealt with the way that Argentine ants (*Iridomyrmex humilis*) search for food. The ants were given a food source that they could reach via two bridges of unequal lengths as depicted in Figure 3.1. The observed behavior was that, after a short while, most ants were traveling along the shorter bridge.

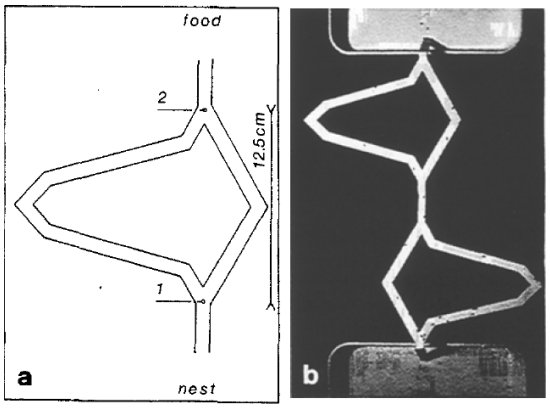


Figure 3.1: a) Schematic representation of the experiment by Gross et al. [19] with choice points 1 and 2. b) Photo of the test setup.

Ants accomplish this task with a small set of simple rules. When an ant is foraging it has to make the decision of which way to go. This choice is a probabilistic one that is influenced by the amount of pheromones that have been deposited on the path. Ants tend to choose the path with higher pheromone levels with a higher probability. On its way to the food and on the way back to the nest each ant deposits pheromones, thereby increasing the amount of pheromones on the specific route that it has traveled.

With this simple mechanism ants find the shorter route in most of the performed experiments. This is achieved by the self-organization process of depositing and following pheromone trails. When the experiment starts the choice of using the longer or the shorter bridge is random since no pheromones are deposited. Ants that choose the shorter bridge will arrive at the food earlier and deposit pheromones on their path. When these ants return to the nest the only pheromones deposited at choice point 2 in Figure 3.1a lead to the shorter bridge. They will use the shorter bridge on their way back with a high probability and deposit pheromones on their way back. The pheromones on the shortest path are enforced more and will dominate other paths.

This way, by depositing pheromones ants can communicate to one another. This process of communication by changing the environment is called stigmercy. This is the basis for ACO. It is important to

keep in mind that there is no central control in the colony, every ant acts as an individual agent that takes individual decisions. All ants together exhibit intelligent behavior, a phenomena known as emergent behavior.

The pheromone trails form a collective memory that every ant uses and to which every ant contributes. When an individual finds food pheromone will be stored in the trails. If the route is fast the pheromone trail will be reinforced, effectively adjusting the collective memory to inform others that the route is good.

3.1 Canonical ACO algorithm

Let us now look at the outline of the ACO optimization algorithm that is inspired by real ants. The pseudo code for a canonical ACO algorithm can be found in Algorithm 1.

Algorithm 1 Canonical ACO

```
initialize pheromones
repeat
    for k = 1 to number of ants m do
        start tour of ant k
        for i = 1 to number of reachable nodes n do
            choose next node of tour
        end for
        local search method (optional)
        local pheromone update (optional)
    end for
    evaluate tours
    update best solution found so far
    global pheromone update
until stop condition
return best solution found
```

During initialization all pheromone levels are set to an initial value. After initialization the main loop of the algorithm starts. One after another, each of the m ants constructs a route that solves the problem. The ant is positioned at an initial node, which may be randomly chosen but for the VRP problem should be the depot. Then it probabilistically chooses the next node to incorporate in its route until all nodes are visited. 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 transition rule in Equation 3.1.

$$p_j^k(v_i) = \begin{cases} \dfrac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{m \in N_i^k} [\tau_{im}]^\alpha \cdot [\eta_{im}]^\beta} & \text{if } j \in N_i^k \\ \\ 0 & \text{if } j \notin N_i^k \end{cases} \tag{3.1}$$

with:

τ_{ij}	:	pheromone level on edge (i, j)
η_{ij}	:	heuristic desirability of edge (i, j)
α	:	influence of τ on the probabilistic value
β	:	influence of η on the probabilistic value
N_i^k	:	set of nodes that can be visited by ant k positioned at node v_i

and $\tau_{ij}, \eta_{ij}, \alpha, \beta \geq 0$.

During the construction of a tour, the set N_i^k contains all nodes that are reachable from the current node and are not yet visited. Nodes that are unavailable or that can not be reached due to constraints are not part of N_i^k. This ensures that no node is visited twice and only feasible solutions are created. The heuristic desirability η_{ij} is a way to incorporate problem specific knowledge into the probabilities. One of the most commonly used settings for the TSP is $\eta_{ij} = 1/d_{ij}$ where d_{ij} is the length of edge (i, j). This heuristic favors shorter edges because they will get a higher value for η and thus a higher value of p_j^k. Many different heuristics have been used for different problems.

Once an ant has constructed a route, most approaches apply a local search algorithm to improve the generated solution with some problem specific knowledge. A well-known local search method for the TSP is 2-opt [8]. This method optimizes tours by eliminating crossing edges. Section 3.3.6 deals with some common local search methods.

When all ants have constructed a tour, these tours can be evaluated after which the pheromone trails are adjusted. All pheromone trails are first decreased with a factor $(1 - \rho)$ to simulate the evaporation of existing trails. Then pheromone is added to edges that are part of tours that were found in the last iteration of the algorithm using Equations 3.2 and 3.3.

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{3.2}$$

$$\Delta\tau_{ij}^k = \begin{cases} 1/L_k & \text{if } (i, j) \in T_k \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

with:

ρ	:	pheromone evaporation parameter
m	:	number of ants
$\Delta\tau_{ij}^k$	:	amount of pheromones deposited by ant k on edge (i, j)
T_k	:	tour found by ant k
L_k	:	length of tour T_k

and $0 \leq \rho \leq 1$ and $\Delta\tau_{ij} \geq 0$.

After this pheromone update the process can be repeated, starting with new ants to discover new routes. Newly discovered routes can be expected to be better than older ones because the ants are guided to more promising solutions by the pheromone trails. Some convergence proofs for ACO algorithms are given by Gutjahr [22] and Stützle and Dorigo [33]. A more detailed discussion on the convergence of ACO algorithms can be found in Section 3 of [11].

The main loop of the algorithm runs until some stopping condition is met. This condition might be a maximum number of iterations, a certain time or until a reasonable solution is found.

3.2 Variants

Many variants have been build on the first ACO algorithm that was proposed. A short review is given below. The main goal of this overview is to show the diversity in the different approaches, not to give an exhaustive overview.

3.2.1 Ant System

The approach that was described in Section 3.1 is almost exactly the same as the Ant System (AS) proposed in [13] by Dorigo, Maniezzo and Colorni. The AS did not use any local optimization or local pheromone updates. All ants contributed to the pheromone trails, based on the Equations 3.2 and 3.3. It is the very first ACO algorithm.

3.2.2 Elitist Ant System

In [13], Dorigo et al. proposed not only the AS but also the Elitist Ant System (EAS). In this strategy all ants deposit pheromones, but the pheromone levels on the edges from the best tour found so far are enforced by elitist ants. The pheromone update rules are given in Equations 3.3, 3.4 and 3.5. Besides the standard pheromone, σ ants deposit pheromones on the edges that belong to T^*, the best solution found so far. The length of T^* is denoted with L^*.

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \sum_{k=1}^{m}\Delta\tau_{ij}^{k} + \sigma\Delta\tau_{ij}^{*} \tag{3.4}$$

$$\Delta\tau_{ij}^{*} = \begin{cases} 1/L^{*} & \text{if } (i,j)\in T^{*} \\ \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

3.2.3 Ant Colony System

The Ant Colony System (ACS) is the first major variation on the AS. Proposed in [12] by Dorigo and Gambardella, the main adjustment is to focus ants on promising parts of the search space. This is done by introducing a local pheromone update and by adjusting the global pheromone update function and transition function.

The ACS uses a local pheromone update to decrease pheromone levels on edges that are traversed by ants. Each time an ant has traversed an edge (i,j), it applies Equation 3.6. 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.

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \rho\cdot\tau_{0} \tag{3.6}$$

with:

τ_0 : initial pheromone value

The global pheromone update rule given in Equations 3.2 and 3.3 are changed slightly as well. 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. This changes Equations 3.2 and 3.3 to Equations 3.7 and 3.8.

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \rho\cdot\sum_{k=1}^{m}\Delta\tau_{ij}^{k}\,, \quad \forall(i,j)\in T^{*} \tag{3.7}$$

$$\Delta\tau_{ij}^{k} = 1/L^{*} \tag{3.8}$$

with:

T^* : best tour found so far
L^* : length of T^*

The transition rule that is used to determine the next node in the tour is altered as well. The AS transition rule, given in Equation 3.1, is only applied with a certain probability q_0. The new function is given by Equation 3.9.

$$p_{j}^{k}(v_i) = \begin{cases} \underset{j\in N_i}{\arg\max}\{[\tau_{ij}]^{\alpha}\cdot[\eta_{ij}]^{\beta}\} & \text{if } q\le q_0 \\ \\ \text{Equation 3.1} & \text{otherwise} \end{cases} \tag{3.9}$$

with:

q : uniform random number on $[0, 1]$
q_0 : parameter that determines the balance between exploitation and exploration, and $0 \leq q_0 \leq 1$

In essence, every ACO algorithm uses the transition rule described by Equations 3.9. When it is not explicitly stated we assume that q_0 is set to 0.

3.2.4 Max-Min Ant System

In the year 1997 Stützle and Hoos [35] proposed the MAX-MIN Ant System (MMAS). Just like the ACS it was meant to increase exploitation and avoid early stagnation. To accomplish this, four adjustments to the AS were proposed.

First the amount of pheromones on edges are limited to the interval $[\tau_{min}, \tau_{max}]$. These boundaries make sure that pheromone trails do not totally evaporate. This increases the changes of certain edges being incorporated in a route, leading to more exploration. Because of this, and because of the upper bound, no single route should be able to dominate other routes. The second and third adjustment are that pheromone trails are initialized at τ_{max} and reinitialized to τ_{max} when the algorithm is stagnating.

Lastly, like in the ACS, only one ant is allowed to update pheromone trails in each iteration. Other than in the ACS variant, in MMAS pheromone evaporation is done on all edges. The adjusted pheromone update function can be seen in Equations 3.10 and 3.11.

$$\tau_{ij} = \rho\, \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij} \tag{3.10}$$

$$\Delta\tau_{ij} = \begin{cases} 1/L^* & \text{if } (i,j) \in L^* \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

The values for τ_{min} and τ_{max} are problem dependent. A technical report [34] describes how to estimate the values. In a later article from them same authors [36] default values of $\tau_{max} = 1/\rho L^*$ and $\tau_{min} = \tau_{max}/2n$ are proposed.

For long runs, it was found that a technique called *smoothing* could help to prevent stagnation. Smoothing involves increasing all pheromone values. The amount of pheromones deposited on each edge is proportional to $\tau_{max} - \tau_{ij}$. This method adds little pheromone to strong pheromone trails, but it adds more pheromone to weak trails. So the order among the trails is kept but the pheromone levels on all of them are increased. This again enforces exploration and should prevent the algorithm from ending up in a local optimum.

3.2.5 Rank Based Ant System

Bullnheimer, Hartl and Strauss [4] proposed variations on the Ant System that use elitist ants and a ranked based approach. First they propose a strategy that enforces the pheromones on edges of the best tour, much like in [13] described in the Section 3.2.2.

After this they propose a new variation called the Rank Based Ant System (RBAS). The global pheromone update of the EAS is adjusted so that each ant deposits pheromones proportional to the rank of its route. The ant with the best tour deposits most pheromones, the second best ant deposits slightly less, the third ant even less, and so on. The new global pheromone update function is described in Equations 3.12 and 3.13.

$$\begin{aligned} \tau_{ij} &= \rho \cdot \tau_{ij} + \Delta\tau_{ij} + \Delta\tau^*_{ij} \\ \Delta\tau_{ij} &= \sum_{\mu=1}^{\sigma-1} \Delta\tau^{\mu}_{ij} \end{aligned} \tag{3.12}$$

$$\Delta\tau_{ij}^{u} = \begin{cases} (\sigma - \mu) \cdot \dfrac{1}{L_u} & \text{if } (i,j) \in T_u \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

$$\Delta\tau_{ij}^{*} = \begin{cases} \sigma \cdot \dfrac{1}{L^*} & \text{if } (i,j) \in T^* \\ 0 & \text{otherwise} \end{cases}$$

with:

σ	:	the number of elitist ants
μ	:	the rank of each ant
T_μ	:	tour found by ant with rank μ
T^*	:	best tour found
L_μ	:	length of T_μ
L^*	:	length of T^*

and $\sigma = n = m$. Recall that n stands for the number of nodes in the problem and m is the number of ants in the colony.

3.2.6 Population Based Ant System

A population bases approach for ACO was proposed by Guntsch and Middendorf in [21, 20]. The population P consists of k tours and can be controlled using different strategies. The approach is similar to the MMAS. The pheromone values have a minimal and a maximal value.

The difference lies in the fact that pheromone evaporation occurs in a different way. When an ant enters P it deposits pheromones and when it leaves P the same amount of pheromones is evaporated. Pheromones are initialized at τ_0. For convenience Guntsch and Middendorf have set τ_{max} to 1. When a tour enters P it deposits $\Delta\tau = (\tau_{max} - \tau_0)/k$ pheromones on the edges visited in the tour.

This way of updating pheromones has two advantages over the traditional pheromone update methods. Firstly it is less computationally intensive. Instead of updating pheromones on all edges after each iteration we now only update pheromones on edges belonging to the two tours that enter and exit the population. Secondly this approach might be more suitable for solving dynamic problems. The pheromone matrix contains a lot of information but at least part of this information stems from old solutions. In a dynamic problem it would be better to base actions on more recent information and with a population based approach it would be easy to store the k last adjustments to the pheromones.

To determine which individuals enter and leave the population, multiple population update strategies have been tested. These are similar to techniques used in genetic algorithms which also keep track of a population. Some solutions that were proposed in [20] are:

- Age: the oldest solution in P is replaced by the best solution from the current iteration of the algorithm. A first implementation from [21] that used this strategy was called the FIFO-Queue ACO algorithm.
- Quality: if the best solution from the current iteration is better than the worst solution in P, then the new solution will enter P and the worst solution will be removed from P.
- Probabilistic: a function probabilistically selects one of the solutions in P unified with the new solution. The chosen solution will not be part of P in the next iteration, it will be formed by all unselected solutions. The probability of choosing a solution of bad quality should be bigger than choosing solutions of better quality, but the strength of this strategy is that every solution can be deleted from P. This reduces the chance that P will be populated by copies of a single best solution.

- Elitism: the best solution found so far is always kept in the population.
- Mixed strategy: strategies can be combined to form a new strategy. In [20] an Age & Probability strategy was applied to solve dynamic problems.

3.2.7 Multi-objective ACO

Barán and Schaerer [1] proposed an ACO algorithm that optimizes multiple objectives to find good solutions to the VRPTW problem. Their work is an improved version of the MACS-VRPTW algorithm proposed in [16] by Gambardella, Taillard and Agazzi. The MACS-VRPTW algorithm tries to optimize two objectives: the amount of vehicles and the total traveling time should both be minimized. But because a lexicographical ordering is used we can not speak of a multi-objective approach. Barán and Schaerer have overcome this by using the Pareto preference relation, which is commonly used in multi-objective optimization.

Barán and Schaerer have specified three objectives, that they try to optimize at the same time. These objectives are the number of vehicles, the traveling time and the total delivery time (traveling time + waiting time). Each tour that is constructed by the ants is compared on the three objective functions. If a solution is not (Pareto) dominated by other solutions it enters the Pareto optimal set P and all members of P that are now dominated are deleted. Because all objectives are equally important, the solutions in P are incomparable. P is not limited in size so that a new non-dominated solution can always be added.

Whenever P is found to be altered at the end of an iteration, the pheromones are reinitialized. This procedure should improve exploration. It forces the ants to explore new routes and makes sure they do not follow false information, e.g., pheromones deposited by routes that are erased from P.

Pareto optimal sets that are found can be compared on different measurements of the Pareto front, for example the overall non-dominated vector generation or the overall non-dominated vector ratio. It goes beyond the scope of this work to get into detail on these measurements, interested readers can refer to [1] for more information and further references.

3.2.8 Decomposition ants

Reimann, Doerner and Hartl [30] proposed the D-Ants (short for Decomposition Ants) in 2004 to solve the VRP. The basis for their algorithm is the Savings based Ant System proposed by Doerner [9] which uses the savings value as a heuristic value, and candidate lists based on the same savings values. The savings value s_{ij} for visiting node v_j after node v_i can be computed with Equation 3.14 and was proposed in 1964 by Clarke and Wright [6].

$$s_{ij} = d_{i0} + d_{0j} - d_{ij} \tag{3.14}$$

The algorithm splits the route construction from the route optimization. Basically one can say that at first they determine the number of trucks and which nodes a truck visits. After that each individual route is optimized. In short the following steps are taken to do this:

1. Run the Savings based Ant System (SbAS) x times. Save the single best solution that is found.
2. Find the center of gravity for each of the v vehicles from the best solution found in step 1.
3. Make i sub-problems using the Sweep algorithm [18] and the centers of gravity from step 2. The parameter i is predefined.
4. Solve each sub-problem by running the SbAS j times. The parameter j is predefined.
5. Repeat from step 1 until stopping condition is fulfilled.

Two kinds of local search are used. The first is a swap algorithm that exchanges nodes between tours. The second one is 2-opt that optimizes an individual tour.

The algorithm uses one pheromone matrix to store all pheromone information. The pheromones are updated when a new best solution is found for one of the sub-problems. Only the solutions of sub-problems contribute to the pheromone trails but the algorithm of step 1 can also use this information to find a better initial solution.

3.3 Extensions

Now that we have seen some variations on the basic ant colony algorithm, a short overview will be given of some extensions to these algorithms. In particular some problem specific solutions, tailored for routing problems will be discussed.

3.3.1 Pheromone initialization

At the start of most ACO algorithms the pheromone matrix is initialized. The standard way to do this is to determine the initial pheromone level τ_0 to which all pheromone levels are set. τ_0 can be set to an arbitrary but low value, like in [7], but there are different methods for estimating a good value.

A simple but effective way to compute the value of τ_0 is by finding an initial solution with a simple heuristic. If we would have to solve TSP, it would be good to find an initial solution with the Nearest Neighbor heuristic [15]. This will result in a reasonable and feasible solution. Although this solution will probably be improved by the ACO algorithm, the length of this initial tour can be used to initialize pheromones to a reasonable value with Equation 3.15 as used in [16].

$$\tau_{ij} = \frac{1}{n \cdot L_{\text{NN}}} \tag{3.15}$$

with:

n	:	number of nodes
L_{NN}	:	length of Nearest Neighbor tour

3.3.2 Heuristic values

The heuristic value η_{ij} that was first introduced in Equation 3.1 should reflect the desirability of incorporating v_j in the tour after the current node v_i. This heuristic should be used to incorporate problem specific knowledge. Some commonly used values for solving routing problems are:

- Length of edge (i, j): $\eta_{ij} = 1/d_{ij}$. Used for solving the TSP using an AS in [7].
- The savings value: $s_{ij} = d_{i0} + d_{0j} - d_{ij}$, introduced by Clarke and Wright [6] and used in [30] to solve the Vehicle Routing Problem. It is a measure that compares visiting v_j directly after v_i and visiting the depot in between these nodes.
- Parametric savings function: $\eta_{ij} = d_{i0} + d_{0j} - g \cdot d_{ij} + f \cdot |d_{i0} - d_{0j}| = s_{ij} - (g-1)d_{ij} + f|d_{i0} - d_{0j}|$, introduced by Paessens [27] and used by Bullnheimer et al. in [5]. This alternative savings function should yield better results, at the cost of optimizing two extra parameters. Paessens suggested to use $0 \leq g \leq 3$ and $0 \leq f \leq 1$.
- Capacity utilization: $\kappa_{ij} = (Q_i + q_j)/Q$, introduced by Bullnheimer et al. [3]. Q_i stands for the current load of vehicle i. Higher values for κ_{ij} indicate that the vehicle will be fuller after visiting node j, compared to nodes that yield lower values for κ.
- The number of times that a customer is not incorporated in a route. This only occurs when time windows form a hard boundary and being late is no option. It might be infeasible to visit some customers (in time), due to their constraints. If a customer can not be incorporated in many consecutively created routes this could be a reason to raise its attractiveness. This idea was incorporated in the heuristic that was used in the MACS-VRPTW algorithm by Gambardella et al. [16].

- Basic features of available nodes such as:
  - Begin of service: $begin_j$ like introduced in Section 2.3.
  - Urgency: end of time window l_j- arrival at node $arrival_j$. One could easily adjust this to resemble the tardiness that is introduced by visiting this customer at this point of the route: l_j- current time $-d_{ij}$.
  - Waiting time: the amount of time waiting at a node, before service can begin.

The list given above is not complete but should give an idea of what can be done. Some of these heuristics, like the Savings values, are powerful enough to be used as the only heuristic. Others have been combined to form a single number, like in [5]. Another option is to use multiple heuristics, each of which can have its on parameter. Equation (3.16) shows the transition rule of [3] by Bullnheimer et al., which is a good example of using multiple heuristics with their own parameter. Of course there is a trade-off between the strength of the heuristic and the amount of parameters that are introduced and need to be optimized.

$$p_j^k = \begin{cases} \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta \cdot [\mu_{ij}]^\gamma \cdot [\kappa_{ij}]^\lambda}{\sum_{m \in N_i} [\tau_{im}]^\alpha \cdot [\eta_{im}]^\beta \cdot [\mu_{ij}]^\gamma \cdot [\kappa_{ij}]^\lambda} & \text{if } j \in N_i \\ 0 & \text{if } j \notin N_i \end{cases} \tag{3.16}$$

3.3.3 Candidate lists

Candidate lists are an extension that can limit the number of nodes that can be reached from every other node. If we set the size of the candidate list at an arbitrary number x, we will never consider more than the x most attractive nodes. The use of candidate list can greatly improve the performance of the algorithm since we consider less options and therefore need less computations.

To decide which x nodes are most attractive and should form a candidate list we can use different values. An important thing to consider is that the chosen value should be computed only once and stored for future use. If the used value is not static, but depends for example on time window constraints, the candidate list shall need to be recomputed at every iteration which neutralizes the positive effects of using a candidate list. Most implementations use the traveling distance [12, 2, 5] or savings value [9, 30] to build candidate lists.

The use of candidate lists adds one more parameter to the algorithm, the size of the lists. Bell and McMullen [2] investigated how to determine the appropriate size by testing neighborhood sizes depending on problem sizes, e.g., the number of nodes in a problem. The results seem to indicate that a fixed size between 10 and 20 works best. Using a list of size 20 is also proposed by Dorigo and Gambardella [12] in their work on the TSP.

3.3.4 Pheromone adjustment techniques

Eyckelhof and Snoek [14] proposed an algorithm for the dynamic TSP in which the traveling times between nodes were made dynamic in order to simulate traffic jams. To cope with these changing traveling times they proposed ways of adjusting the pheromone levels.

A method called *shaking* is introduced to adjust pheromone levels. This method lowers pheromone levels on all edges (global shaking) or just on edges near a traffic jam (local shaking). All pheromone levels are lowered, but the hierarchy among edges is kept. An initial pheromone level τ_0 is used. The shaking method decreases all pheromones to τ_0 but high values are decreased more than low values. This technique is similar to the *smoothing* discussed in Section 3.2.4 on MMAS.

The authors also suggest to take a look at changing the values of α and β (parameters of pheromones and heuristics in the probability function) after a change in the problem. This could be done locally or globally.

3.3.5 Multiple colonies

Several authors have proposed algorithms that use multiple ant colonies to solve the VRP. The idea is that different colonies can work more efficient because they have their own pheromone trails. This means that each colony can focus on its own problem and is not distracted by pheromones trails that belong to other parts of the problem.

Bell and McMullen [2] use separate colonies to optimize the route of each vehicle in the VRP. Their results indicate that the multiple colony approach found better results on average. They also show that the use of multiple colonies decreased the CPU time that was needed to find the best results.

3.3.6 Local search

Local search methods are simple heuristics that are used in ACO algorithms to improve on routes that have been found. Because they are not inspired by ants they are no real extensions of the ACO algorithms. It is better to think of them as an ideal combination. The ant colony builds initial solutions which the local search methods can fine-tune. Because the use of local search methods dramatically improves results almost all implementations use them.

The most common methods are *k-interchange* methods. These replace k edges from the graph by k different edges if and only if this improves the route. If it is impossible to improve the tour by replacing k edges, the tour is said to be *k-optimal*. Because the computational complexity of the algorithms increases with the size of k it is only practical to check for $k = 2$ (2-opt) or $k = 3$ (3-opt).

2-opt was first introduced by Croes [8] and is depicted in Figure 3.2. The edges (X, X') and (Y, Y') are replaced by the edges (X, Y) and (X', Y'). The edges that are replaced might be positioned in different tours like in Figure 3.2a or within a single tour like in Figure 3.2b. This replacement only yields an improvement if Equation 3.17 holds and is therefore very easy to check.

$$d_{X,X'} + d_{Y,Y'} > d_{X,Y} + d_{X',Y'} \tag{3.17}$$

The 3-opt method is an extension in which 3 edges are reconnected, which can be done in 8 different ways. An important problem with applying 2-opt and 3-opt is that it might change the orientation of an edge. An example can be found in Figure 3.2b, where 2-opt is applied within a single tour. The orientation of the sub-tour Y, X', is reversed and this would almost certainly lead to a violation of time window constraints.

Some local search methods are specifically constructed with time window constraints in mind. One example is Cross Exchange, proposed in [37] by Taillard et al. It exchanges two sub-tours that can be situated within the same tour different tours. Also, one of the two sub-tours may be empty so that the result is a sub-tour insertion. Because the orientation of the sub-tours does not change this method is more suitable for problems with time window constraints. An example can be found in Figure 3.3.

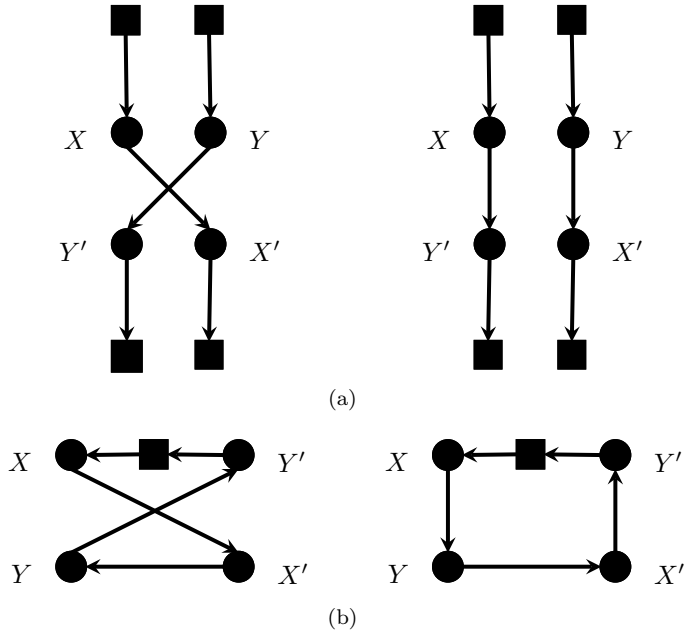


Figure 3.2: Examples of 2-opt edge replacements. Squares represent depots, circles represent nodes. (a) demonstrates a move with edges from different tours. (b) is an example of a move within a single tour.

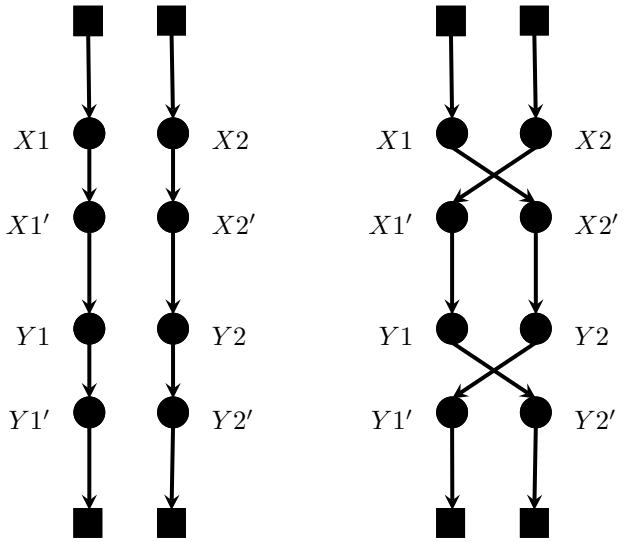


Figure 3.3: Example of Cross Exchange. Squares represent depots, circles represent nodes. The sub-tours $X1'Y1$ and $X2'Y2$ are swapped. These sub-tours vary in length and one of them may be empty.

Chapter 4

MACS-VRPTW

In the next chapter we propose an ACO algorithm that solves the Dynamic Vehicle Routing Problem with Time Windows. This chapter describes the implementation that we will take as the basis for the new implementation: the MACS-VRPTW algorithm proposed by Gambardella, Taillard and Agazzi [16].

The MACS-VRPTW algorithm solves the VRPTW problem with great success and is still a state-of-the-art ACO algorithm[1]. To our knowledge there is no ACO algorithm that yields better results on the VRPTW problem. Because the DVRPTW problem is an extension of the VRPTW problem this algorithm should be a good basis. It will also give us a good source for results with which we can compare our own algorithm. The description and results will be reported in detail because implementing certain parts of the algorithm on basis of the original description turned out to be difficult.

The next section describes the MACS-VRPTW algorithm in detail. Section 4.2 shows our results on the Solomon benchmark problems to verify that our implementation matches that of Gambardella et al.

4.1 Algorithm description

The MACS-VRPTW algorithm consists of three basic parts: the controller, the ACS-TIME colony and the ACS-VEI colony. Each of these parts is first described in more detail. After that the problem representation and the most important functions are explained.

4.1.1 The controller

The controller is the central part that reads the benchmark data, initializes data structures, builds an initial solution and starts the colonies. The colonies then try to find solutions that improve the global best solution found so far, T^*. If a solution with less vehicles is found the controller restarts the colonies. The controller stops the colonies when the stop condition, which was originally set to 100 seconds of CPU time, is fulfilled. A pseudo-code of the controller can be seen in Algorithm 2.

4.1.2 The colonies

Each one of the colonies 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 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

[1] see `http://web.cba.neu.edu/~msolomon/heuristi.htm` for best results on the Solomon benchmark problems.

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.

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 3 and 4 respectively.

4.1.3 Problem representation

An important part of the 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. The number of copies is the same as the maximum number of vehicles allowed, which is different for ACS-TIME and ACS-VEI. By copying the depots we allow the ants to deposit different pheromone levels to different depots, which enables them to find better solutions. How we use this problem representation can be seen in the ConstructTour method described next.

4.1.4 Constructing a tour

Algorithm 5 shows a pseudo-code of the ConstructTour method used by the colonies. Each 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 for ant k situated at node i. Nodes that cannot be visited within capacity or time window constraints are not part of $\mathcal{N}_i^k$.

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 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 if 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.

4.1.5 Nearest neighbor heuristic

The nearest neighbor heuristic [15] 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.

4.1.6 Inserting missing nodes

When a tour has been created for each ant, a heuristic is applied to insert missing nodes into these tours. All missing nodes are ordered by decreasing delivery quantities. Then for each node the best place of insertion is found, based on traveling distance and given that no constraints are violated.

4.1.7 Cross Exchange local search

The ACS-TIME colony uses a variant of the Cross Exchange local search method from Taillard et al. [37]. What method they exactly used is unknown but the main difference with [37] will be that MACS-VRPTW deals with hard time window constraints instead of soft time window constraints, for which it was originally used. Two sub-tours are exchanged, as illustrated in Figure 3.3. One of the sub-tours can be empty which will result in a sub-tour insertion. Also, the two sub-tours can be part of different vehicles or the same one. A pseudo-code for Cross Exchange can be found in Algorithm 6.

Algorithm 2 Controller

Given: n is the number of nodes

$T^* \leftarrow$ NearestNeighbor(n)
$\tau_0 \leftarrow 1/(n \cdot$ length of $T^*)$

repeat
    Start ACS-TIME(vehicles in T^*) in new thread
    Start ACS-VEI(vehicles in $T^* - 1$) in new thread

    while colonies are active **do**
        Wait until a solution T is found
        if vehicles in $T <$ vehicles in T^* **then**
            Stop threads
        end if
        $T^* \leftarrow T$
    end while
until stop condition

return T^*

Algorithm 3 ACS-VEI(v)

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

Initialize pheromones to τ_0
Initialize IN to 0
$T^{\text{VEI}} \leftarrow$ NearestNeighbor(v)

repeat
    for each ant k **do**
        $T^k \leftarrow$ ConstructTour(k, IN)
        for each nodes $i \notin T^k$ **do**
            $\text{IN}_i = \text{IN}_i + 1$
        end for
        Local pheromone update on edges of T^k using Equation 3.6
        $T^k \leftarrow$ InsertMissingNodes(k)
    end for

    Find ant l with most visited nodes
    if nodes in $T^l >$ nodes in T^{VEI} **then**
        $T^{\text{VEI}} \leftarrow T^l$
        Reset IN to 0
        if T^{VEI} is feasible **then**
            return T^{VEI} to controller
        end if
    end if

    Global pheromone update with T^* and Equation 3.7
    Global pheromone update with T^{VEI} and Equation 3.7
until controller sends stop signal

Algorithm 4 ACS-TIME(v)

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

Initialize pheromones to τ_0

repeat
    for each ant k **do**
        $T^k \leftarrow$ ConstructTour(k, 0)
        Local pheromone update on edges of T^k using Equation 3.6
        $T^k \leftarrow$ 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**
        return T^l to controller
    end if

    Global pheromone update with T^* and Equation 3.7
until controller sends stop signal

Algorithm 5 ConstructTour(k, IN)

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

Select a random depot duplicate i
$T^k \leftarrow \langle i \rangle$
current time$_k \leftarrow 0$
load$_k \leftarrow 0$

repeat
    for each $j \in \mathcal{N}_i^k$ **do**
        delivery time$_j \leftarrow \max($current time$_k + t_{ij}, e_j)$
        delta time$_{ij} \leftarrow$ delivery time$_j -$ current time$_k$
        distance$_{ij} \leftarrow$ delta time$_{ij} \times (l_j -$ current time$_k)$
        distance$_{ij} \leftarrow \max(1.0, ($distance$_{ij} -$ IN$_j))$
        $\eta_{ij} \leftarrow 1.0/$ distance$_{ij}$
    end for

    Pick node j using Equation 3.9
    $T^k \leftarrow T^k + \langle j \rangle$
    current time$_k \leftarrow$ delivery time$_j +$ service time$_j$
    load$_k \leftarrow$ load$_k + q_j$
    if j is a depot copy **then**
        current time$_k \leftarrow 0$
        load$_k \leftarrow 0$
    end if
    $i \leftarrow j$
until $\mathcal{N}_i^k = \{\}$

return T^k

Because Cross Exchange can occasionally produce a tour with an empty vehicle there is a small repair operation afterwards. This checks the tour for consecutive depot copies and deletes one of these when they are found.

Note that Cross Exchange should only be executed on feasible tours because otherwise the resulting tour will also be infeasible and the computational effort will be wasted.

4.2 Results

MACS-VRPTW was tested on the 56 benchmark problems by Solomon [32]. 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.

The tests were performed using the default parameters given by Gambardella et al. and listed in Table 4.1.

Parameter	**Value**
m	10
α	1
β	1
q_0	0.9
ρ	0.1

Table 4.1: Default parameter values for the MACS-VRPTW algorithm.

The results of MACS-VRPTW and our own implementation can be found in Table 4.2. Gambardella et al. ran their algorithm 3 times on each problem and then averaged the results over all runs per benchmark category. These results can be found in the first row. We ran our algorithm 10 times on each problem. We report the averaged results over all runs in the row labeled 'Avg'. We also report the best values per benchmark problem, averaged per benchmark category, in the row labeled 'Best'.

Our implementation was executed on a Intel Core i5, 3.2GHz CPU with 4GB of RAM memory.

	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 4.2: Comparison of results between the original MACS-VRPTW and our implementation.

From Table 4.2 we can see that on average our algorithms performs as good as the original one. On some problems we score less good, on others we perform better. We have noticed that getting good results consistently is hard on some of the benchmark problems, which is why some of our results are not as good as reported for the original implementation. If we look at our best results we see that these are much better, it outperforms the original algorithm on all problem categories.

Even though our implementation achieves similar results, we can not state the we have implemented exactly the same algorithm. There are two reasons for this: the local search method is not clearly described and the stop condition is too vague.

Algorithm 6 LocalSearch(k)

score = length(T^k)
bestscore = -1
for X1 = first depot → last depot **do**
    for Y1 = X1.next → last depot **do**
        if length of sub-tour X1-Y1 > 3 **then**
            break
        end if
        for X2 = Y1.next → last depot **do**
            for Y2 = Y2.next → last depot **do**
                if length of sub-tour X2-Y2 > 3 **then**
                    break
                end if
                newscore = score + edges that will be added - edges that will be removed
                if (newscore < bestscore) ∨ (bestscore < 0) **then**
                    $T^{k'}$ ← exchange sub-tours in T^k
                    if constraints are met **then**
                        bestscore ← newscore
                        $T^{best} \leftarrow T^{k'}$
                    else
                        newscore = -1
                    end if
                end if
            end for
        end for
    end for
end for
if (bestscore > 0) ∧ (bestscore < score) **then**
    $T^k \leftarrow T^{best}$
end if
return T^k

The Cross Exchange method is clearly described in [37]. In MACS-VRPTW a variant is used because it deals with hard time window constraints and Cross Exchange was originally proposed for soft time window constraints. An essential part of the local search consists of ways to limit the search, so that less promising moves will not be tested. This saves a lot of time and makes the overall algorithm perform much better. Because Gambardella et al. have not described what methods they have used to limit the search we have chosen to only allow sub-tours with a maximum length of 3 nodes.

The stop condition of the algorithm is set to 100 seconds of CPU time. This means that not only the strength of that algorithm is important, but also the efficiency of the programming code with which the algorithm is implemented. Because of this it is impossible to say if results are achieved because the algorithm is good or because the implementation is fast. An efficient implementation could perform many more iterations in the same amount of CPU time and therefore achieve better results.

Averaged over all the benchmark problems our approach is comparable with MACS-VRPTW. If we look at individual categories the results are somewhat less consistent, and if we look at individual benchmark problems this difference is even bigger. Because of reasons stated above this can be caused by either a different implementation or by a less efficient implementation.

Difference between benchmark problems

In [16] it is reported that MACS-VRPTW performs especially good on C1- and RC2 problems. Also it is clear that MACS-VRPTW finds good results in a very short amount of time compared to other algorithms they compared with.

Our results deviate a bit from the original results and we think this is due to the performance on a few very specific benchmark problems. It seems like our implementation finds good results for C and RC problems but not for R problems. Figure 4.1 shows the development of the average score over time for an easy benchmark problem (R101) and a hard benchmark problem (R202). It is clear that the R101 problem can be solved more consistently and in far less iterations. The R202 problem is hard to solve and the resulting scores therefor more than on the R101 problem.

Optimizations

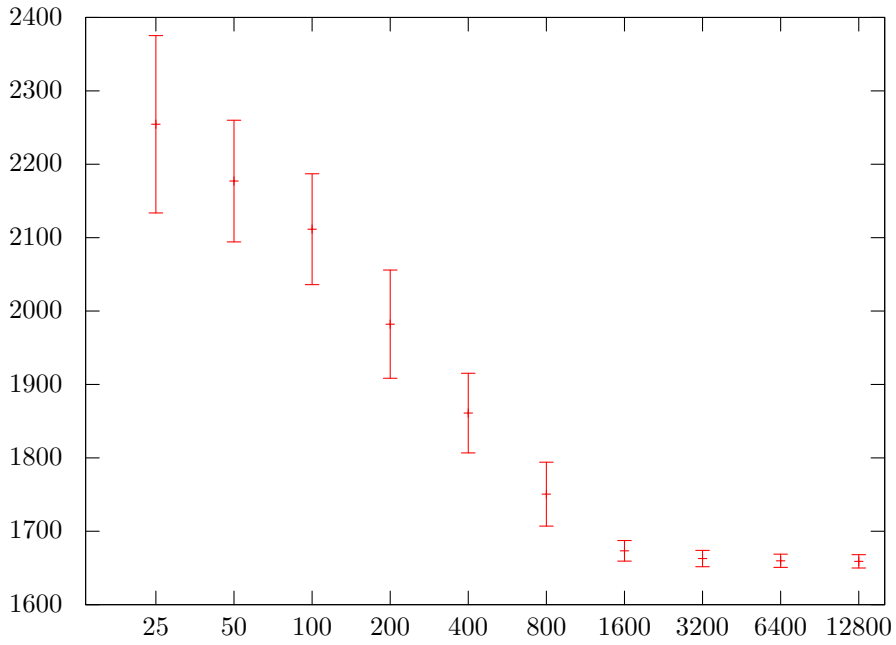
In the end, to get to these results our algorithm created an average of about 171.000 ants, which is an equal number of tours. This is the rounded averaged number of ants over all benchmark problems. These ants are distributed over the two colonies but most tours are generated in the ACS-VEI colony because that lacks the local search method, which is a time consuming operation. In order to achieve this many iterations of the algorithm, multiple optimizations of the source code were needed.

One of the most important performance issues is computing the feasibility and length of a tour. Iterating over all nodes, one has to check the constraints on time windows and capacity, and if these are successful the distance of traveling to this node has to be added to the total distance. This can be done much more efficient by storing the time at which servicing ends, the current load of the vehicle, and the current traveling distance in each node of the tour. By doing this it is often unnecessary to recompute these values for all nodes in the tour. After the insertion of a node we now just have to check the part of the tour between the inserted node and the next depot. It also saves a lot of time during local search.

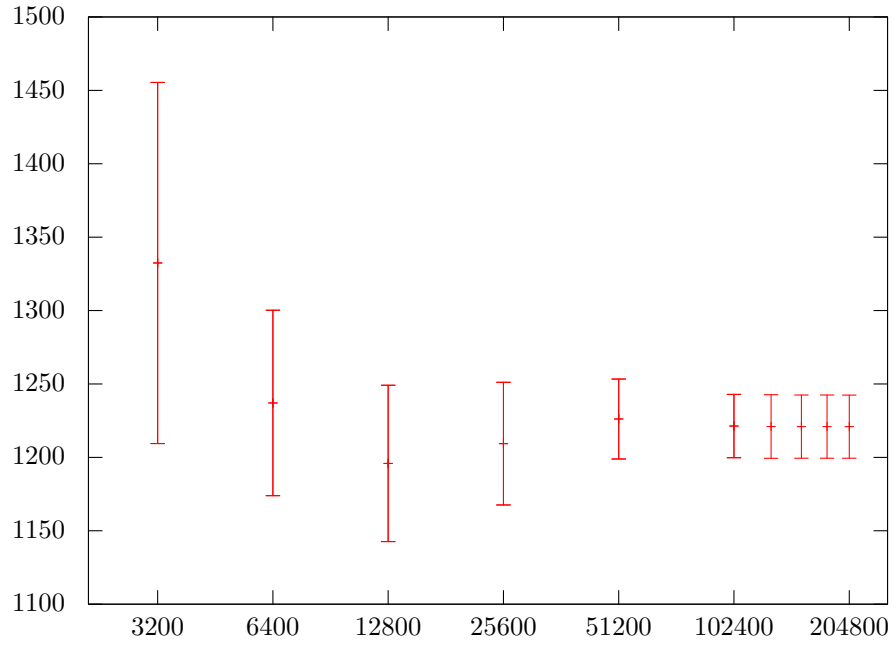
Reproducible results

Many more optimizations can be thought of and have been used. Because we do not know how many ants the original algorithm created it is very hard to compare results. That is why we publish a second set of results that was obtained by changing the stop condition of our algorithm to 100.000 ants. We have also synchronized the colonies so that they create an equal number of ants. Otherwise ACS-VEI would create many more ants, because its creates ants without local search which is much faster.

With these synchronized colonies and a limited number of ants we have created new results that can be found in Table 4.3. These results should be much easier to reproduce because they are independent



(a) R101 problem



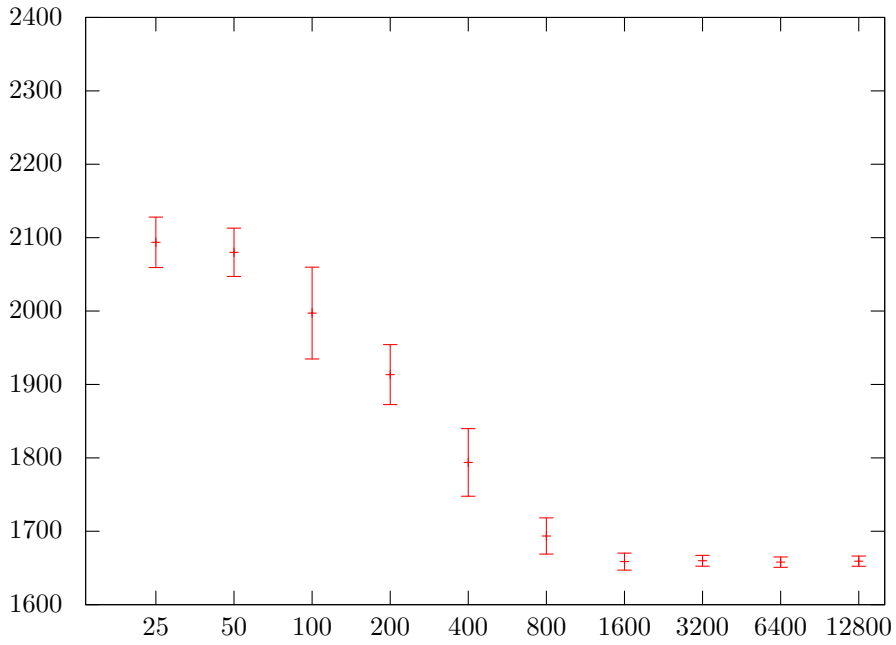
(b) R202 problem

Figure 4.1: Averaged score development of MACS-VRPTW on two different problem instances using a stop condition of 100 CPU seconds.

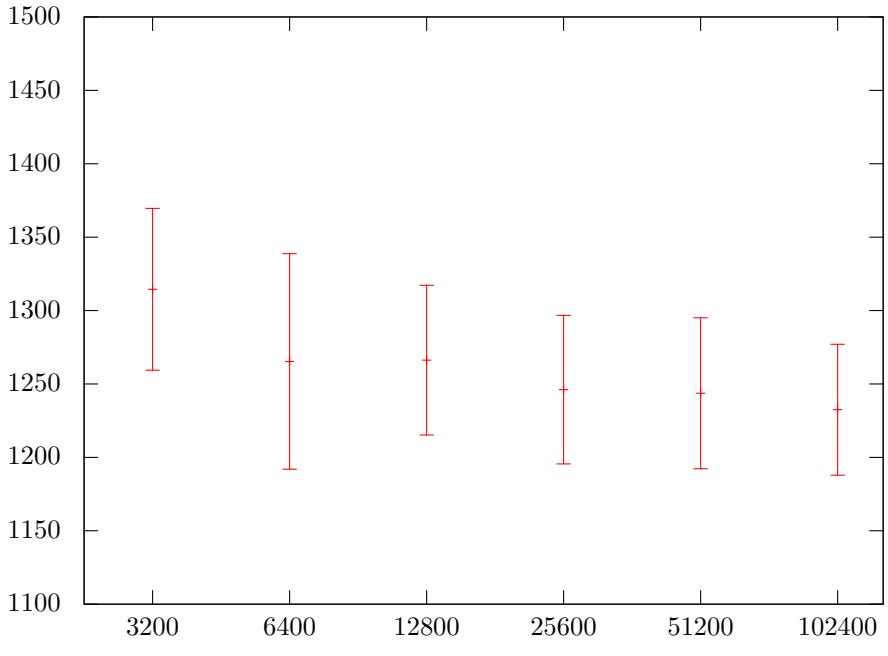
	C1		C2		R1		R2		RC1		RC2	
	dist	**vei**	**dist**	**vei**	**dist**	**vei**	**dist**	**vei**	**dist**	**vei**	**dist**	**vei**
Original	828.40	10.00	593.19	3.00	1214.80	12.55	971.97	3.05	1395.47	12.46	1191.87	3.38
Avg	828.67	10.00	590.14	3.00	1212.87	12.97	976.09	3.09	1377.80	12.67	1151.34	3.47
Best	828.37	10.00	589.85	3.00	1199.49	12.66	940.85	3.09	1369.41	12.12	1119.40	3.37

Table 4.3: Comparison of results between the original MACS-VRPTW and our implementation with a limit on the number of created tours and with synchronized colonies.

of the efficiency of the implementation or the machine on which the tests are run. Figure 4.2 contains the score development plots of the reproducible algorithm, which are very similar to the original ones in Figure 4.1. The differences are caused by the synchronization of the colonies.



(a) R101 problem



(b) R202 problem

Figure 4.2: Averaged score development of MACS-VRPTW on two different problem instances using a stop condition of 100000 iterations.

Chapter 5

MACS-DVRPTW

Now that we have described routing problems, ant colony optimization and the MACS-VRPTW algorithm into detail, it is time to look at the new algorithm that we propose. Our algorithm is designed to solve the Dynamic VRPTW and is based on the MACS-VRPTW algorithm, we have therefor called it MACS-DVRPTW. To our knowledge this is the first ACO algorithm to solve this problem.

We will first describe how a dynamic problem can be simulated and solved in Section 5.1. After that we will describe the MACS-DVRPTW algorithm in Section 5.2. Section 5.3 describes the benchmark problems on which the algorithm was tested and Section 5.4 shows computational results and a comparison to MACS-VRPTW. Section 5.5 concludes with some extensions that were tested to improve the performance of MACS-DVRPTW.

5.1 Solving dynamic problems

To be able to solve a dynamic problem we first have to simulate a form of dynamicity. Kilby, Prosser and Shaw [24] have described a method to do this, which is also used by Montemanni et al. [26]. 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, incorporating all nodes available a priori. This tentative schedule is updated throughout the simulation so that there always is a feasible solution to the current problem. The working day is divided into n_{ts} time slices of length T_{wd}/n_{ts}, also notated with t_{ts}. 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. By dividing the day into time slices we allow the algorithm to work unhindered for a fixed amount of time.

At the beginning of each time slice the algorithms checks for nodes that have become available during the last time slice. These are then inserted into the tentative solution. Next the algorithm commits to parts of the tentative solution that will occur during the next time slice. Commitments are made at the latest possible moment, i.e., we only commit to a node at time t if the vehicle has to leave before $t + t_{ts}$ seconds.

Advance commit Some articles use an advance commit time t_{ac}. This means that a node has to be committed to t_{ac} seconds earlier than would normally be the case, to allow for communication between the planner and the driver. Except for the length of a time step t_{ts}, mentioned above, we have chosen not to use an advance commit time.

Scaling All time-related variables have to be scaled if we choose T_{wd} to be different than the end of the depots time window l_0. This means that all time windows, traveling times and service times have to be multiplied with the factor

$$\frac{T_{wd}}{l_0 - e_0}$$

with l_0 and e_0 being the end and the beginning of the depots time window respectively.

Available times For a problem with time windows, the available times would normally be randomly generated on the interval $[0, e_i - t_{ts}]$. This allows every node to be available before its time window starts. We have extended this, following the example of Gendreau et al. [17], to generating an available time 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. By generating available times on this interval we make sure that the optimal solution can still be attained. This makes it easier to compare this implementation with MACS-VRPTW while it does not necessarily make the problem easier.

Pickups only The static VRP problem can represent a real-world problem involving either the pickup or delivery of packages. In the dynamic problem we alter vehicle tours when they are on their way. This means that it can only represent real-world problems involving picking up packages, because otherwise the truck would have to visit the depot each time a new customer was assigned to the truck.

5.2 Algorithm description

The basic structure of the MACS-VRPTW algorithm stays unchanged, it consists of a controller and two colonies. All parts have their own extensions to deal with the dynamicity of the problem, which we will describe here. The objectives for MACS-DVRPTW are the same as for MACS-VRPTW. The first is to minimize the number of vehicles, the second is to minimize the traveling distance.

5.2.1 The controller

The controller reads the benchmark data and initializes data structures. Afterwards it builds an initial solution using the nodes that are available at $t = 0$ and is ready to start the first time slice. At this point 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 and v_j is the next node, the v_j is committed if $e_j - t_{ij} < t + t_{ts}$.

When the necessary commitments have been made the colonies are (re)started. Each colony tries to find solutions that improve the global best solution found so far, T^*. If a solution with less vehicles is found the colonies are restarted without any changes. 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 7.

The colonies are only restarted when new nodes have become available or when part of T^* will be defined. Otherwise the controller will leave colonies running and wait for the end of the next time-step.

Algorithm 7 Controller

Set time $t = 0$
Set available nodes n
$T^* \leftarrow$ NearestNeighbor(n)
$\tau_0 \leftarrow 1/(n \cdot$ length of $T^*)$
Start measuring CPU time t
Start ACS-TIME(vehicles in T^*) in new thread
Start ACS-VEI(vehicles in $T^* - 1$) in new thread

repeat
    while colonies are active and time step is not over **do**
        Wait until a solution T is found
        if vehicles in $T <$ vehicles in T^* **then**
            Stop threads
        end if
        $T^* \leftarrow T$
    end while

    if time-step is over **then**
        if new nodes are available or new part of T^* will be defined **then**
            Stop threads
            Update available nodes n
            Insert new nodes into T^*
            Commit to nodes in T^*
        end if
    end if

    if colonies have been stopped **then**
        Start ACS-TIME(vehicles in T^*) in new thread
        Start ACS-VEI(vehicles in $T^* - 1$) in new thread
    end if
until $t \geq T_{wd}$

return T^*

5.2.2 The colonies

Other than the fact that the colonies only operate on available nodes nothing changes. Both optimize their own objective although the ACS-VEI colony is limited in its operation because when a vehicle is committed it cannot be optimized out of the solution. Therefore ACS-VEI is only started if there is at least one vehicle in T^* with undefined nodes. Pseudo-codes for ACS-VEI and ACS-TIME can be found in Algorithm 3 and 4 from Section 4.1 respectively.

5.2.3 Problem representation

The representation of the problem does not change from the description in Chapter 4. There is however an extra attribute to every node in T^* that determines whether a node is committed or not. Once a node is committed this can not be undone and that part of the tour cannot be changed anymore.

5.2.4 Constructing a tour

The ConstructTour is significantly changed because of the dynamic problem. One major change is that $\mathcal{N}_j^k$ depends on the availability of nodes and whether or not they are committed. Also, because parts of T^* are committed, these have to be incorporated in every tour that is created by the algorithm. An adjusted pseudo-code for the ConstructTour method can be found in Algorithm 8.

5.2.5 Inserting missing nodes

The method with which missing nodes are inserted is now also called by the controller to insert new nodes. The basic idea is still the same but when the controller calls this function the node should always be added to the problem, even when this leads to the addition of an extra vehicle. This is the way to guarantee that there is always a feasible tentative solution. When this method is called by the colonies it is not possible to create an extra vehicle, only real insertions are allowed. Furthermore, nodes can not be inserted between nodes that are already committed.

5.2.6 Cross Exchange local search

The local search method does not really change. The only difference is that parts of the tour that are committed can not be changed any more. This means that sub-tours that are swapped may not contain committed nodes and that sub-tours cannot be inserted between committed nodes.

5.3 Benchmark problems

The benchmarks that were created for this algorithm are based on the same Solomon benchmark problems on which MACS-VRPTW was tested. Each node was given an additional available time that was described in Section 5.1.

Benchmark problems were created for different degrees of dynamicity ranging from 0 to 50 percent. The 0 percent benchmarks are used to compare with the static problems. For a problem with 10 percent dynamicity it means that each node has a 10 percent change to get a non-zero available time. Because available times are generated on the interval $[0, \overline{e_i}]$, this does not necessarily mean that the generated time will be non-zero because a node's time window might start at zero.

Thus, a benchmark problem with 10 percent dynamicity means that at most 10 percent of all nodes have a non-zero available time. Table 5.3 lists the average amount of nodes that are known a priori for the benchmarks with different degrees of dynamicity.

The benchmark problems will be made available at `natcomp.liacs.nl` along with this report and additional support material.

Algorithm 8 ConstructTour(k, IN)

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

Current vehicle $x \leftarrow 0$
Select a random depot duplicate i
$T^k \leftarrow \langle i \rangle$
current time$_k \leftarrow 0$
load$_k \leftarrow 0$
for each committed node v_i of the x^{th} vehicle of T^* **do**
    $T^k \leftarrow \langle i \rangle$
    current time$_k \leftarrow$ delivery time$_i$ + service time$_i$
    load$_k \leftarrow$ load$_k + q_i$
end for

repeat
    for each $j \in \mathcal{N}_i^k$ **do**
        delivery time$_j \leftarrow \max($current time$_k + t_{ij}, e_j)$
        delta time$_{ij} \leftarrow$ delivery time$_j -$ current time$_k$
        distance$_{ij} \leftarrow$ delta time$_{ij} \times (l_j -$ current time$_k)$
        distance$_{ij} \leftarrow \max(1.0, ($distance$_{ij} -$ IN$_j))$
        $\eta_{ij} \leftarrow 1.0/$ distance$_{ij}$
    end for

    Pick node j using Equation 3.9
    $T^k \leftarrow T^k + \langle j \rangle$
    current time$_k \leftarrow$ delivery time$_j +$ service time$_j$
    load$_k \leftarrow$ load$_k + q_j$
    if j is a depot copy **then**
        current time$_k \leftarrow 0$
        load$_k \leftarrow 0$
        $x \leftarrow x + 1$
        for each committed node v_i of the x^{th} vehicle of T^* **do**
            $T^k \leftarrow \langle i \rangle$
            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

Dynamicity	Nodes known a priori
0%	100
10%	93
20%	86
30%	79
40%	72
50%	65

Table 5.1: Average number of nodes known a priori for created benchmark problems with different degrees of dynamicity.

5.4 Results

Our new algorithm is an extension of MACS-VRPTW, made suitable for the dynamic VRPTW problem. If we run MACS-DVRPTW on the benchmark problems with 0 percent dynamicity we can see what an effect the dynamic nature of the problem problem has on the solutions that are found.

Because the algorithm has to keep track of available nodes and account for more dynamicity in the problem it can perform less computations than MACS-VRPTW. Also, because parts of the current best solution are progressively committed, less of the solution can be changed. This makes it harder to find the same solutions that were found without these limitations.

The tests were performed using the default parameters for the MACS-VRPTW algorithm and additional parameters for simulating the dynamic problem, all of which can be found in Table 5.2. Each benchmark problem was solved 10 times and the amount of vehicles and the total distance of all solutions was averaged.

Parameter	Value
m	10
α	1
β	1
q_0	0.9
ρ	0.1
T_{wd}	100
n_{ts}	50

Table 5.2: Default parameter values for the MACS-DVRPTW algorithm.

Table 5.3 shows the performance of MACS-DVRPTW on problems with different degrees of dynamicity. For the problem with 0 percent dynamicity MACS-DVRPTW achieves scores that are quite similar to those of MACS-VRPTW. The amount of vehicles is practically the same and the difference in distance is somewhat larger. This can be explained by the fact that part of the tour is fixed during the execution of the algorithm and can therefor not be changed anymore. Also, because colonies are restarted more often (possible at the end of each time-step and every time a solution with less vehicles is found) it cannot spend a long consecutive time on solving the problem. When the colony is started anew its pheromones are reset and the IN array of ACS-VEI is reinitialized. This might also influence the results in a negative way.

When the degree of dynamicity is increased the scores get progressively worse. This is just what one would expect. Because less vehicles are known a priori and thus have to be fitted into the tentative schedule it gets more difficult to find high quality solutions.

Despite the enormous amount of literature on routing problems there are only few articles that consider the DVRPTW. Also, many algorithms make other assumptions or use other benchmark problems, which makes it especially difficult to make a good comparison with a state-of-the-art DVRPTW solver.

Algorithm	Dynamicity	Vehicles	Distance
MACS-VRPTW	static	7.35	1030.82
MACS-DVRPTW	0%	7.39	1046.06
MACS-DVRPTW	10%	7.91	1095.10
MACS-DVRPTW	20%	8.37	1131.47
MACS-DVRPTW	30%	8.79	1180.36
MACS-DVRPTW	40%	9.03	1216.72
MACS-DVRPTW	50%	9.41	1230.21

Table 5.3: Results of MACS-DVRPTW compared to MACS-VRPTW on different degrees of dynamicity.

One of the article that is somewhat related to this work is by Gendreau et al. [17]. It deals with the DVRPTW problem with soft time window constraints, uses their own variant of the Solomon benchmark problems and their algorithm is run for 15 or 60 minutes. Their parallel tabu search algorithm seems to find better solutions but as we have explained it is hard to compare these results with ours.

5.5 Extensions

We can now conclude that solving the DVRPTW is a hard task and it gets even harder as the degree of dynamicity increases. To counter this logical but unwanted decrease of solution quality some extensions to MACS-DVRPTW were tested.

5.5.1 Influence of the a priori solution

One of the main difficulties of solving the dynamic VRPTW is that part of the tentative schedule is committed to and cannot be changed afterwards. The solution of the a priori problem, i.e., the problem with only the nodes that are available at $t = 0$, might be of great importance. If no good initial solution is found it might be that some of these nodes are committed at the start of the algorithm, which could greatly affect the performance of the algorithm.

The dynamic problem is solved by simulating a working day. In the same way it would be understandable to simulate some extra time before the working day to solve the a priori problem. A logistics company could do this at night with all the customer requests that have arrived the previous day.

To test this setup, the a priori problem was solved by the algorithm for 20 seconds, which is long enough to find significant improvements on all problems. After this initial period the algorithm is run in the normal way. Only on the problem with 0 percent dynamicity a significant improvement of both objectives is found, with supports this explanation.

	Normal		Improved initial solution	
Dynamicity	Vehicles	Distance	Vehicles	Distance
0%	7.39	1046.06	7.35	1035.86
10%	7.91	1095.10	7.93	1087.06
20%	8.37	1131.47	8.38	1131.41
30%	8.79	1180.36	8.78	1177.96
40%	9.03	1216.72	9.02	1212.11
50%	9.32	1241.32	9.36	1236.36

Table 5.4: Results of MACS-DVRPTW on different degrees of dynamicity without and with an improved solution to the a priori problem.

The results, that can be found in Table 5.4 are almost identical to the original results and so we can not say that optimizing the initial problem yields a real improvement. 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.

5.5.2 Pheromone preservation

The changing nature of the dynamic problem makes it hard to find and keep track of an optimal solution. In MACS-DVRPTW and MACS-VRPTW the pheromone trails for the colonies are reset each time a colony is restarted. By resetting the pheromone trails, the collective memory of the colony is also reset. It seems like a good idea to maintain (part of) the pheromone trails when the colony is restarted. This idea is described among others by Eyckelhof and Snoek [14] and Montemanni et al. [26]. It is called *shaking*, *smoothing* or *pheromone preservation*. We have chosen to follow the implementation of Montemanni et al. and adjust all pheromone levels with

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

where γ is a parameter that is set to 0.3.

Pheromone preservation was tested as a separate extension so the initial problem was not optimized. The results can be seen in Table 5.5. We can see that the average number of vehicles that is needed is a little higher while the average distance of the solutions is lower, with the exception of the 0 percent problems.

These results indicate that pheromone preservation works better on the ACS-TIME colony than on the ACS-VEI colony. The explanation for this could be that solutions with a shorter length have a higher similarity to previous solutions than solutions with less vehicles. Preliminary tests were run that reduced the pheromone preservation only on the ACS-VEI colony, but this does not seem to yield any improvements.

	Normal		**With pheromone preservation**	
Dynamicity	**Vehicles**	**Distance**	**Vehicles**	**Distance**
0%	7.39	1046.06	7.35	1043.13
10%	7.91	1095.10	7.93	1087.98
20%	8.37	1131.47	8.39	1127.67
30%	8.79	1180.36	8.79	1175.14
40%	9.03	1216.72	9.04	1210.38
50%	9.32	1241.32	9.34	1235.90

Table 5.5: Results of MACS-DVRPTW on different degrees of dynamicity without and with pheromone preservation.

5.5.3 Max-Min Ant System

The last extension that was put to the test was incorporating some ideas from the Max-Min Ant System described in Section 3.2.4. The most important idea from MMAS is to enforce lower- and upper boundaries on the pheromone trails, which should prevent early stagnation. This is exactly what intuitively important for solving DVRPTW. Because the problem keeps changing it is important to keep some diversity in the solutions that are generated in order to overcome changes in the fitness landscape.

The pheromone management of MMAS is used. This means that no local pheromone update is applied. The global pheromone update function is applied to all edges and is described in more detail in Section 3.2.4. The MMAS variant described by Stützle and Hoos in [36] was implemented. A pheromone preservation method is used, when a colony is restarted all pheromone trails are reset using

$$\tau_{ij} = \tau_{ij} + \gamma(\tau_{max} - \tau_{ij})$$

where γ is again a parameter controlling how much the pheromones are reset. Like in MACS-DVPTW, γ was set to 0.3 because Stützle and Hoos described that they did not optimize this parameter and this makes for a more equal comparison. MMAS uses an evaporation rate ρ of 0.8.

All other elements of the algorithm, like the transition rule, are kept the same. This leads to a mixture of ACS with MMAS. The results of our tests with this extension can be found in Table 5.6.

	Normal		**MMAS variant**	
Dynamicity	**Vehicles**	**Distance**	**Vehicles**	**Distance**
0%	7.39	1046.06	7.40	1050.06
10%	7.91	1095.10	7.95	1093.66
20%	8.37	1131.47	8.44	1136.97
30%	8.79	1180.36	8.88	1183.02
40%	9.03	1216.72	9.08	1212.48
50%	9.32	1241.32	9.34	1235.90

Table 5.6: Results of MACS-DVRPTW on different degrees of dynamicity without and with MMAS extension.

It is clear that the results attained with the MMAS extension do not improve on the standard MACS-DVRPTW approach. On the benchmark problems with a higher degree of dynamicity MMAS finds solution with a shorter length, but the average number of vehicles is too high on all problems. Also, it looks like the average solution length is not consistent as it is sometimes lower and sometimes higher than MACS-DVRPTW. Though the MMAS algorithm intuitively suits dynamic problems we have not been able to get good results with it. There are however two notes on the performance of this extensions.

The first is that better parameters might increase the performance of the algorithm. Tuning the minimum and maximum pheromone bounds is difficult but plays an essential role in MMAS. Second, due to the extensive checking of pheromone trail bounds the algorithm produces about 30 percent less tours in 100 seconds of CPU time. If the number of created tours would be equal it could be that MMAS performs as well as ACS.

Chapter 6

Two-stage planning

6.1 Introduction

Two-stage stochastic programming is a technique that can be used to solve optimization problems with uncertainty and is formulated in Equation 6.1. To find a solution we (iteratively) choose the best possible value for x. To decide on x it is needed to compute the cost of choosing x at this moment, which is the first stage and is denoted by $f(x)$. Furthermore we have to calculate the expected future costs of choosing x, which is the second stage and is denoted by $E[Q(x, \xi)]$.

$$x = \arg\max_{x \in X}(f(x) + E[Q(x, \xi)]) \tag{6.1}$$

In stochastic programming we make the assumption that the data ξ of the second stage has a probability distribution and is therefor not totally uncertain. For many situations this is a justified assumption, as we will show for the VRP in the next sections.

We can now reformulate $E[Q(x, \xi)]$ as in Equation 6.2, as a summation over all k scenarios in ξ that have a probability of p_k. $Q(x, \xi_k)$ is the estimated quality of the solution that is formed by x and ξ_k. If K would be infinitely large the expected future costs would converge to the real future costs since we sample the full distribution of options.

$$E[Q(x, \xi)] = \sum_{k=1}^{K} p_k \cdot Q(x, \xi_k) \tag{6.2}$$

6.2 Application to the Stochastic VRP

This thesis has focused on the static and dynamic variants of routing problems. The stochastic problems have been ignored so far because in ACO algorithms there is no intuitive way to solve this. Stochastic programming is a good method to solve stochastic problems and the next sections will describe how we can apply it to the Stochastic Vehicle Routing Problem (SVRP).

Before going into details lets list all different variables that are relevant to the algorithm. At the moment that we decide on variable x we might have already made earlier decisions, called $X_{committed}$, which is an ordered set containing all our previous choices of x. There are nodes that are available to select, called $X_{available}$. One of these available nodes will become our new choice of x. Besides these known variables there is also the uncertainty of nodes that might become available in the future, called X_{future}. These nodes are uncertain and are only used to compute the expected future score $Q(x, \xi)$.

We will now discuss how to compute a score for both the first and the second stage. Then a simulator is described that could be used to generate data (X_{future}) on which the expected future cost is based.

6.2.1 Computing the score

Two different scores have to be calculated in the process of deciding on x. The first one is trivial: what is the score of the tour if node x is added? This can be computed by summing up all traveling times in the tour described by $X_{committed} + x$.

The expected score for the second stage can be computed with Equation 6.2. A set of scenarios ξ_k with an associated probability p_k is needed. Each scenario consists of a set of nodes X_{future}. Suppose we have a simulator that can generate this set X_{future} and assign an appropriate probability to it. We now need a mechanism that can create a tour that starts at node x and incorporates all nodes in X_{future}. This can be done by any algorithm that can solve the static VRP, like MACS-VRPTW. This algorithm should be run for a limited time and return the best solution it has found. The score of this solution is $Q(x, \xi_k)$ for this k^{th} scenario.

So, to calculate the expected score of choosing x we need to generate a sufficiently large set of k scenarios. The bigger k, the better our expected score will estimate the real score, but the greater the computational costs. It is therefor not only important to choose k wisely but also to choose a VRP-solving algorithm that does not require a lot of time.

6.2.2 A simple simulator

Building a simulator for the SVRP can be a very complicated task. Here we will propose a very basic approach to give an idea of what the simulator does and how this can be accomplished.

The simulator will have to return a set of nodes that is representative for nodes that will be visible to the two-stage algorithm in the future. The simulator can generate these scenarios in many ways but our first choice would be to use existing data. Let us assume we have a year worth of data from a logistics company, that is, 365 records that form our distribution of "what happens on a typical day".

When the simulator is called it should return a representative set of nodes. The more representative this set is, the better the simulators output will resemble the real world. One of the easiest solutions would be to return a random record from the data-set. Maybe the returned data should be sanitized first, nodes that are already part of $X_{committed} \cup X_{available}$ should be excluded. One might also limit the size of the returned set to the average size over all records - $|X_{committed} \cup X_{available}|$, with a lower limit to make sure we always output something.

Besides a set of nodes, the simulator should also output the probability of the returned scenario. In this simple example that is always 1 divided by the number of records in the data-set.

We could improve this simulator in many ways. The fact that today happens to be a Wednesday can have a large impact on which customers will require service. In that case, selecting a scenario that was created on a Wednesday might greatly improve the representativeness of the returned nodes. We could think of many more refinements that could make the simulator better.

Chapter 7

Conclusions

This report has given an overview of different routing problems and ways to solve them with Ant Colony Optimization. We have seen a wide variety of ACO algorithms and extensions. The MACS-VRPTW is discussed as it is a state-of-the-art ACO algorithm for solving the VRPTW problem. Then the MACS-DVRPTW is proposed that can solve the DVRPTW problem with different degrees of dynamicity. To our knowledge this is the first ACO algorithm that solves the DVRPTW problem.

The results on the DVRPTW look promising but it is very hard to compare these with results from other articles. Some extensions to the MACS-DVRPTW algorithm were tested and these increase the performance slightly. The introduction of Max-Min Ant System principles did not yield better results but more tests are needed to see if different parameter settings can make this approach work.

A new set of benchmark problems was generated based on the classic benchmark set of Solomon. These benchmark problems can be used to simulate the DVRPTW and will be made public so that others can make a comparison with our algorithm.

Chapter 6 introduced the basics of two-stage planning and how this can be applied to solving the Stochastic VRP. This approach looks very promising and should be implemented to see how it performs compared to other algorithms.

Chapter 8

Further work

The MACS-DVRPTW can be improved in a number of ways to make it more efficient at solving the DVRPTW problem. Its parameters should be properly tuned, preferably with an meta-optimizer like a mixed-integer genetic algorithm or a particle swarm optimizer. The same meta-optimization can be applied to the parameters of the MMAS extensions.

More extensions can be tested. An example is the use of candidate lists based on time windows. Many nodes cannot be reached because the time window constraints do not allow them to be service after each other. Precomputing and storing these limitations could speed up the algorithm. Also it would be a good idea to test a changing evaporatuin rate to increase exploration of the algorithm at critical moments, for example when new nodes are added to the problem.

Lastly, a two-stage planning method based on ACO should be implemented to tests its performance. The MACS-VRPTW algorithms has proved itself as a fast optimization algorithm and this would be a good candidate for implementation with the two-stage planning principle.

Nomenclature

Routing problems

TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows
CVRPTW	Capacitated Vehicle Routing Problem with Time Windows, same as VRPTW
DVRPTW	Dynamic Vehicle Routing Problem with Time Windows
DCVRPTW	Dynamic Capacitated Vehicle Routing Problem with Time Windows, same as DVRPTW

Ant Colony Optimization algorithms

AS	Ant System
EAS	Elitist Ant System
ACS	Ant Colony System
MMAS	Max-Min Ant System
RBAS	Rank Based Ant System
SbAS	Savings based Ant System

Problem variables

n	Number of nodes
d_{ij}	Traveling distance between nodes v_i and v_j
t_{ij}	Traveling time between nodes v_i and v_j
Q	Maximum capacity of a vehicle in the Vehicle Routing Problem
q_i	Demand for goods at node v_i
v_0	Depot node
$[e_i, l_i]$	Time window constraint for node v_i
e_i	Earliest beginning of service for node v_i
l_i	Latest beginning of service for node v_i

Algorithm variables

τ_{ij}	Pheromone level on edge (i, j)
η_{ij}	Heuristic desirability of edge (i, j)
ρ	Evaporation rate
m	Number of ants
$\mathcal{N}_i^k$	Set of nodes that can be visited by ant k positioned at node v_i
T_k	Tour of ant k
L_k	Length or total traveling distance of a tour T_k
T^*	Best solution found so far
L^*	Length of the best solution found so far (T^*)
T^{VEI}	Best solution of the ACS-VEI colony
τ_0	Initial pheromone level
q_0	Parameter determining the balance between exploitation and exploration

Dynamic problem variables

T_{wd}	Duration (in seconds) of the simulated working day
n_{ts}	Number of time-steps into which the working day is divided
t_{ts}	Length of each time-step in seconds, equal to T_{wd}/n_{ts}
t	Current time during the simulation

Bibliography

[1] B. Barán and M. Schaerer. A multiobjective ant colony system for vehicle routing problem with time windows. In M. Hamza, editor, Applied Informatics, 2003., 21st IASTED International Conference on, pages 97–102. IASTED, feb 2003.

[2] J.E. Bell and P.R. McMullen. Ant colony optimization techniques for the vehicle routing problem. Advanced Engineering Informatics, 18(1):41–48, 2004.

[3] B. Bullnheimer, R.F. Hartl, and C. Strauss. Applying the ant system to the vehicle routing problem. In Metaheuristics, 1997., 2nd International Conference on, Sophia-Antipolis, France, 1997.

[4] B. Bullnheimer, R.F. Hartl, and C. Strauß. A new rank based version of the ant system - a computational study. Central European Journal for Operations Research and Economics, 7:25–38, 1997.

[5] B. Bullnheimer, R.F. Hartl, and C. Strauss. An improved ant system algorithm for thevehicle routing problem. Annals of Operations Research, 89:319–328, 1999. 10.1023/A:1018940026670.

[6] G. Clarke and J.W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research, 12(4):568–581, 1964.

[7] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In F. Varela and P Bourgine, editors, Proceedings of the First European Conference on Artificial Life, pages 134–142, Paris, France, 1991. Elsevier.

[8] G.A. Croes. A method for solving traveling-salesman problems. Operations Research, 6(6):791–812, 1958.

[9] K. Doerner, M. Gronalt, R. Hartl, M. Reimann, C. Strauß, and M. Stummer. Savingsants for the vehicle routing problem. In Stefano Cagnoni, Jens Gottlieb, Emma Hart, Martin Middendorf, and Günther Raidl, editors, Applications of Evolutionary Computing, volume 2279 of Lecture Notes in Computer Science, pages 73–109. Springer Berlin / Heidelberg, 2002.

[10] M. Dorigo. Ottimizzazione, Apprendimento Automatico, ed Algoritmi Basati su Metafora Naturale (Optimization, Learning and Natural Algorithms). PhD thesis, Politecnico di Milano, Italy, 1992.

[11] M. Dorigo and C. Blum. Ant colony optimization theory: A survey. Theoretical Computer Science, 344(2–3):243–278, 2005.

[12] M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. Evolutionary Computation, IEEE Transactions on, 1(1):53 –66, apr 1997.

[13] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 26(1):29–41, feb 1996.

[14] C. Eyckelhof and M. Snoek. Ant systems for a dynamic tsp. In Marco Dorigo, Gianni Di Caro, and Michael Sampels, editors, Ant Algorithms, volume 2463 of Lecture Notes in Computer Science, pages 88–99. Springer Berlin / Heidelberg, 2002.

[15] M.M. Flood. The traveling-salesman problem. Operations Research, 4(1):61, 1956.

[16] L.M. Gambardella, É. Taillard, and G. Agazzi. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. In New Ideas in Optimization, chapter 5, pages 63–76. McGraw-Hill, 1999.

[17] M. Gendreau, F. Guertin, J.-Y. Potvin, and É Taillard. Parallel tabu search for real-time vehicle routing and dispatching. Transportation Science, 33(4):381, 1999.

[18] B.E. Gillett and L.R. Miller. A heuristic algorithm for the vehicle-dispatch problem. Operations Research, 22(2):340, 1974.

[19] S. Goss, S. Aron, J.L. Deneubourg, and J.M. Pasteels. Self-organized shortcuts in the argentine ant. Naturwissenschaften, 76(12):579–581, 1989.

[20] M. Guntsch and M. Middendorf. Applying population based aco to dynamic optimization problems. In M. Dorigo, G. Di Caro, and M. Sampels, editors, Ant Algorithms, volume 2463 of Lecture Notes in Computer Science, pages 97–104. Springer Berlin / Heidelberg, 2002.

[21] M. Guntsch and M. Middendorf. A population based approach for aco. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. Raidl, editors, Applications of Evolutionary Computing, volume 2279 of Lecture Notes in Computer Science, pages 165–180. Springer Berlin / Heidelberg, 2002.

[22] W.J. Gutjahr. Aco algorithms with guaranteed convergence to the optimal solution. Information Processing Letters, 82(3):145–153, 2002.

[23] R.M. Karp. Reducibility among combinatorial problems. Complexity of Computer Computations, 40(4):85–103, 1972.

[24] P. Kilby, P. Prosser, and P. Shaw. Dynamic vrps: a study of scenarios. Technical Report APES-06-1998, University of Strathclyde, sep 1998.

[25] K Menger. Ein theorem über die bogenlänge. Akademie der Wissenschaften in Wien – Mathematisch-naturwissenschaftligge Klasse, (65):264–266, 1928.

[26] R. Montemanni, L.M. Gambardella, A.E. Rizzoli, and A.V. Donati. Ant colony system for a dynamic vehicle routing problem. Journal of Combinatorial Optimization, 10:327–343, 2005. 10.1007/s10878-005-4922-6.

[27] H. Paessens. The savings algorithm for the vehicle routing problem. European Journal of Operational Research, 34(3):336–344, 1988.

[28] V. Pillac, M. Gendreau, C. Guéret, and A.L. Medaglia. A review of dynamic vehicle routing problems. European Journal of Operational Research, 225(1):1 – 11, 2013.

[29] H.N. Psaraftis. Dynamic vehicle routing: Status and prospects. Annals of Operations Research, 61:143–164, 1995.

[30] M. Reimann, K. Doerner, and R.F. Hartl. D-ants: Savings based ants divide and conquer the vehicle routing problem. Computers & Operations Research, 31(4):563–591, 2004.

[31] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.-F. Puget, editors, Principles and Practice of Constraint Programming — CP98, volume 1520 of Lecture Notes in Computer Science, pages 417–431. Springer Berlin Heidelberg, 1998.

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

[33] T. Stützle and M. Dorigo. A short convergence proof for a class of ant colony optimization algorithms. Evolutionary Computation, IEEE Transactions on, 6(4):358–365, aug 2002.

[34] T. Stützle and H. Hoos. Improving the ant system: A detailed report on the max-min ant system. Technical Report AIDA-96-12, FG Intellektik, TH Darmitadt, aug 1996.

[35] T. Stützle and H. Hoos. Max-min ant system and local search for the traveling salesman problem. In Evolutionary Computation, 1997., IEEE International Conference on, pages 309 –314, apr 1997.

[36] T. Stützle and H. Hoos. Max–min ant system. Future Generation Computer Systems, 16(8):889 – 914, 2000.

[37] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science, 31(2):170, 1997.

Appendix A

Best solutions of MACS-VRPTW

For generating the DVRPTW benchmarks of Chapter 5 the best results of my implementation of the MACS-VRPTW algorithm were used. For complete documentation these best solutions are included below, in the format that was also used by Gambardella et al. on `http://www.idsia.ch/~luca/macs-vrptw/solutions/welcome.htm`.

```
c101
828.936867 10
0 5 3 7 8 10 11 9 6 4 2 1 75
0 20 24 25 27 29 30 28 26 23 22 21
0 98 96 95 94 92 93 97 100 99
0 81 78 76 71 70 73 77 79 80
0 43 42 41 40 44 46 45 48 51 50 52 49 47
0 67 65 63 62 74 72 61 64 68 66 69
0 13 17 18 19 15 16 14 12
0 57 55 54 53 56 58 60 59
0 32 33 31 35 37 38 39 36 34
0 90 87 86 83 82 84 85 88 89 91

c102
828.936867 10
0 67 65 63 62 74 72 61 64 68 66 69
0 20 24 25 27 29 30 28 26 23 22 21
0 43 42 41 40 44 46 45 48 51 50 52 49 47
0 90 87 86 83 82 84 85 88 89 91
0 5 3 7 8 10 11 9 6 4 2 1 75
0 81 78 76 71 70 73 77 79 80
0 57 55 54 53 56 58 60 59
0 13 17 18 19 15 16 14 12
0 32 33 31 35 37 38 39 36 34
0 98 96 95 94 92 93 97 100 99

c103
828.064882 10
0 90 87 86 83 82 84 85 88 89 91
0 5 3 7 8 10 11 9 6 4 2 1 75
0 67 65 62 74 72 61 64 68 66 69
0 43 42 41 40 44 46 45 48 51 50 52 49 47
0 98 96 95 94 92 93 97 100 99
0 57 55 54 53 56 58 60 59
0 13 17 18 19 15 16 14 12
0 81 78 76 71 70 73 77 79 80 63
0 32 33 31 35 37 38 39 36 34
0 20 24 25 27 29 30 28 26 23 22 21

c104
824.776708 10
0 13 17 18 19 15 16 14 12 10
0 5 3 7 8 11 9 6 4 2 1 75
0 90 87 86 83 82 84 85 88 89 91
0 67 65 62 74 72 61 64 68 66 69
0 20 24 25 27 29 30 28 26 23 22 21
```

```
0 43 42 41 40 44 45 46 48 51 50 52 49 47
0 55 54 53 56 58 60 59 57
0 81 78 76 71 70 73 77 79 80 63
0 98 96 95 94 92 93 97 100 99
0 32 33 31 35 37 38 39 36 34

c105
828.936867 10
0 90 87 86 83 82 84 85 88 89 91
0 5 3 7 8 10 11 9 6 4 2 1 75
0 67 65 63 62 74 72 61 64 68 66 69
0 43 42 41 40 44 46 45 48 51 50 52 49 47
0 20 24 25 27 29 30 28 26 23 22 21
0 81 78 76 71 70 73 77 79 80
0 98 96 95 94 92 93 97 100 99
0 13 17 18 19 15 16 14 12
0 57 55 54 53 56 58 60 59
0 32 33 31 35 37 38 39 36 34

c106
828.936867 10
0 43 42 41 40 44 46 45 48 51 50 52 49 47
0 90 87 86 83 82 84 85 88 89 91
0 20 24 25 27 29 30 28 26 23 22 21
0 5 3 7 8 10 11 9 6 4 2 1 75
0 81 78 76 71 70 73 77 79 80
0 67 65 63 62 74 72 61 64 68 66 69
0 98 96 95 94 92 93 97 100 99
0 32 33 31 35 37 38 39 36 34
0 57 55 54 53 56 58 60 59
0 13 17 18 19 15 16 14 12

c107
828.936867 10
0 43 42 41 40 44 46 45 48 51 50 52 49 47
0 90 87 86 83 82 84 85 88 89 91
0 81 78 76 71 70 73 77 79 80
0 20 24 25 27 29 30 28 26 23 22 21
0 5 3 7 8 10 11 9 6 4 2 1 75
0 67 65 63 62 74 72 61 64 68 66 69
0 98 96 95 94 92 93 97 100 99
0 32 33 31 35 37 38 39 36 34
0 13 17 18 19 15 16 14 12
0 57 55 54 53 56 58 60 59

c108
828.936867 10
0 32 33 31 35 37 38 39 36 34
0 81 78 76 71 70 73 77 79 80
0 5 3 7 8 10 11 9 6 4 2 1 75
0 67 65 63 62 74 72 61 64 68 66 69
0 20 24 25 27 29 30 28 26 23 22 21
0 57 55 54 53 56 58 60 59
0 13 17 18 19 15 16 14 12
0 90 87 86 83 82 84 85 88 89 91
0 98 96 95 94 92 93 97 100 99
0 43 42 41 40 44 46 45 48 51 50 52 49 47

c109
828.936867 10
0 57 55 54 53 56 58 60 59
0 98 96 95 94 92 93 97 100 99
0 20 24 25 27 29 30 28 26 23 22 21
0 67 65 63 62 74 72 61 64 68 66 69
0 5 3 7 8 10 11 9 6 4 2 1 75
0 81 78 76 71 70 73 77 79 80
0 13 17 18 19 15 16 14 12
0 43 42 41 40 44 46 45 48 51 50 52 49 47
0 32 33 31 35 37 38 39 36 34
0 90 87 86 83 82 84 85 88 89 91
```

c201
591.556557 3
0 20 22 24 27 30 29 6 32 33 31 35 37 38 39 36 34 28 26 23 18 19 16 14 12 15 17
13 25 9 11 10 8 21
0 93 5 75 2 1 99 100 97 92 94 95 98 7 3 4 89 91 88 84 86 83 82 85 76 71 70 73 80
79 81 78 77 96 87 90
0 67 63 62 74 72 61 64 66 69 68 65 49 55 54 53 56 58 60 59 57 40 44 46 45 51 50
52 47 43 42 41 48

c202
591.556557 3
0 20 22 24 27 30 29 6 32 33 31 35 37 38 39 36 34 28 26 23 18 19 16 14 12 15 17
13 25 9 11 10 8 21
0 93 5 75 2 1 99 100 97 92 94 95 98 7 3 4 89 91 88 84 86 83 82 85 76 71 70 73 80
79 81 78 77 96 87 90
0 67 63 62 74 72 61 64 66 69 68 65 49 55 54 53 56 58 60 59 57 40 44 46 45 51 50
52 47 43 42 41 48

c203
591.173443 3
0 20 22 24 27 30 29 6 32 33 31 35 37 38 39 36 34 28 26 23 18 19 16 14 12 15 17
13 25 9 11 10 8 21
0 93 5 75 2 1 99 100 97 92 94 95 98 7 3 4 89 91 88 84 86 83 82 85 76 71 70 73 80
79 81 78 77 96 87 90
0 67 63 62 74 72 61 64 66 69 68 65 49 55 54 53 56 58 60 59 57 40 44 46 45 51 50
52 47 42 41 43 48

c204
590.598746 3
0 67 63 62 74 72 61 64 66 69 68 65 49 55 54 53 56 58 60 59 57 40 44 46 41 42 45
51 50 52 47 43 48
0 20 22 24 27 30 29 6 32 33 31 35 37 38 39 36 34 28 26 23 18 19 16 14 12 15 17
13 25 9 11 10 8 21
0 93 5 75 2 1 99 100 97 92 94 95 98 7 3 4 89 91 88 84 86 83 82 85 76 71 70 73 80
79 81 78 77 96 87 90

c205
588.875963 3
0 20 22 24 27 30 29 6 32 33 31 35 37 38 39 36 34 28 26 23 18 19 16 14 12 15 17
13 25 9 11 10 8 21
0 93 5 75 2 1 99 100 97 92 94 95 98 7 3 4 89 91 88 86 84 83 82 85 76 71 70 73 80
79 81 78 77 96 87 90
0 67 63 62 74 72 61 64 66 69 68 65 49 55 54 53 56 58 60 59 57 40 44 46 45 51 50
52 47 43 42 41 48

c206
588.492849 3
0 20 22 24 27 30 29 6 32 33 31 35 37 38 39 36 34 28 26 23 18 19 16 14 12 15 17
13 25 9 11 10 8 21
0 93 5 75 2 1 99 100 97 92 94 95 98 7 3 4 89 91 88 86 84 83 82 85 76 71 70 73 80
79 81 78 77 96 87 90
0 67 63 62 74 72 61 64 66 69 68 65 49 55 54 53 56 58 60 59 57 40 44 46 45 51 50
52 47 42 41 43 48

c207
588.286321 3
0 20 22 24 27 30 29 6 32 33 31 35 37 38 39 36 34 28 26 23 17 18 19 16 14 12 15
13 25 9 11 10 8 21
0 93 5 75 2 1 99 100 97 92 94 95 98 7 3 4 89 91 88 86 84 83 82 85 76 71 70 73 80
79 81 78 77 96 87 90
0 67 63 62 74 72 61 64 66 69 68 65 49 55 54 53 56 58 60 59 57 40 44 46 45 51 50
52 47 42 41 43 48

c208
588.323801 3
0 67 63 62 74 72 61 64 66 69 68 65 49 55 54 53 56 58 60 59 57 40 44 46 45 51 50
52 47 42 41 43 48
0 20 22 24 27 30 29 6 32 33 31 35 37 38 39 36 34 28 26 23 18 17 19 16 14 12 15
13 25 9 11 10 8 21

```
0 93 5 75 2 1 99 100 97 92 94 95 98 7 3 4 89 91 88 86 84 83 82 85 76 71 70 73 80
79 81 78 77 96 87 90

r101
1650.799240 19
0 63 64 49 48
0 45 82 18 84 60 89
0 95 98 16 86 91 100
0 2 21 73 41 56 4
0 92 42 15 87 57 97
0 33 29 78 34 35 77
0 5 83 61 85 37 93
0 28 12 40 53 26
0 62 11 90 20 32 70
0 14 44 38 43 13
0 36 47 19 8 46 17
0 27 69 76 79 3 54 24 80
0 52 6
0 65 71 81 50 68
0 72 75 22 74 58
0 39 23 67 55 25
0 30 51 9 66 1
0 59 99 94 96
0 31 88 7 10

r102
1486.119471 17
0 28 29 78 34 35 3 77
0 62 88 8 46 17 93 59
0 94 96 99 6
0 50 33 30 51 9 66 1
0 65 71 81 20 32 70
0 92 37 98 85 16 86 91 97 13
0 52 7 11 19 49 48 82
0 40 53
0 42 15 41 75 56 74 21
0 63 64 90 10 31
0 87 57 2 58
0 83 45 61 84 5 60 89
0 14 44 38 43 100 95
0 73 22 72 54 24 80 12
0 27 69 76 79 68
0 36 47 18
0 39 23 67 55 4 25 26

r103
1292.675510 13
0 71 65 78 34 35 81 77 28
0 42 43 15 57 41 74 72 73 21 58
0 2 22 75 56 4 25 54
0 27 69 30 9 66 20 51 1
0 40 53 12 68 80
0 60 45 83 5 99 6
0 26 39 23 67 55 24 29 3
0 7 19 11 8 46 47 48 82 18 89
0 92 98 14 44 38 86 16 61 85 91 100 37
0 36 64 49 63 90 32 70
0 94 96 95 97 87 13
0 52 62 88 84 17 93 59
0 50 33 76 79 10 31

r104
986.025445 10
0 6 96 99 84 5 93 59 97 95 94
0 1 50 33 81 51 9 35 34 78 3 77 28
0 27 69 76 79 29 24 68 80 12 26
0 70 30 20 71 65 66 32 90 63 10
0 89 60 83 17 45 8 46 47 48 82 18
0 42 43 15 87 57 41 22 74 73 21 40
0 53 13 2 58
```

```
0 92 98 14 44 38 86 16 61 85 91 100 37
0 31 88 62 11 64 49 36 19 7 52
0 72 75 56 23 67 39 55 4 25 54

r105
1377.111018 14
0 95 92 98 99 94 6 96
0 59 5 61 16 85 84 17 60 89
0 21 73 75 22 41 56 74 58
0 33 65 71 9 66 20 1
0 28 12 29 79 3 50 68 24 80
0 31 30 51 81 78 34 35 77
0 63 64 11 90 10 32 70
0 72 39 23 67 55 54 4 25
0 42 14 44 38 86 37 97 13
0 47 36 19 49 46 48
0 2 15 57 87 43 100 91 93
0 27 69 76 40 53 26
0 45 83 82 8 18
0 52 62 88 7

r106
1259.708773 12
0 27 28 40 87 57 43 100 37 98
0 50 33 65 71 66 20 32 70 1
0 26 39 23 67 55 54 4 25
0 48 47 36 19 49 46 82 7 52
0 2 15 73 41 22 75 56 74 72 21 58
0 42 14 44 38 86 91 97 13
0 94 95 92 59 99 6 53
0 63 64 11 90 10 31
0 5 45 16 61 84 17 85 93 96
0 29 76 79 78 68 24 80 12
0 62 88 18 8 83 60 89
0 69 30 51 81 9 35 34 3 77

r107
1119.185972 10
0 33 81 65 71 9 66 20 51 1
0 12 54 39 23 67 55 4 25 26
0 94 96 92 59 99 6 87 97 95 13
0 42 43 14 44 38 86 16 91 100 37 98
0 52 7 11 62 88 31 10 63 90 32 70
0 27 69 30 79 78 34 35 3 77
0 2 57 15 41 22 75 56 74 72 73 21 58
0 60 83 45 46 8 84 5 17 61 85 93
0 48 47 36 64 49 19 82 18 89
0 28 50 76 40 53 68 29 24 80

r108
974.158678 9
0 2 57 15 43 42 87 97 95 94 13 58
0 72 75 56 23 67 39 55 4 25 54
0 20 66 65 71 9 35 34 78 77 28
0 52 7 48 82 8 46 47 36 49 19
0 27 69 50 76 53 40 21 73 74 22 41
0 31 88 62 11 64 63 90 32 10 70 1
0 6 96 59 93 99 5 84 17 45 83 60 18 89
0 30 51 81 33 79 3 29 24 68 80 12 26
0 92 98 91 44 14 38 86 16 61 85 100 37

r109
1210.570802 11
0 27 69 33 78 81 9 35 34 24 80
0 52 88 19 47 36 49 48
0 2 72 73 21 40 53 26 54 55 25
0 83 45 82 7 18 8 46 17 60 89
0 95 42 15 57 87 94 6 96 97 13
0 39 67 23 75 22 41 74 56 4
0 28 12 76 79 29 68 3 50 77
```

```
0 62 11 64 63 90 32 10
0 5 59 99 85 61 16 84 93 37 100 91
0 92 98 14 44 86 38 43 58
0 31 30 51 71 65 66 20 70 1

r110
1089.253442 11
0 27 31 30 51 33 81 78 34 24 80 68
0 95 98 44 16 86 38 14 43 42 58
0 69 9 35 71 65 66 20 70 1
0 59 99 93 5 84 61 85 91 100 37 92
0 28 12 76 77 29 79 3 50
0 21 72 73 22 75 23 67 39 56 74
0 88 62 11 64 63 90 32 10
0 53 40 26 54 55 25 4
0 6 18 83 8 46 45 17 60 89
0 2 41 15 57 87 94 96 97 13
0 52 82 7 19 47 49 36 48

r111
1055.311998 11
0 28 76 79 78 29 24 68 80 12 26
0 95 42 14 44 38 86 16 17 5 60 89
0 27 69 30 20 65 66 32 70 1
0 52 88 62 11 63 90 10 31
0 50 33 81 51 9 71 35 34 3 77
0 92 98 37 59 99 97 87 6 94 13
0 83 18 45 8 84 61 91 100 85 93 96
0 53 40
0 73 75 23 67 39 55 4 25 54
0 7 19 64 49 36 46 47 48 82
0 2 57 15 43 41 22 74 56 72 21 58

r112
964.407242 10
0 69 30 51 9 35 71 65 66 20 1
0 21 73 72 75 56 23 67 39 25 55 4
0 2 57 87 42 43 15 22 41 74 58
0 28 76 79 33 81 78 34 29 24 54
0 18 82 8 45 17 84 83 60 89
0 52 7 19 64 49 36 46 47 48
0 53 40 26 12 80 68 77 3 50
0 5 61 16 86 38 14 44 91 100 37 96
0 6 94 95 59 99 93 85 98 92 97 13
0 27 31 88 62 11 63 90 32 10 70

r201
1259.785690 4
0 33 65 63 31 69 52 27 28 12 29 76 30 71 9 51 81 79 78 34 3 50 26 68 54 56 74 55
4 25 24 80 77
0 95 59 92 42 15 14 98 61 16 44 38 86 85 99 84 8 49 46 48 60 17 91 100 93 89
0 2 72 39 75 23 67 21 73 40 53 87 57 22 41 43 37 97 96 13 58
0 5 83 45 82 47 36 64 11 19 62 88 7 18 6 94 90 10 20 66 35 32 70 1

r202
1191.702969 3
0 83 45 48 47 36 63 64 11 19 62 88 30 71 78 79 81 9 51 90 49 46 82 7 10 20 32 66
35 68 12 26 13 58
0 50 33 65 34 29 3 28 27 69 76 67 73 40 53 87 57 41 22 75 56 74 54 55 4 25 24 80
77 1 70 31 52
0 96 59 92 37 98 85 91 14 42 2 21 72 39 23 15 38 44 16 61 99 18 8 84 86 5 6 94
95 97 43 100 93 17 60 89

r203
943.489924 3
0 89 60 83 45 46 36 64 11 62 69 88 30 1 76 3 79 78 9 51 20 66 71 35 68 12 26 13
95 59 93 17 61 91 100 98 37 97 58
0 27 94 92 42 57 15 43 14 44 38 86 16 85 99 96 6 5 84 8 48 47 49 19 63 90 32 10
70 31 7 82 18 52
0 50 33 81 65 34 29 24 39 67 23 72 73 21 40 53 87 2 41 22 75 56 74 4 55 25 54 80
```

77 28

r204
829.769039 2
0 27 52 18 82 7 88 10 70 1 69 31 62 11 19 48 45 17 86 44 38 14 99 87 84 8 46 47
36 49 64 63 90 32 30 20 66 65 71 35 34 78 81 33 3 77 68 29 24 25 21 40 58
0 6 96 92 97 42 43 15 57 41 22 75 56 23 67 39 80 76 79 9 51 50 28 53 26 12 54 55
4 72 74 73 2 13 94 95 59 93 85 98 37 100 91 16 61 5 83 60 89

r205
1015.527502 3
0 95 59 92 98 14 42 15 2 72 39 67 23 75 22 41 73 21 40 76 50 3 79 81 9 78 34 35
66 20 32 10 70 48 17 60 89
0 27 28 12 29 33 65 71 51 30 90 11 19 49 46 8 18 6 94 53 26 74 56 4 25 55 54 24
80 68 77 1
0 5 83 45 82 47 36 64 63 62 7 88 31 69 52 84 61 16 44 38 86 85 99 97 87 57 43 91
100 37 93 96 13 58

r206
919.323182 3
0 82 48 47 36 45 5 59 92 42 15 14 44 38 86 16 61 99 96 6 94 95 97 87 43 57 2 74
75 56 4 25 55 54 24 80 68 77
0 27 69 1 50 33 65 71 30 63 90 62 11 64 49 19 88 7 18 8 46 84 17 83 60 93 85 91
100 98 37 13 58
0 28 26 12 29 39 67 23 41 22 72 73 21 40 53 76 3 79 81 51 9 78 34 35 66 20 32 10
70 31 52 89

r207
908.684017 2
0 2 57 42 92 96 83 45 46 36 64 11 62 88 52 27 69 30 51 9 78 79 3 76 28 53 40 21
73 72 41 22 74 75 56 4 25 55 54 80 68 77 12 26 58 13 95 59 93 60 18 89
0 1 50 33 81 71 65 34 29 24 39 67 23 15 43 14 44 38 86 16 85 98 97 87 6 99 5 84
8 48 47 49 19 10 63 90 32 66 35 20 70 31 7 82 17 61 91 100 37 94

r208
731.584755 2
0 89 6 94 95 92 37 98 93 85 16 86 44 38 14 43 15 41 22 75 56 23 67 39 24 29 79
78 34 9 35 71 65 66 20 51 81 33 3 77 68 80 54 55 25 4 72 74 73 21 40 58
0 27 52 18 7 88 31 10 62 11 63 90 32 30 70 69 1 50 76 12 26 28 53 87 97 59 96 99
5 84 60 83 8 82 48 19 64 49 36 47 46 45 17 61 91 100 42 57 2 13

r209
930.281266 3
0 95 59 98 2 15 42 14 44 16 61 5 83 18 7 88 62 11 64 49 36 46 8 45 17 84 6 96 92
37 100 91 97 13 58
0 27 69 31 52 82 47 19 63 90 30 51 71 9 81 33 76 50 3 79 78 34 35 65 66 20 32 70
10 48 60 89
0 28 12 29 39 67 23 75 72 73 21 40 53 57 87 94 99 93 85 86 38 43 41 22 74 56 4
26 54 55 25 24 80 68 77 1

r210
947.543649 3
0 27 52 7 19 36 47 48 82 45 61 16 86 38 44 14 91 85 99 96 87 2 53 40 21 73 72 22
41 75 74 56 4 55 25 54 26 58
0 28 33 71 65 30 1 69 31 88 62 11 64 49 46 8 84 83 18 6 94 13 97 43 100 37 93 17
5 60 89
0 95 59 92 98 42 15 57 23 67 39 12 76 3 79 29 78 81 9 51 20 32 90 63 10 70 66 35
34 24 80 68 77 50

r211
784.698173 3
0 21 39 67 23 75 72 73 2 57 42 87 95 99 59 92 98 85 61 16 86 38 44 14 43 15 41
22 74 40 58 53
0 28 12 76 69 31 88 62 63 90 30 51 9 81 33 79 29 78 34 35 71 65 66 20 32 10 70 1
50 3 77 68 80 24 54 55 25 56 4 26
0 27 52 18 7 19 11 64 49 36 46 47 48 82 8 45 17 84 5 83 60 89 6 94 96 93 91 100
37 97 13

rc101
1696.949157 14

```
0 64 90 84 56 66
0 72 36 38 41 40 43 37 35
0 63 76 51 22 49 20 24
0 28 33 85 89 91
0 65 52 99 57 86 74
0 5 45 2 7 6 8 3 1 70 100
0 27 29 31 30 34 26 32 93
0 82 11 15 16 9 10 13 17
0 14 47 12 73 79 46 4 60
0 69 98 88 53 78 55 68
0 59 75 87 97 58 77
0 83 23 21 19 18 48 25
0 39 42 44 61 81 54 96
0 92 95 62 67 71 94 50 80

rc102
1477.541997 13
0 91 64 86 87 59 97 75 58
0 69 88 99 57 74 52
0 1 3 45 5 8 7 6 46 4 2
0 39 36 40 38 41 43 35 37 72
0 42 44 61 81 54
0 14 47 73 79 78 60 100 70
0 33 28 27 26 31 34 50 93 94 80
0 92 95 62 29 30 32 89
0 65 83 19 23 21 18 48 25 77
0 82 11 15 16 9 10 13 17 12 98
0 96 71 67 84 56 66
0 90 53 55 68
0 85 63 76 51 22 49 20 24

rc103
1263.539437 11
0 61 42 43 44 40 38 41 72 71 93 94 80
0 2 45 1 3 5 8 6 55 68
0 20 18 48 21 23 22 49 19 25 24
0 33 27 30 32 28 26 29 31 34
0 92 62 50 67 84 56 66
0 12 14 15 11 9 87 59 74 86 52 82 90
0 81 39 36 35 37 54 96
0 64 51 76 89 63 85 95 91
0 69 88 53 10 13 16 17 47
0 65 83 57 99 97 75 58 77
0 98 60 73 78 79 7 46 4 100 70

rc104
1145.939275 10
0 69 98 53 9 86 74 57 83
0 82 10 11 15 16 17 47 14 12 13 52 90
0 88 60 78 73 79 7 6 2 55
0 42 44 43 38 37 35 36 40 39 41
0 24 22 49 19 23 21 48 18 25 77
0 66 64 20 51 76 89 63 85 56 91
0 81 61 1 3 5 8 46 45 4 100 70 68
0 65 99 87 59 97 75 58
0 92 50 33 32 30 28 26 27 29 31 34
0 80 95 62 67 84 94 93 71 72 54 96

rc105
1540.181361 14
0 64 86 87 59 97 75 58
0 42 61 81 41 72 54
0 65 99 52 57 74
0 90 53 98
0 83 19 23 18 22 49 20 77
0 12 14 47 15 16 9 10 13 17
0 69 88 79 55 68
0 2 45 3 5 8 6 7 46 4 1 100
0 33 63 85 84 56 66
0 51 76 89 48 21 25 24
```

```
0 31 29 27 30 28 26 32 34 50 91
0 92 95 62 67 71 94 93 96 80
0 82 11 73 78 60
0 39 36 44 38 40 37 35 43 70

rc106
1384.035522 12
0 11 12 14 47 15 16 9 10 13 17
0 42 44 39 40 36 38 41 43 37 35
0 31 29 27 26 28 89 91
0 90 53 55 70 68
0 65 52 87 59 75 97 58 74
0 83 64 19 23 21 18 48 25 77
0 2 45 5 8 7 6 46 4 3 1 100
0 95 62 63 85 76 51 84 56 66
0 92 67 33 30 32 34 50 93 80
0 69 98 88 78 73 79 60
0 82 99 86 57 22 49 20 24
0 71 72 61 81 94 96 54

rc107
1232.262370 11
0 92 95 84 85 63 51 76 89 56
0 31 29 27 28 26 30 32 34 33
0 72 71 93 94 67 50 62 91 80
0 82 9 87 59 75 97 58 74
0 88 12 14 47 17 16 15 13 11 10
0 83 64 19 49 18 21 48 23 25 77
0 2 6 7 8 5 3 1 45 46 4 100
0 65 99 52 86 57 24 22 20 66
0 41 38 39 42 44 43 40 37 35 36
0 69 98 53 78 73 79 60 55 70 68
0 61 90 81 54 96

rc108
1139.821433 10
0 71 67 62 50 63 85 84 92 56 91 80
0 64 51 76 89 18 48 19 20 66
0 69 98 88 53 78 73 79 60 55 70 68
0 90 61 81 96 94 93 72 54
0 82 99 52 86 87 59 97 75 58 74
0 41 42 44 43 40 38 37 35 36 39
0 95 33 30 28 26 27 29 31 34 32
0 2 6 7 8 46 4 45 5 3 1 100
0 12 14 47 17 16 15 13 9 11 10
0 65 83 57 24 22 49 21 23 25 77

rc201
1406.940088 4
0 69 82 52 83 64 19 23 21 18 76 85 84 51 49 22 20 66 56 96 54 37 43 35 93 91 80
0 65 14 47 59 75 16 15 11 12 73 78 79 7 6 8 46 3 4 1 55 68
0 92 95 63 33 28 27 29 31 30 62 67 71 90 99 57 86 87 9 53 10 97 74 13 17 60 100
70
0 42 36 39 72 45 5 2 98 88 61 44 40 38 41 81 94 50 34 32 26 89 48 24 25 77 58

rc202
1367.092177 3
0 98 45 5 3 1 42 36 39 44 69 88 73 16 99 53 78 79 8 6 46 2 55 68 54 43 35 37 72
96 93 94 80
0 65 82 12 14 47 15 11 83 64 23 19 51 76 18 22 57 86 87 9 10 97 59 74 13 17 7 4
60 100 70
0 91 92 95 85 63 33 28 26 27 29 31 30 62 67 71 61 41 38 40 81 90 84 49 20 66 56
50 34 32 89 48 21 24 25 77 75 58 52

rc203
1061.835206 3
0 1 3 5 45 46 7 60 73 16 15 11 69 88 98 99 52 86 87 9 53 78 79 8 6 4 2 55 68 70
100
0 81 61 42 43 36 39 54 96 94 62 64 24 19 23 21 48 18 89 76 63 85 51 49 22 20 84
56 66 82 10 12 14 47 17 13 74 59 97 75 58 77 25 57 83
```

0 90 65 91 92 95 50 33 32 34 31 29 27 26 28 30 67 44 41 38 40 35 37 72 71 93 80

rc204
799.060513 3
0 81 96 54 41 39 42 44 43 40 36 35 37 38 72 71 93 67 84 85 63 33 32 30 28 26 27
29 31 34 50 95 56 64 66
0 69 98 82 10 11 15 16 17 47 14 12 53 60 78 73 79 7 8 46 45 5 3 1 4 6 2 88 55
100 70 61 68
0 80 91 92 94 62 51 89 76 18 23 21 48 22 24 57 99 52 86 87 9 13 74 59 97 75 58
77 25 19 49 20 83 65 90

rc205
1303.804439 4
0 92 95 33 28 27 29 31 30 63 76 85 67 84 51 49 22 20 24 74 13 17 60
0 2 45 5 42 39 36 72 71 62 94 44 40 38 41 61 81 53 82 90 66 56 50 34 32 26 89 48
25 77 58
0 69 98 11 15 16 47 14 12 88 78 73 79 7 6 8 46 4 3 1 70 100
0 65 83 64 19 23 21 18 57 86 52 99 9 87 59 75 97 10 55 68 43 35 37 54 96 93 91
80

rc206
1153.937166 3
0 65 83 92 95 62 31 29 27 28 30 33 76 63 85 51 64 22 23 21 18 19 49 57 86 87 97
9 10 13 17 60 55 100 70 68
0 69 98 82 12 14 47 16 15 11 59 75 52 99 53 88 78 73 79 7 6 8 46 4 3 1 43 35 37
54 96 93
0 2 45 5 44 42 72 39 38 36 40 41 61 81 71 67 94 90 66 56 84 50 34 32 26 89 20 48
25 77 58 74 24 91 80

rc207
1071.249939 3
0 82 99 52 9 11 15 16 47 14 12 53 78 73 79 7 6 8 46 45 3 43 36 35 37 39 41 54 96
0 65 83 64 95 67 31 29 28 30 63 76 51 19 21 18 23 75 59 87 74 86 57 22 20 49 25
77 58 97 13 10 17 60 4 1 70 100 55 68
0 69 98 88 2 5 42 44 40 38 72 71 93 61 81 90 94 92 84 85 62 50 34 27 26 32 33 89
48 24 66 56 91 80

rc208
833.228164 3
0 92 95 64 83 65 82 99 52 9 11 12 14 47 16 15 87 59 75 58 77 25 23 21 48 18 19
49 20 22 24 57 86 74 97 13 10 17 60 46 4 70 100 55 68
0 69 98 88 53 78 73 79 7 6 2 8 45 5 3 1 42 44 43 40 36 35 37 38 39 41 72 54 96
93 91 80
0 90 61 81 71 94 67 62 50 34 31 29 27 26 28 30 32 33 76 89 63 85 51 84 56 66