D-ants: Savings Based Ants Divide and Conquer the Vehicle Routing Problem

Marc Reimann
Karl-Franzens-Universität Graz
77 PUBLICATIONS **1,173** CITATIONS

Karl F. Doerner
University of Vienna
122 PUBLICATIONS **3,388** CITATIONS

Richard F. Hartl
University of Vienna
279 PUBLICATIONS **6,974** CITATIONS

Available online at www.sciencedirect.com
SCIENCE DIRECT®

Computers & Operations Research 31 (2004) 563–591

computers & operations research

www.elsevier.com/locate/dsw

D-Ants: Savings Based Ants divide and conquer the vehicle routing problem

Marc Reimann*, Karl Doerner, Richard F. Hartl

Department of Production and Operations Management, Institute of Management Science, University of Vienna, A-1210 Vienna, Austria

Abstract

This paper presents an algorithm that builds on the Savings based Ant System presented in [Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002), Morgan Kaufmann, San Francisco, 2002] and enhances its performance in terms of computational effort. This is achieved by decomposing the problem and solving only the much smaller subproblems resulting from the decomposition.

The computational study and statistical analysis conducted both on standard benchmark problem instances as well as on new large scale Vehicle Routing Problem instances will show that the approach does not only improve the efficiency, but also improves the effectiveness of the algorithm leading to a fast and powerful problem solving tool for real world sized Vehicle Routing Problems.
© 2003 Elsevier Ltd. All rights reserved.

Keywords: Meta-heuristics; Ant System; ACO; Decomposition; Vehicle Routing Problem

1. Introduction

Goods distribution contributes approximately 20% to the total costs of an average product. This is caused by the fact that transportation occurs between any two subsequent members of a goods' supply chain and also between the chain and the final consumers. Both firms and academic researchers have recognized the huge potential for optimization in this area.

Formally, most problems in the domain of goods distribution can be viewed as Vehicle Routing Problems (VRP). The objective of the VRP is to find a set of minimum cost routes, starting and ending at a single depot and serving the known demands of a number of customers. Each customer must be served exactly once, i.e. order splitting is not an option. Furthermore, vehicle capacity and

* Corresponding author.
E-mail addresses: marc.reimann@univie.ac.at (M. Reimann), karl.doerner@univie.ac.at (K. Doerner), richard.hartl@univie.ac.at (R.F. Hartl).

0305-0548/04/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0305-0548(03)00014-5

(possibly) driving time restrictions must not be violated. A formulation of this problem can be found in [1].

The VRP has been studied extensively for the last 40 years. As an NP-hard problem real world sized problem instances cannot be solved to optimality within reasonable time (in fact there are no exact algorithms available that consistently solve problems with more than 50–75 customers, see [2]). For more information about the theory of NP-completeness we refer to [3].

However, enormous progress was made concerning approximate solution techniques. Starting with 'simple' constructive mechanisms such as the Clarke and Wright Savings algorithm [4] or basic improvement methods such as the 2-opt local search [5] researchers have turned to ever more complicated methods. This transition has led to the development of so-called meta-heuristics, which guide subordinate heuristics to avoid or overcome local optimality. To date the best approaches for the VRP are based on Tabu Search (TS). In [6] we have presented a Savings based Ant System (SbAS) and have shown that this approach is competitive to the best TS algorithms. However, while these approaches (TS or SbAS) find excellent solutions, their computational requirements are huge compared to the 'simple' techniques of the early years. More specifically, most of the meta-heuristic approaches show poor scaling behavior such that the runtime for real world sized problem instances is prohibitive. For example, for 50 customer instances our SbAS was able to find solutions with 0% relative deviation from the best known results within 3 s. Solving problems with 199 customers took up to 40 min and the obtainable solution quality was almost 2% off the best known solution. Similar scaling applies to most TS approaches.

Thus, most commercial tools for solving VRP like problems of real world size are based on rather simple constructive or improvement techniques. Recently, various researchers have acknowledged this fact and various methods to speed up the search (cf. e.g. [7,8]) have been proposed.

On the other hand, commercial use of optimization tools depends not only on the obtainable solution quality and the time required to reach a solution, but also on the flexibility and the simplicity of an approach (cf. [9]). Most meta-heuristics also score low on these two dimensions when compared to classic heuristics.

Apart from these algorithmic observations we learned from our industry partners that human dispatchers tend to cluster customers according to postal codes or other regional characteristics—like rivers, highways or mountains—before they solve the much smaller subproblems for each cluster separately.

These above-mentioned issues led to the development of our new algorithm. The main idea is to decompose large problems into a number of smaller problems that can be solved both more effectively (i.e. with a better solution quality) and more efficiently (i.e. with smaller runtime) using our SbAS. In the remainder of this paper, we will refer to our approach as D-Ants (for Decomposition-Ants). Note, that a related decomposition approach was used in combination with a TS algorithm by Taillard (cf. [7]), where the TS procedure employed solves only some subproblems. Besides the obvious difference in the solution technique used—TS versus Ant System—the approaches differ with respect to the decomposition method.

The aim of our paper is threefold. First, we want to show that our D-Ants find competitive solutions to the best TS algorithms both in terms of solution quality and computation times for small problem instances. Second, for new large scale problem instances we want to show that our algorithm scales well, leading to some new best found solutions and 'short' computation times. Thus, we aim to establish our Ant System as an alternative approach for VRPs. Third, we target flexibility and simplicity. Our decomposition approach is based on the closeness of vehicle routes. It can be easily understood and implemented. Further, it can without modifications be applied to a number of VRPs with additional side constraints.

The outline of the remainder of this paper is as follows. In the next section, we discuss related literature for the VRP with particular emphasis on two approaches that explicitly aim to speed up the search, namely the algorithm of Taillard (cf. [7]) and the approach of Toth and Vigo (cf. [8]). After that we focus on the detailed description of our new approach in Section 3. We will present results of a comprehensive computational study in Section 4 before we conclude with an outlook on future research.

2. Solving (large scale) VRPs

In this section we will review some related approaches for the VRP. More specifically, we will first review two constructive heuristics, namely the Clarke and Wright Savings algorithm proposed in [4] and the Sweep algorithm due to Gillett and Miller (cf. [10]) as these algorithms are incorporated in our D-Ants approach.

We will then briefly discuss the latest TS developments for the VRP. In this context we will particularly focus on two different approaches to efficiently solve (large scale) VRPs by using different techniques to speed up the search. The first one, namely Taillard's algorithm (cf. [7]) is based on problem decomposition. The second one, namely the Granular TS by Toth and Vigo (cf. [8]) is based on a significantly reduced graph of the complete problem.

2.1. Constructive heuristics for the VRP

One of the most elegant algorithms for the VRP is the Savings algorithm proposed in 1964 by Clarke and Wright (cf. [4]). Starting from an initial solution, where all customers i are assigned to separate tours 0–i–0, the savings of combining any two customers i and j are computed as

$$s_{ij} = d_{i0} + d_{0j} - d_{ij},$$

where d_{i0} corresponds to the distance between customer i and the depot 0. The resulting savings values are then sorted in decreasing order. Iteratively, customers are combined to partial tours according to the sorted savings list until no more combinations are feasible. A combination is infeasible if it violates either the capacity or the tour length restrictions.

Another constructive algorithm for the VRP is the Sweep algorithm proposed by Gillett and Miller (cf. [10]). The algorithm requires the coordinates of the customers and the depot as inputs. Given these coordinates and an arbitrarily chosen seed customer, the remaining customers are sorted according to the polar angle they span with the depot and the seed customer. Iteratively, the algorithm extends its current solution by taking the next customer from the list and either adding it to the current tour (if this is feasible) or closing the current tour and initializing a new one with the chosen customer. Once all customers are assigned to tours the algorithm stops. In fact, it is customary to run the algorithm starting with each customer as a seed customer once and taking the best solution.

The two algorithms can be extended by applying some local search to the resulting tours. The main difference between the algorithms lies in the fact, that the Savings algorithm builds tours in parallel, while the Sweep algorithm considers only one tour at a time.

2.2. Recent TS algorithms for the VRP

As stated in the introduction, the most powerful approaches for the VRP are meta-heuristics and among those TS algorithms. The main idea of TS is to iteratively change a given solution of the problem according to a predefined neighborhood function. As long as improving moves are possible within the neighborhood of the current solution the algorithm resembles a simple descent to a (local) optimum. However, once a local optimum is reached the algorithm accepts non-improving moves to leave this local optimum. Due to the deterministic neighborhood search specific for TS, this would lead to cycling around the local optimum such that a so-called short term memory is activated, which prevents previously made moves to be reversed. For more details about TS we refer to [11].

The basic setup of TS has been augmented and extended to improve the performance of the algorithm. Most prominently, intensification and diversification techniques have been added, to either concentrate the search on a promising region or divert it from an area of the search space that has been heavily examined.

One of the first applications of TS to the VRP is due to Osman (cf. [12]). Osman used very simple neighborhood operators and allowed only feasible solutions during the whole process. The first algorithm to exploit infeasible solutions was the Taburoute algorithm proposed in [13]. In that algorithm, infeasible solutions were allowed during the search, but were penalized in the objective function. To return the search from an infeasible to a feasible region of the search space, the penalties were adjusted adaptively during the search. Another major conceptual add-on to the basic TS procedure was the proposal of an Adaptive Memory by Rochat and Taillard (cf. [14]). This Adaptive Memory stores characteristics of good solutions, more specifically tours which are part of good solutions. Each time a new best solution is found, the Adaptive Memory is updated. Frequently, tours are extracted from the Adaptive Memory and combined to re-initialize the TS with a ‘good’ solution.

Another direction of research was concerned with more complex neighborhoods. Rego and Roucairol (cf. [15]) proposed the use of ejection chains as a neighborhood operator. Moreover, they proposed a parallel implementation of a TS for the VRP.

More recently, some other TS algorithms were proposed for the VRP, which are mainly based on combinations of the issues discussed above. For a review we refer to [16]. Let us now turn to two approaches that explicitly address scaling issues both with respect to solution quality and computational requirements.

2.2.1. Taillard’s decomposition algorithm

Taillard (cf. [7]) realized that good solutions to most VRP instances of moderate to large size feature some spatial characteristics that allow to exploit problem decomposition. In terms of TS or more generally Local Search it is obvious that (local) changes only make sense if the nodes involved are close to each other. Moreover, simple change operators generally involve only two tours. Thus, different moves can be evaluated and performed simultaneously.

Based on these observations, Taillard suggests to decompose the problem and proposes two distinct methods for partitioning a problem instance. For uniform problems, a partition into sectors is suggested, while for non-uniform problems Taillard uses a partitioning method based on trees and associated shortest path. The core of the algorithm consists of the iteration of the following steps. First, the problem is partitioned with the appropriate method (as mentioned above). Each partition is then solved using a standard TS approach. After a certain number of iterations the subproblems are re-joined.

The main feature of these approaches is that the partitioning is done on the level of the individual customers, by exploiting the closeness of these customers.

Through these methods, Taillard was able to find almost all the best known solutions for the classic benchmark instances for the VRP using a simple implementation of TS. However, the algorithm's performance depends crucially on the initial decomposition and the partitioning method is problem dependent. Further, for problems with additional constraints such as time windows the algorithm has to be significantly adapted as in these problems geographical closeness of customers has to be traded off against closeness in time.

2.2.2. Toth and Vigo's Granular TS

Recently, Toth and Vigo (cf. [8]) have analyzed the best solutions for the classic VRP. They found that the average length of arcs in these best solutions is significantly smaller than the average length of arcs in the complete problem graph. More precisely, the vast majority of arcs in the problem graph is prohibitively long to be included in the solution.

On the other hand, the most successful TS, or more general Local Search algorithms (e.g. [13]) work with various, sophisticated neighborhood structures that lead to large neighborhoods. These large neighborhoods are necessary to make sure that the algorithms reach solutions close to the global optimum. However, using such neighborhoods leads to large computation times that render these TS approaches useless for real world applications.

Based on these observations Toth and Vigo propose a reduction of the search space, by ruling out 'unpromising' options. That way, sophisticated neighborhoods can be used and still the computation times are small as only the most promising alternatives have to be considered. More specifically, Toth and Vigo initially reduce the problem graph by eliminating all arcs that exceed a certain threshold level. This threshold level is determined by estimating the average arc length expected in a good solution. Note, that arcs to and from the depot are excluded from this elimination to ensure feasible solutions. Toth and Vigo then apply a TS algorithm to this reduced problem. Repeatedly the threshold is updated and the problem graph is augmented, by eliminating some arcs and adding others.

The results using this approach are excellent. Toth and Vigo report solution qualities for the classic benchmark problem instances comparable to those obtained by the previous best approaches. However, computation times are smaller by an order of magnitude. This is true even after adjusting for the differences in the machines used for the experiments. For the large scale instances Toth and Vigo also find reasonable results in very short computation times. We will discuss their results for these large scale instances below when we compare our algorithm with other approaches.

The problem associated with this approach is the appropriate choice of the threshold level. In fact, an inappropriate choice can exclude the global optimum from the search space. Further, the threshold will be problem dependent. Toth and Vigo address this problem by automatically adjusting the threshold parameter frequently.

A second problem concerns the applicability of the approach to other VRPs with additional constraints. Again, for the VRP with Time Windows, the choice of the thresholds may be tedious as arcs in good solutions cannot simply be evaluated by their length but also depend on the time windows of the adjacent customers.

3. D-Ants for VRPs

As stated above, the D-Ants approach builds on the SbAS we proposed in [17] and improved in [6]. For the sake of completeness let us now briefly review the SbAS.

3.1. A Savings based AS for the VRP

Ant Systems (AS) have been first proposed in [18] and are based on stigmergetic learning. More precisely, a population of artificial agents repeatedly constructs solutions to the problem using a joint population memory and some heuristic information. After each member of the population has constructed its next solution, the memory is updated with a bias towards better solutions found. Gradually, the memory will build up, thus gaining stronger influence on the solutions built by the artificial agents and the solutions will evolve towards the global optimum. Convergence proofs for generalized versions of AS algorithms can be found in [19–21].

Various versions and frameworks of the basic AS algorithm have recently been proposed (cf. [22–26]). Applications cover the TSP, graph coloring, quadratic assignment problems, scheduling and vehicle routing. For an overview see [27].

Ant System algorithms mainly consist of the iteration of three steps:

- generation of solutions by ants according to private and pheromone information,
- application of a local search to the ants' solutions,
- update of the pheromone information.

In addition to that our approach features a fourth step, namely:

- Augmentation of the Attractiveness list, which stores the desirability of all feasible combinations.

The implementation of these four steps is described below. As for all other meta-heuristics, the stopping criterion can either be static, i.e. a fixed number of iterations or dynamic, i.e. a fixed number of non-improving iterations. When using our SbAS without the divide and conquer strategy, we apply a static stopping criterion, namely a fixed time limit. However, as the sizes of the instances vary, this time limit will be tight for some instances while it will not be constraining for others. Thus, we also use a dynamic restart criterion that restarts the algorithm after n non-improving iterations, as long as the time limit is not reached. Here, n denotes the number of customers.

3.1.1. Generation of solutions

Prior to our work, the solution generation mechanism of the AS for the VRP was the Nearest Neighbor heuristic (see e.g. [28,29]). As opposed to that we use the Savings algorithm described in the last section to generate solutions. To that end we need to modify the deterministic version of this algorithm. We will now explain how this modification was done in order to be able to use it in the Ant System context.

Our modifications are related to the use of pheromone information in the decision-making. Initially, we generate a sorted list of attractiveness values ξ_{ij} in decreasing order. These attractiveness values feature both, the Savings values as well as the pheromone information.

Thus the list consists of the following values:

$$\xi_{ij} := [s_{ij}]^{\beta}[\tau_{ij}]^{\alpha}, \tag{1}$$

where τ_{ij} denotes the pheromone concentration on the arc connecting customers i and j, and α and β bias the relative influence of the pheromone trails and the savings values, respectively. The pheromone concentration τ_{ij} contains information about how good the combination of two customers i and j was in previous iterations.

In each decision step of an ant, we consider the k best combinations still available, where k is a parameter of the algorithm which we will refer to as 'neighborhood' below.

Let Ω_k denote the set of k neighbors, i.e. the k feasible combinations (i, j) yielding the largest savings, considered in a given decision step, then the decision rule is given by Eq. (2), where $\mathscr{P}_{ij}$ is the probability of choosing to combine customers i and j subsequently on one tour:

$$\mathscr{P}_{ij} := \begin{cases} \dfrac{\xi_{ij}}{\sum_{(h,l)\in\Omega_k} \xi_{hl}} & \text{if } \xi_{ij} \in \Omega_k, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The construction process is stopped when no more feasible combinations are possible.

3.1.2. Local search

After the ants have constructed their solutions but before the pheromone is updated each ants' solution is improved by applying a local search. We first apply a local search based on *swap* moves to an ants solution. The use of *swap* moves was proposed in [12] for the VRP. A *swap* move aims at improving the clustering of the solution by exchanging two customers from different tours, i.e. a customer i from tour k is exchanged with a customer j from tour l.

Following [29] we then apply the 2-opt algorithm (cf. [5]) originally developed for the TSP. This algorithm iteratively exchanges two edges with two new edges until no further improvements are possible. In the context of the VRP it is applied separately to all vehicle routes built by the ants, thus aiming to improve the routing of each tour, while leaving the clustering unchanged. Thus, we first improve the clustering and once no more improvements are possible, we improve the routing.

3.1.3. Pheromone update

After all ants have constructed their solutions and all ants' solutions have been taken to a local optimum, the pheromone trails are updated on the basis of the local optimal solutions. According to the rank based scheme the pheromone update is as follows (cf. [22]):

$$\tau_{ij} := \rho\tau_{ij} + \sum_{r=1}^{n_e-1} \Delta\tau_{ij}^{r} + n_e\Delta\tau_{ij}^{*}, \tag{3}$$

where $0 \leqslant \rho \leqslant 1$ is the trail persistence and n_e is the number of elitists. Using this scheme two kinds of trails are laid. First, the best solution found during the process is updated as if n_e ants had traversed it. The amount of pheromone laid by the elitists is $\Delta\tau_{ij}^{*} = 1/L^{*}$, where L^{*} is the objective value of the best solution found so far. Second, the $n_e - 1$ best ants of the iteration are allowed to lay pheromone on the arcs they traversed. The quantity laid by these ants depends on their rank r as well as their solution quality L^{r}, such that the rth best ant lays $\Delta\tau_{ij}^{r} = (n_e - r)/L^{r}$. Arcs belonging to neither of those solutions just lose pheromone at the rate $(1 - \rho)$, which constitutes the trail evaporation.

3.1.4. Augmentation of the savings list

After the pheromone information has been updated the attractiveness values ξ_{ij} are augmented with the new pheromone information as in Eq. (1).

3.2. The Divide and conquer framework

Our strategy for solving large scale VRPs is based on the fact that the VRP is a generalization of the TSP. Thus, besides the routing aspect already existing in the TSP one has to find an assignment (or clustering) of customers to vehicles. Once this assignment is done, the problem reduces to independently solving a number of TSPs, one for each vehicle tour. As large problems generally consist of a significant number of tours, and these tours will cover distinct geographical areas, we propose to decompose the set of tours that constitute the complete problem into a number of smaller sets of geographically close tours, and to solve these resulting subproblems using the SbAS described above. Let us now turn to a more detailed description of our D-Ants. A high level description of the D-Ants algorithm is given in Table 1.

Initially, an Ant System solves the master problem for a given number of iterations (Step I). Given the best found solution so far our algorithm determines for each route of this solution the center of gravity according to the modified Miehle algorithm (cf. [30]) (Step II) . We then cluster these route centers using the Sweep algorithm as proposed by Gillett and Miller (cf. [10]) (Step III). Each of the resulting clusters is then solved independently by applying our SbAS for a given number of iterations (Step IV). After all subproblems have been solved we re-assemble the global solution and update the global pheromone information and if applicable the global best solution (Step V). The Steps I–V described above are repeated until a pre-specified time limit is reached.

As some of the instances can be solved much faster, we have modified our D-Ants algorithm by allowing restarts. The D-Ants restart, if the Ant System that solves the complete problem converges to the best solution found during the run. In preliminary tests we identified that most of the improvements are found by the Ant System processes solving the subproblems. Once the master converges to the best found solution in the sense that it finds the best found solution in three consecutive iterations only minor improvements can be expected and the time between any two improvements can be large.

Let us now turn to each of the five main steps of our D-Ants in more detail, where we will also discuss implementation issues.

- *Step* I: Generation of initial solutions.
  The initial solutions are generated by applying the SbAS as described in detail above.

Table 1
D-Ants procedure

```
procedure D-Ants{
    Read the input data;
    Initialize the system (parameters and global pheromone matrix);
    repeat {
        for a pre-specified number of iterations {
            Solve the complete problem using the Savings based Ant System; (Step I)
        }
        for the best solution found so far {
            Compute the center of gravity of each route; (Step II)
        }
        Decompose the best solution into a pre-specified number of subproblems
        by applying the Sweep algorithm to the centers of gravity; (Step III)
        for each subproblem {
            for a pre-specified number of iterations {
                Solve the subproblem with the Savings based Ant System
                by using the relevant part of the global pheromone matrix locally;(Step IV)
                If applicable, update best solution; (Step V)
            }
        }
        Update the global memory (global pheromone matrix);
    } until a pre-specified stopping criterion is met;
}
```

The argument for using the decomposition approach was that solving the complete problem is too time consuming. Clearly, the larger a problem, the more iterations it will take to solve this problem to a certain level of quality. However, a few or even a single iteration of the algorithm can be performed in very short time. As we will see below in our results section, we solve the master problem only a few times and we will still be able to exploit the pheromone information. This is due to the fact, that we only use a single global pheromone information that is used by both the master process and the processes that solve the subproblems. We will discuss this interaction in more detail in Step V below.

Given the best solution found by the algorithm so far, we next try to decompose this solution by tours. Clearly, we want to find a decomposition that leads to subproblems with geographically close tours in order to be able to then solve these regional subproblems efficiently. This should help us to improve both the routing on these tours but also the clustering. We use elitism in this decomposition approach by always considering the best solution found so far. Note that the intuition for this is twofold. First, it can be expected that solving the subproblems often leads to an improvement of the best found solution. This was also confirmed by the computational experiments we performed and report on below. Second, even if the best known solution remains unchanged, it is well known that the Sweep algorithm performs best if it is applied a number of times, each time starting with a different seed node. As we draw the seed node randomly, we exploit this knowledge through the elitism used in decomposition.

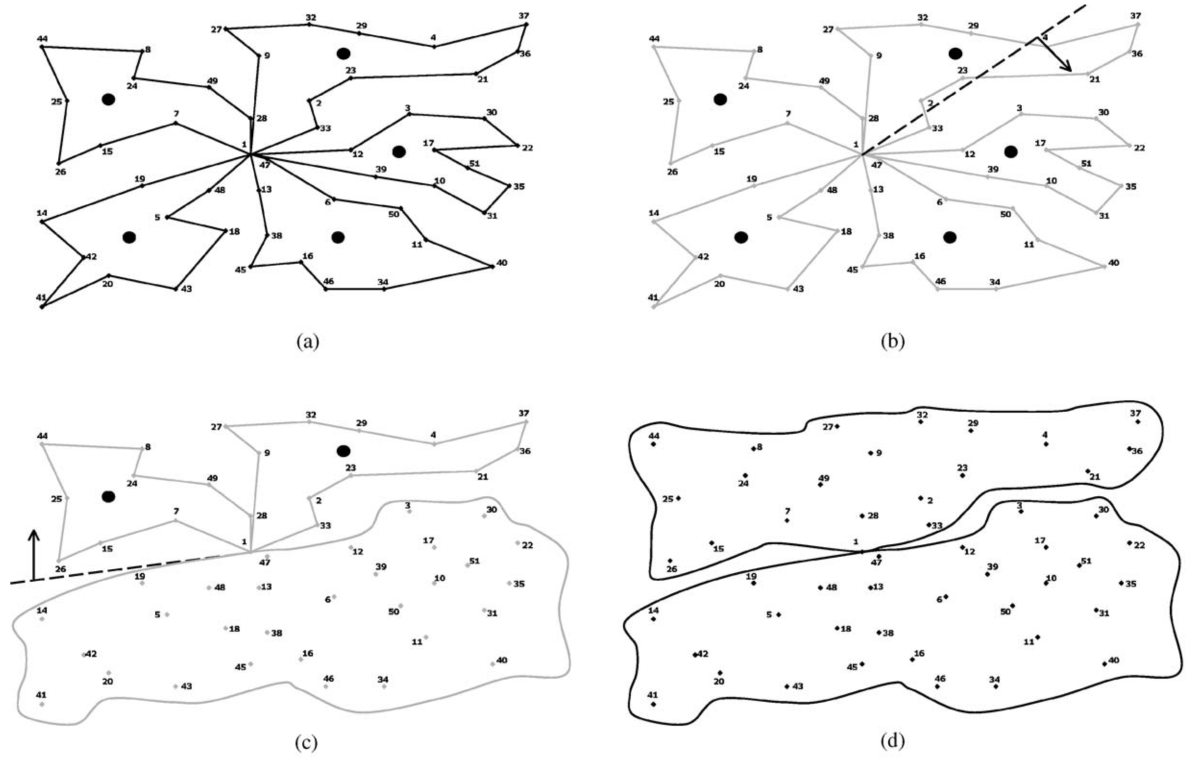

Fig. 1. Decomposition steps of our D-Ants.

In order to be able to perform the decomposition, we need to define and compute closeness of tours. This is done in the following way. We define closeness of two tours by the polar angle between the centers of gravity of the tours and the depot. We then compute closeness using the Sweep algorithm to cluster the centers of gravity. Let us first turn to the computation of the centers of gravity, before we review the Sweep algorithm.

- *Step* II: The modified Miehle algorithm
  The modified Miehle algorithm is used to compute the center of gravity for each vehicle route of the best solution found so far. This algorithm iteratively adjusts the coordinates of the center, until the change in the weighted distance of all customers to the center is minimal. More specifically, the distances are weighted by the demands of the customers. The term *modified* in the name denotes that the algorithm can handle a situation where the center of gravity coincides with the location of a customer. Possibly we could also ignore the demands of the customers. However using the demands causes no increasing computational effort and yields some extra information. For a detailed description of the modified Miehle algorithm we refer to [30]. Fig. 1a shows an arbitrary solution for a 50 customer instance consisting of five tours. For each tour, the black dot represents the center of gravity.

- *Step* III: The Sweep algorithm
  Having computed the centers of gravity for each vehicle route, we then ignore the actual customer locations and apply the Sweep algorithm (cf. [10]) to cluster the nodes corresponding to the centers of gravity. In Fig. 1b only the centers of gravity are highlighted, thus emphasizing that the actual customer nodes are not considered in the Sweep step.
  The number of clusters is pre-defined by choosing the number of subproblems the master problem is to be decomposed to. Given that the number of clusters is known, we then compute the number of tours to be assigned to each cluster by dividing the total number of tours by the number of clusters and taking the next largest integer. More formally, let n_t be the number of tours in the best solution, and n_s be the number of subproblems then the number of tours per subproblem is given by

$$n_{\mathrm{s,t}} = \left\lceil \frac{n_\mathrm{t}}{n_\mathrm{s}} \right\rceil . \tag{4}$$

  Note that this may lead to problems. Consider the case where four subproblems should be built and the solution consists of six tours. Clearly the result of applying Eq. (4) is two. However, by assigning two tours to each subproblem we end up with three subproblems. To overcome this problem, we require that each subproblem receives at least one tour. Thus, in the given example we would end up with two subproblems each assigned two tours, and two subproblems each assigned one tour.
  In the original implementation of the Sweep algorithm each node is used as a seed node once leading to a number of solutions where the best one is chosen. In our approach the clustering is done in order to then be able to 'optimize' the resulting subproblems. This optimization is done using a meta-heuristic, hence starting with each center of gravity as a seed node once would be computationally expensive. Thus, we choose a starting node randomly and the remaining nodes are sorted according to their polar angle with the depot and the randomly chosen starting node. Using the resulting order, the nodes are then assigned to the clusters. This is shown in Fig. 1b and c. In Fig. 1b, we randomly place the sweeping beam thus determining the seed node. Fig. 1c shows a situation where the first subproblem has been defined.
  As discussed above we might end up using several different partitions for the same master solution due to the elitism we use in choosing the solution to be decomposed.
- *Step* IV: Solution of the subproblems
  Subproblems are solved using the SbAS already utilized in Step I. However, in this step each Ant System solves only the partial problem it is assigned in Step III. This can be seen from Fig. 1d, where one subproblem consists of the customers in the upper bubble, while the other subproblem consists of the customers in the lower bubble. Clearly the depot occurs in all subproblems.
- *Step* V: Communication between processes
  The main notion with respect to the communication between the different processes is *master pheromone information*. In fact, we use one global memory for our algorithm.
  The communication between the subproblems and the master process is based on two important components. First, each subproblem receives only those parts of the master pheromone information necessary to solve its part of the problem. The subproblems then change this pheromone information *locally* as they iteratively solve their instances.

Second, after a subproblem has been solved, the corresponding process returns the best found solution and compares it with the previous best found solution for the corresponding part of the master problem. If an improvement was achieved, the best found solution is updated and pheromone reinforcement in the master pheromone information occurs. This additional pheromone update reflects the communication between the master and the subproblems. In fact, any improvement in one of the subproblems will trigger the reinforcement of this new improved solution in the global pheromone information. Using this mechanism, the master process can converge to a good solution using far less (master) iterations than the standard rank based Ant System without decomposition. Thus, computation times can be significantly reduced.

Note that this algorithm can be applied to any vehicle routing problem. In particular, it can be applied to the VRP with Time Windows, the VRP with Backhauls or the VRP with Backhauls and Time Windows without modifications. It could also be applied to multi-depot VRPs without modifications. However, for these problems one could also consider a decomposition into several single-depot problems first. These single-depot problems could then be further decomposed according to the scheme shown in Table 1.

4. Numerical analysis

In this section, we will report on a large number of experiments we performed to test our D-Ants. The line of argument we follow is set out to answer the following questions:

1. Does problem decomposition contribute favorably to the obtainable solution quality and to reducing the computational effort required by our SbAS?
2. Do our D-Ants find solutions which are competitive to those provided by state-of-the-art TS approaches?
3. Are our D-Ants a valid alternative to established approaches when applied to real world sized problem instances?

The first question relates to the contribution of design issues to the performance of our algorithm. This question will be answered by comparing the SbAS without decomposition with the D-Ants under different settings for the number of subproblems n_s solved.

The second question looks at the absolute performance of our D-Ants when compared with the best alternative approaches.

These two questions will be answered by analyzing results for the 14 classic VRP benchmark instances, as for these instances many different algorithms were tested within the last 20 years.

The third question relates to the main issue we pointed out in the introduction, namely the applicability of meta-heuristics to problems of real world size. To answer this question, our analysis will be based on results for 20 relatively new large scale problem instances.

4.1. Benchmark problem instances

As lined out above, our experiments were based on two sets of problem data. Let us now, in turn, describe these problem instances.

Table 2
Characteristics of the benchmark problem instances

Instance	n	Q	L	δ	Best publ.
Random problems					
C1	50	160	∞	0	524.61 [7]
C2	75	140	∞	0	835.26 [7]
C3	100	200	∞	0	826.14 [7]
C4	150	200	∞	0	1028.42 [7]
C5	199	200	∞	0	1291.45 [14]
C6	50	160	200	10	555.43 [7]
C7	75	140	160	10	909.68 [7]
C8	100	200	230	10	865.94 [7]
C9	150	200	200	10	1162.55 [7]
C10	199	200	200	10	1395.85 [14]
Clustered problems					
C11	120	200	∞	0	1042.11 [7]
C12	100	200	∞	0	819.56 [7]
C13	120	200	720	50	1541.14 [7]
C14	100	200	1040	90	866.37 [7]

Note: n, number of customers; Q, vehicle capacity; L, maximum tour length; δ, service time; best publ., best published solution.

4.1.1. *Classic benchmark problem instances*

The classic set of benchmark problems was first described in [31]. The instances can be found at http://www.ms.ic.ac.uk/jeb/orlib/vrpinfo.html. Information on these instances is collected in Table 2.

4.1.2. *Large scale benchmark problem instances*

The new problem instances are comprised of the 20 large scale instances proposed in [19]. All of the instances feature one central depot, the customers are located either in concentric circles or squares around the depot. Table 3 provides some more details on the benchmark data sets. These instances can be obtained from the authors upon request.

4.2. *Parameter settings*

Having described the problem instances our analysis will be based upon, let us now turn to another important issue, namely parameter settings.

Parameter fine tuning is a science in itself and should generally be done on instances different from those that are actually used for computational comparisons. Apart from that the goal should be to find some robust parameters that allow an algorithm to find good solutions for a wide range of problem instances with different features. Acknowledging that and trying to keep things simple we start out by sticking to the parameter values that were found to be favorable in [4] for the rank based AS in general, and in [6] for the SbAS in particular. More precisely, these are n ants, $n_e = 3$ elitists, a trail persistence of $\rho = 0.95$, a neighborhood size of $k = n/4$ and $\alpha = \beta = 5$.

Table 3
Characteristics of the new large scale benchmark problem instances

Large scale VRP instances					
Instance	n	Q	L	δ	Best publ.
1	240	550	650	0	5646.63 [33]
2	320	700	900	0	8447.92 [33]
3	400	900	1200	0	11,036.22 [33]
4	480	1000	1600	0	13,624.52 [33]
5	200	900	1800	0	6460.98 [33]
6	280	900	1500	0	8412.80 [33]
7	360	900	1300	0	10,195.59 [33]
8	440	900	1200	0	11,828.78 [33]
9	255	1000	∞	0	587.09 [32]
10	323	1000	∞	0	746.56 [32]
11	399	1000	∞	0	932.68 [32]
12	483	1000	∞	0	1133.79 [33]
13	252	1000	∞	0	868.8 [8]
14	320	1000	∞	0	1086.24 [33]
15	396	1000	∞	0	1363.34 [8]
16	480	1000	∞	0	1650.42 [8]
17	240	200	∞	0	709.9 [8]
18	300	200	∞	0	1014.80 [33]
19	360	200	∞	0	1376.49 [33]
20	420	200	∞	0	1846.55 [33]

Note: n, number of customers; Q, vehicle capacity; L, maximum tour length; δ, service time; best publ., best published solution.

Nonetheless, our D-Ants feature the following parameters that need to be adjusted:

- it_{master}, number of iterations to solve the master problem;
- it_{sub}, number of iterations to solve each subproblem;
- t_{tot}, total admissible runtime;
- n_{s}, number of subproblems.

These parameters influence the performance of the algorithm both in terms of solution quality and computational effort. Let us first consider computational effort. Clearly, an increase in the first three parameters given above will increase the computational effort required, while an increase in the number of subproblems should—all else equal—reduce the computational effort, as smaller subproblems can be solved more efficiently (as discussed above).

Concerning solution quality, the effect of the first three parameters should be in the same direction. Increasing the number of iterations and/or the total admissible runtime will—up to a certain level—improve solution quality. The effect of the number of subproblems on solution quality is not clear and will be studied below.

The statements made in the last paragraph have to be clarified with respect to the basis of comparison. Clearly, increasing the number of iterations devoted to solving the master problem should ultimately (after ‘enough’ iterations) improve solution quality. However, as discussed in the introduction to this paper, the number of iterations devoted to solving the master problem should be kept small to find solutions within reasonable time, in particular for problems of real world size. So there is a tradeoff between solution quality found at the end of the runtime and solution quality found after short time.

In the remainder of this section we will proceed as follows. We will set $it_{\text{master}} = 1$. Thus, in each main loop of our algorithm we solve the complete problem for one iteration before we decompose it into the subproblems. The other important parameter we will keep fixed is the number of iterations assigned to the Ant System processes that solve the subproblems. We set $it_{\text{sub}} = 75$.

These two parameter settings were found to be reasonable in preliminary runs. More specifically, the number of iterations assigned to solving the subproblems has shown to be important for the solution quality that can be obtained by the algorithm. If the number of subproblem iterations is too small, the subproblems will be solved insufficiently well and the algorithm may converge to a poor solution. If the number of subproblem iterations is too large, the algorithm will waste resources and converge too slowly. The settings chosen provide a good compromise with respect to this tradeoff and have shown to work well independent of the problem instance solved.

Having fixed these two parameters we will consider and analyze different settings for the number of subproblems n_s in order to answer the first question posed above.

The total admissible runtime t_{tot} clearly depends on the type of question we want to answer. For a comparison with state-of-the-art competing approaches we will have to consider computation times which are comparable at least with respect to the order of magnitude. For all design issues we will analyze solution quality evolution over the runtime.

4.3. *Computational experiments*

All our computational experiments were performed on a Pentium with 900 MHz. The codes were written in C and run under Windows 2000.

4.3.1. *Results based on classic benchmark problem instances*

Q1. *Does problem decomposition contribute positively to the obtainable solution quality and computational effort required by our Savings based Ant System*?

To answer this question we compare the results obtained for the following runs. We have applied the SbAS and the D-Ants with $n_s = 2,3$ and 4 to the 14 classic benchmark instances. For each instance we have run all our algorithms 10 times and the results presented below are based on averages over these 10 runs. Further, we have restricted all runs to 10 min. To get a first impression about the influence of problem decomposition let us look at Fig. 2.

Fig. 2 shows for the problem instances with 199 customers the average evolution of the solution quality. Clearly, the SbAS converges very slowly, and is not competitive within the time limit of 600 s. Dividing the problem into two subproblems yields an improvement both in solution quality and evolution over time. Further dividing the problem leads to even faster improvements of the solution quality, while the long run behavior does not change radically anymore. Thus, the main effect of increasing the number of subproblems n_s is to find good solutions more quickly.

A more detailed analysis of the runtime behavior is given in Appendix A, where we provide information about the computation times required to reach solutions of predefined quality for each

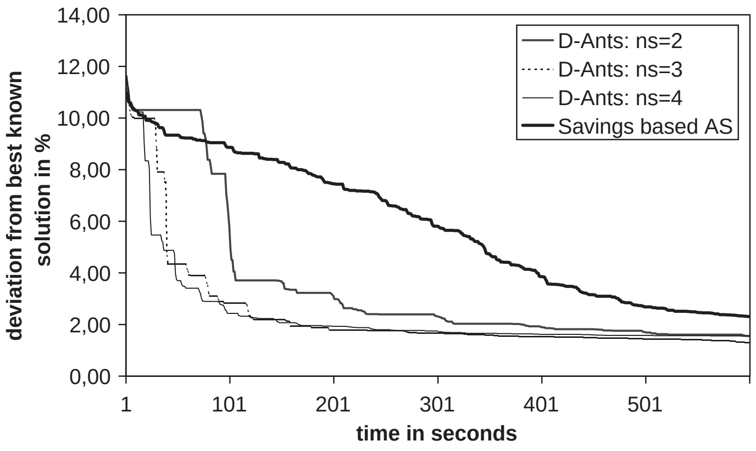

Fig. 2. Evolution of solution quality obtained by the SbAS and the D-Ants for the instances with 199 customers.

Table 4
Effects of the number of subproblems n_s on the solution quality (RPD) after 600 s

Problem		D-Ants		
size	SbAS	$n_s = 2$	$n_s = 3$	$n_s = 4$
50	**0.00**	**0.00**	**0.00**	–
75	**0.40**	0.52	0.64	0.68
100	0.75	**0.24**	0.25	0.43
150	1.72	0.95	**0.89**	1.20
199	1.56	1.38	**1.21**	1.50
50	**0.00**	**0.00**	**0.00**	–
75	**0.20**	1.11	0.86	0.89
100	0.17	0.04	**0.00**	0.03
150	1.55	**0.60**	0.98	0.77
199	3.07	1.74	**1.41**	1.60
120	0.10	**0.06**	0.13	0.09
100	**0.00**	**0.00**	**0.00**	**0.00**
120	**0.25**	0.28	0.37	0.30
100	**0.00**	**0.00**	**0.00**	**0.00**
Averages	0.70	0.49	0.48	n.a.

of the 14 benchmark instances and all our algorithms. The main finding is that the ‘best’ number of subproblems increases with increased problem size.

Now, we turn to the contribution of the problem decomposition after 600 s for all problems. Table 4 provides information on the average deviation from the best known solutions (RPD—relative percentage deviation) for the SbAS and the D-Ants with different values of n_s after 10 min runtime. Bold entries indicate the best average deviation for each instance.

Table 5
Results of the statistical tests

Hyp.	Test statistic	*p*-Value
1	−0.65	0.2578
2	3.72	0.0001

Table 4 reveals that for small problems with up to 75 customers as well as for clustered problems the decomposition has no positive effect on solution quality. The reason for that is clear. For these problems the SbAS finds excellent solutions in very short time such that there is little space for improvements through solving smaller subproblems. Furthermore, for these problems 10 minutes are more than sufficient to solve the problems.

In fact, for the small random problems the effect seems to be reversed, i.e. the SbAS seems to outperform the D-Ants approach. For the random problems with 100 and more customers, the D-Ants clearly find better solutions than the original SbAS.

We verified these results through a statistical analysis where we have tested the following hypotheses using the Mann–Whitney U-test. Note that the Mann–Whitney U-test is the non-parametric counterpart of the standard t-test, while the Wilcoxon Signed Rank Test is the non-parametric version of the paired samples t-test.

Hypothesis 1. For problems with up to 75 customers, the SbAS outperforms the D-Ants.

$$H_0 : RPD_{\mathrm{SbAS}} = RPD_{\mathrm{D\text{-}Ants}},$$

$$H_a : RPD_{\mathrm{SbAS}} < RPD_{\mathrm{D\text{-}Ants}}.$$

To test this hypothesis we compared the RPD obtained with the D-Ants approach using $n_s = 2$ subproblems, with the RPD obtained with the SbAS for the 4 problem instances with 50 and 75 customers. As we have performed 10 runs for each instance this leads to a sample size of 40.

Hypothesis 2. For random problems with more than 75 customers, the D-Ants algorithm outperforms the SbAS, thus showing the contribution of problem decomposition.

$$H_0 : RPD_{\mathrm{D\text{-}Ants}} = RPD_{\mathrm{SbAS}}$$

$$H_a : RPD_{\mathrm{D\text{-}Ants}} < RPD_{\mathrm{SbAS}}$$

To test this hypothesis we compared the RPD obtained with the D-Ants approach using $n_s = 2$ subproblems, with the RPD obtained with the SbAS for the 6 random problem instances with 100, 150 and 199 customers. As we have performed 10 runs for each instance this leads to a sample size of 60.

Both Hypotheses were tested using one-sided tests and our conclusions will be drawn at a 99% level of confidence. Table 5 provides information about the test statistic and the p-value for the Mann–Whitney U-test.

The results in Table 5 show that the null hypothesis can only be rejected in the second case. While there are significant differences in the performances of the D-Ants and SbAS algorithms for large problems, the statistical analysis suggests that for the small problems there is no statistically

Table 6
Summary of computational comparison of various meta-heuristics

Algorithm	RPD	CPU time	Parallel	Computers
Osman [12]	1.03	33.60	*n*	VAX 8600
Taburoute [13]	0.88	46.8	*n*	Silicon Graphics 36 MHz
PTS [15]	0.55	24.65	*y*	4 Sun Sparc IPC
Flower [34]	1.54	2.32	*n*	HP 9000/712
GTS [8]	0.64	3.84	*n*	Pentium 200 MHz
UTSA [16]	0.69	13.75	*n*	Sun Ultrasparc 10 440 MHz
D-Ants (avg.)	0.48	3.80	*n*	Pentium 900 MHz
D-Ants (Best)	0.15	3.28	*n*	Pentium 900 MHz
Taillard [7]	0.05	n.a.	*y*	Silicon Graphics 100 MHz
Rochat and Taillard [14]	0.00	n.a.	*n*	Silicon Graphics 100 MHz
SbAS (overall)	0.05	n.a.	*n*	Pentium 900 MHz

significant difference between the performance of the D-Ants and the SbAS. In the latter cases there is neither a gain nor a loss through problem decomposition.

Finally, let us again take a look at Table 4. The bold entries show that for the problems with up to 75 customers the SbAS performs best, whereas for the larger (random) problems n_s varies between $n_s = 2$ and 3. Particularly, for the largest instances $n_s = 3$ is clearly favorable.

Overall, these results show that the D-Ants approach is superior to the original SbAS for larger problem instances. Moreover, the results suggest that subproblems with sizes between 50 and 75 customers seem to work best. Most important, the computational advantage that can be achieved through using the D-Ants algorithm increases as problem size increases.

Q2. *Do our D-Ants find solutions which are competitive to those provided by state-of-the-art TS approaches*?

While our results presented so far clearly show the contribution of problem decomposition, we have not yet evaluated the absolute performance of our D-Ants. This will be done now.

In Table 6, we provide information about the comparison of our D-Ants with state-of-the-art TS results. A comprehensive list of TS results is provided in [16]. In our comparison, we focus on the overall performance over the 14 benchmark problems.

In the first column, the names of the researchers (or the algorithms) are given. The column *RPD* refers to the relative percentage deviation from the best published results, CPU time is given in minutes, the column *Parallel* refers to the fact whether or not a parallel implementation was used and the last column provides some information about the computers used for the respective approaches. The sorting of the algorithms in Table 6 generally follows the publication date of the associated papers. Only the last three lines diverge from this logic, as these results lack information about the computational effort required to reach the solutions.

Besides the results of various TS implementations, Table 6 contains three lines with information about our Ant System. More precisely we present results about the RPD of both the average and the best solution quality obtained by our D-Ants with $n_s = 3$ over ten runs for each of the 14 problem instances (D-Ants (avg.), D-Ants (best)). In addition to that we present the best results we obtained during all our experiments with various parameter settings (SbAS (overall)).

The results show that a ‘fair’ comparison is difficult to establish. First, there is a significant tradeoff between solution quality and computational effort. Second, computational effort is hard to evaluate as the machines used differ greatly. Furthermore, runtime not only depends on the CPU of the machines but also on the operating system, the compiler, the programming language and the precision used during the execution of the run. Third, the number of runs needed to obtain the respective solutions are generally not reported (see also [10]) and it is unclear whether the reported results are best or average results.

Given these issues, we can make the following statements. It seems that our D-Ants with $n_s = 3$ outperforms all the TS approaches with respect to solution quality as both the best and the average RPD of our D-Ants are below those known for the TS approaches. Concerning computational effort required to obtain the presented results, it seems that our algorithm is at least competitive to the TS approaches. While most of the TS approaches were run on ‘slower’ machines the computation times our D-Ants consume are also generally significantly smaller. Those algorithms that require approximately the same runtime find quite poor solutions (e.g. Flower).

Comparing the best results obtainable overall, we see that our SbAS (with or without decomposition) finds solutions with an RPD of 0.05%. This RPD corresponds exactly to the deviation reported by Taillard (cf. [7]). Only the Adaptive Memory TS of Rochat and Taillard (cf. [14]) finds better solutions.

Summarizing, we believe that the results indicate the strengths of our D-Ants. In the next section we will turn to the large scale instances which are the true focus of our paper.

4.3.2. *Results based on large scale benchmark problem instances*

As we have found above, the size of the subproblems should range between 50 and 75 customers. Given these results we have modified the D-Ants algorithm in the following way. In each main loop of the algorithm the number of subproblems n_s is randomly drawn from the set of integer values $\lceil n/75 \rceil \leqslant n_s \leqslant \lfloor n/50 \rfloor$.

Q3. *Are our D-Ants a valid alternative to established approaches when applied to real world sized problem instances*?

To answer the last question we apply the D-Ants algorithm to the 20 large scale instances described in Section 4.1.2. For these problems, there are currently two state-of-the-art algorithms available, namely those of Prins (cf. [33]) and Toth and Vigo (cf. [8]).

The comparison with their results will be done in the following way. Clearly, the question relates to both solution quality and computational effort needed. While for the small instances described in the last section it seems to be of minor interest whether it takes 2 or 3 s to solve a problem with 50 customers, it certainly makes a difference if an algorithm takes 20 or 30 min to solve a problem with 400 customers and computation times are crucial.

However, besides this speed issue solution quality is important, as the problem instances are rather new. It can be expected that the best results known for these large scale instances can still be improved. So, we first set out to find solutions that are as good as possible. Thus, for each of the 20 large scale instances we perform 10 runs of 240 min.

The computational results are given in Table 7. Listed are our D-Ants, the GA of Prins and the Granular TS (GTS) approach of Toth and Vigo. While the results for the GA and for the GTS are best results given a fixed (the most favorable?) parameter constellation, we report both best and

Table 7
Summary of computational results for large scale problem instances

Algorithm	RPD	CPU time	Parallel	Computers
GTS [8]	2.1	17.54	*n*	Pentium 200 MHZ
Prins [33]	0.56	66.6	*n*	Pentium 1000 MHZ
D-Ants (avg.)	0.50	160.98	*n*	Pentium 900 MHZ
D-Ants (best)	0.01	144.53	*n*	Pentium 900 MHZ

average results over 10 runs for our Ant System. The computers used are a 200 MHz Pentium for the GTS, a 1 GHz Pentium for the GA and a 900 MHz Pentium for our Ant System. In the first column, the names of the researchers (or the algorithms) are given. The column *RPD* refers to the relative percentage deviation from the best published results, CPU time is given in minutes, the column *Parallel* refers to the fact whether or not a parallel implementation was used and the last column provides some information about the computers used for the respective approaches.

Table 7 clearly shows that concerning solution quality our D-Ants outperform the competing two algorithms both for the best run and on average. Moreover, our algorithm finds 9 new best solutions, whereas Prins has 2 best known solutions. Toth and Vigo have no best known solution with the chosen parameter setting. Note, that both Prins and Toth and Vigo have run their algorithms with different parameter constellations and through these overall tests have indeed found some more best known solutions as given in Table 3.

Generally, with respect to solution quality the algorithm of Toth and Vigo is not really competitive. However, it finds reasonable solutions so quick that it still seems to be a good alternative. This leads us to the issue of computation times. Obviously, for these large scale instances the tradeoff between solution quality and computational effort is significant. Our Ant System consumes more than twice the computation times reported by Prins, while we still take approximately 35 times (!) as much runtime as Toth and Vigo (given the differences in the machines used).

Another striking observation concerns the differences in the runtime for all approaches. While some instances can be solved very efficiently, others take very long computation times until reasonable solution qualities can be reported. Moreover, the three approaches show very different behavior on particular instances. On some instances our D-Ants find a new best solution within shorter time than the GA of Prins or the GTS of Toth and Vigo need to find poorer solutions. This effect is reversed for other instances.

Thus, a fair comparison of runtime is difficult. However, we will take the results obtained by our algorithm after computation times equivalent to those provided by the other authors. To do so, we have to take into account the differences in the machines used. Above we mentioned that computation times not only depend on the speed of the CPU but also on other factors such as programming language and compiler, operating system and precision used during the execution of the algorithm. However, following [33] we use a crude simplification and assume a linear relationship between the clock rate of the computer and the computational speed. This means that given our computer which is a 900 MHz Pentium we use a scaling factor of 4.5 for the comparison with the GTS and a scaling factor of 0.9 for the comparison with the GA. More specifically, we will for each instance divide the runtime reported by Toth and Vigo by 4.5 to compute the admissible runtime for our

Table 8
Runtime based comparison of three meta-heuristics

	GTS	GA
D-Ants	8.92(2.1)	1.06(0.56)

D-Ants for the comparison with the GTS, while we will divide the runtime reported by Prins by 0.9 to compute the admissible runtime for our D-Ants for the comparison with the GA. Note however, that this comparison is biased against our algorithm because of the runtime behavior discussed in the previous paragraph.

Table 8 shows the average RPD of the best solutions obtained by our D-Ants after a runtime equivalent to those consumed by the other two approaches. The numbers in brackets refer to the RPD of the approach we compare with.

Clearly the results show that concerning runtime our approach is outperformed by the other two algorithms. However, while the gap between the results of the D-Ants and the GTS of Toth and Vigo is huge, it is much closer when comparing the results of our D-Ants with the GA of Prins. These results suggest that especially in the beginning of a run our D-Ants perform poor as compared to the other approaches.

Overall, none of the three approaches dominates or is dominated by any other approach when assessed by the two goals solution quality and computational effort. However, the poor behavior of our algorithm in the very short run has led us to analyze our algorithm in order to find the main drivers of this computational effect.

Basically, we identified two key factors that lead to these results. The first one is the number of ants. In [29] it was found that for the application of the rank based AS to the VRP the number of ants should equal the number of customers. Looking at these large scale instances we realize that a population of up to 483 ants for the largest instance may not be reasonable. It takes a long time to generate 483 solutions and the pheromone update will consider only very few of those as the number of elitists is small. Thus, time is wasted especially in the master process.

The second factor is the size of the neighborhood. In [12] we found that the size of the neighborhood should be $n/4$, where n is the number of customers. This leads to a neighborhood size of up to 120 for the largest instances which causes a broad search in the initial phase of the algorithm. First, it takes a considerable amount of time to evaluate 120 alternatives in each decision and second the solution quality will generally be worse if that many alternatives are considered. Again this leads to large runtime to reach a solution of pre-specified quality.

We have analyzed the effects of changing these two factors separately on one of the largest problem instances. A detailed description of these tests can be found in Appendix B. The results show that some reduction in both the number of ants and the neighborhood size leads to improved solution quality and faster convergence. However, beyond a certain level the short run effects in solution evolution become small, while the long run behavior deteriorates. Further, the two effects are related and measures against them are to some extent substitutes. Taking these issues into account we performed our final runs with the number of ants fixed at $n/4$ and the upper bound on the size of the neighborhood set to $UB = 25$.

Table 9
Performance of three meta-heuristics on large scale problem instances

Instance	D-Ants			GA		GTS	
♯ (size)	Avg. RPD	Best RPD	Min	RPD	Min	RPD	Min
1 (240)	0.78	−0.04	62.52	0.03	32.42	1.59	4.98
2 (320)	0.34	0.01	57.67	0.14	77.92	1.24	8.28
3 (400)	0.74	0.00	21.92	0.00	120.83	3.32	12.94
4 (480)	0.96	0.55	119.12	0.77	187.60	9.44	15.13
5 (200)	0.00	0.00	0.87	0.00	1.04	3.66	2.38
6 (280)	0.00	0.00	5.72	0.00	9.97	6.54	4.65
7 (360)	0.14	0.00	14.03	0.71	39.05	3.45	11.66
8 (440)	0.33	0.00	35.30	0.31	88.3	1.75	11.08
9 (255)	0.52	0.26	21.52	1.71	14.32	1.10	11.67
10 (323)	0.83	0.62	17.48	0.65	36.58	0.68	15.83
11 (399)	1.00	0.57	96.88	1.34	78.5	0.95	33.12
12 (483)	0.93	0.62	61.38	1.68	30.87	1.18	42.9
13 (252)	0.58	0.17	87.20	1.46	15.3	0.43	11.43
14 (320)	1.07	0.95	25.85	0.34	34.07	0.92	14.51
15 (396)	0.29	0.22	23.80	0.99	110.48	0.83	18.45
16 (480)	0.25	**−0.14**	39.90	0.82	130.97	0.91	23.07
17 (240)	0.17	0.06	68.50	1.18	5.86	0.33	14.29
18 (300)	−0.41	**−0.82**	42.73	1.16	39.33	0.97	21.45
19 (360)	0.21	0.1	112.80	1.35	74.25	2.47	30.06
20 (420)	−0.33	**−0.65**	71.42	0.64	210.42	4.42	43.05
Averages	RPD	RPD	CPU	RPD	CPU	RPD	CPU
	0.42	0.12	49.33	0.76	66.6	2.31	17.54

Table 9 shows the results we obtained. Again we have run the algorithm 10 times on each instance. We now report the relative percentage deviation (RPD) from the best known solutions for our D-Ants (both for the best run and on average), the GA of Prins and the Granular TS of Toth and Vigo for each of the 20 instances. Note, that we already updated the best known solutions with the improved solutions we found in our initial results. We also report the computation times to reach the best solutions. The last row of the table shows the average RPD and runtime over all instances for the different approaches.

Apparently, our D-Ants again outperform the other approaches with respect to solution quality. However, now our algorithm outperforms the GA also with respect to computational effort required to obtain the presented solution quality. While the GA finds solutions which are on average 0.76% above the best known results within about 66.6 min, our D-Ants require 49.33 min to come up with solutions that are on average 0.12% above the best known solutions. Note further, that the real difference in runtime is even larger as Prins used a faster computer for his simulations.

Now that we know, that Prins' algorithm is outperformed by our D-Ants with respect to both dimensions, solution quality and runtime, let us again look at a runtime based comparison between our D-Ants and the GTS by Toth and Vigo. The final solutions obtained by our algorithm are much better than the ones reported for the GTS. However, the runtime for our approach is also much larger especially if we consider the differences in machines (the computers we used are approximately 4.5 times faster). So, we again look for a fair comparison by considering the results proposed by the D-Ants after a runtime equivalent to the ones reported for the GTS. As lined out above, this comparison is somewhat unfair against our approach due to different runtime requirements of the different approaches for the same instances. Disregarding this fact, we find that our D-Ants produce solutions which on average are 2.57% above the best known solutions. Compared to the 2.31% for the GTS, we see that the Granular TS still outperforms our approach for very short runtime. However, now the gap is much smaller than in our initial experiments, when we found solutions which on average were around 9% above the best known solutions.

These results show that the D-Ants can find reasonable solutions for the large scale problem instances within very short runtime. Allowing a liberal amount of computational time, the D-Ants come up with excellent solutions many of which improve upon the previously best known solutions. In fact, in our initial experiments we found 9 new best solutions and tied on three more instances. The results in Table 9 show that we found one more best known solution (for instance 1) and further improved three instances (instances 16, 18 and 20). Altogether this means that through our runs we found 10 new best solutions, we tied on 5 more instances, while we did not find the best known solution for the remaining instances. A more detailed list of the best found results can be found in Appendix C.

Finally, we have to consider the following issues. The SbAS which is embedded in our D-Ants, was shown to be very effective for clustered problems in [6] and above. The large scale benchmark problems are highly structured—more specifically customers are located on a grid. This situation can be very detrimental for the solution quality obtainable with the SbAS as this structure cannot be exploited as well as the clustered structure. In other words, the variance in the solutions found can be huge. While the D-Ants framework helps to partially overcome this difficulty, there could be clearly more powerful constructive techniques for problems on a grid. However, our experience shows that real world problems are rarely grid-problems, but are rather clustered ones. Given these comments, we believe that we have shown the potential of the D-Ants for solving real world sized problems.

5. Conclusions

In this paper we have presented an approach to enhance the efficiency of an Ant System for the VRP. The main idea is to decompose the problem into several disjoint subproblems based on a starting solution, each of which is then solved using an Ant System process. We have first shown that the chosen approach outperforms the original SbAS for large problems. In this context we have also analyzed the effects of the number of subproblems the main problem is decomposed into both on solution quality and runtime behavior. Using these results, we have then shown that the algorithm is competitive when compared with other state-of-the-art techniques for problems of real world size.

As pointed out above, the main advantage of our approach is that it can be applied to the VRP with Time Windows, the VRP with Backhauls or the VRP with Backhauls and Time Windows without modifications. It could also be applied to multi-depot VRPs without modifications. However, for these problems one could also consider a decomposition into several single-depot problems first.

Furthermore, the same idea can be used in dynamic, real-time vehicle routing problems to determine whether a new request can be accustomed or not, by optimizing only the subset of tours serving the area where the new request comes from. Finally, given the structure of the D-Ants approach it should be a promising candidate for a parallel implementation, which could further reduce runtime. These issues will be subject of further research.

Acknowledgements

The authors are grateful for financial support from the Austrian Science Foundation (FWF) under grant SFB #010 'Adaptive Information Systems and Modelling in Economics and Management Science' and from the Oesterreichische National bank (OENB) under grant #8630.

Appendix A. Runtime behavior of our D-Ants

In this appendix, we will look at the influence of problem decomposition on solution evolution in general, and at the influence of the number of subproblems n_s in particular.

More specifically, we will compare the SbAS without problem decomposition with the D-Ants algorithm with $n_s = 2, 3$ and 4 based on the 14 classic benchmark instances.

Table 10 summarizes our results concerning the average time needed to reach a solution with 5%, 2% and 1% deviation from the best known result.

Table 10
Effects of the number of subproblems n_s on the time (in seconds) needed to reach a pre-specified solution quality

				D-Ants								
Problem	SbAS			$n_s = 2$			$n_s = 3$			$n_s = 4$		
size	5%	2%	1%	5%	2%	1%	5%	2%	1%	5%	2%	1%
50	1	1	1	1	2	3	1	2	3	–	–	–
75	1	4	6	1	5	11	1	3	21	1	3	32
100	2	17	87	6	12	28	3	8	19	1	6	61
150	56	125	> 600	25	75	471	11	41	512	7	56	> 600
199	246	399	> 600	79	231	> 600	31	119	> 600	19	148	> 600
50	1	1	1	1	1	2	1	3	5			
75	1	8	69	2	9	> 600	1	9	119	1	24	160
100	12	20	23	7	14	20	5	9	18	3	11	21
150	109	158	> 600	33	125	220	24	86	566	17	131	354
199	437	> 600	> 600	104	498	> 600	41	196	> 600	48	189	> 600
120	1	1	21	1	1	12	1	1	6	1	1	4
100	1	1	1	1	1	1	1	1	1	1	1	1
120	1	51	117	1	21	75	1	9	93	1	11	169
100	1	1	1	1	1	6	1	1	1	1	1	1

Table 10 clearly shows that the original SbAS cannot compete with the D-Ants. While there is no difference in the runtime behavior for the small problems, on the larger instances the D-Ants outperform the SbAS that always solves the complete problem.

Obviously, the effect increases with problem size. Especially for the instances with 150 customers, the D-Ants produce results with less than 1% deviation from the best known solution within the time limit of 600 s, whereas the SbAS does not succeed to find that solution quality within 10 min.

Further analysis of the results in Table 10 shows that the 'best' number of subproblems increases with increased problem size. While this has been expected, the results reported highlight this effect nicely. Let us again look at the instances with 150 customers. The time needed to find solutions within 2% of the best known solution falls from between 125 and 158 s for the SbAS to between 75 and 125 s if two subproblems are used and further to between 41 and 86 s if $n_s = 3$. A further increase in n_s, while still decreasing the time needed to reach a solution within 5% of the best known solution, slows down the further progress in solution quality and for one instance even leads to the effect that within the time limit of 600 s the average solution does not reach the bound of 1% deviation from the best known solution. The same holds true for the largest instances. While increasing the number of subproblems over a certain level leads to a faster progress in the beginning of the algorithm—the 5% bound is reached earlier—it slows down the long run progress and the final solution quality starts to decline.

Appendix B. Effects of the number of ants and the neighborhood size on solution quality

Let us first look at the influence of the number of ants on solution quality and evolution. To that end we chose instance 4, which is one of the largest ones and ran our D-Ants 10 times for 4 h with different settings concerning the number of ants. More specifically, we considered $n, n/2, n/4, n/8$ and $n/16$ ants. The results are shown in Fig. 3.

Fig. 3 clearly shows that reducing the number of ants improves the performance of the algorithm with respect to runtime. After a very short time a significant gap between the solution qualities

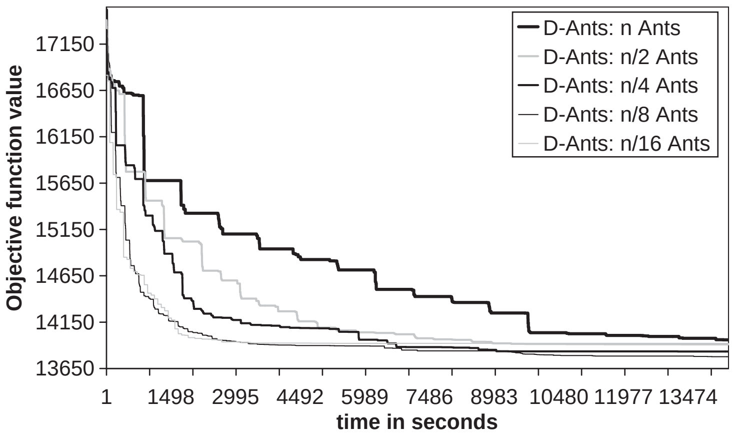

Fig. 3. Influence of the number of ants on the evolution of solution quality for large scale problem instance 4.

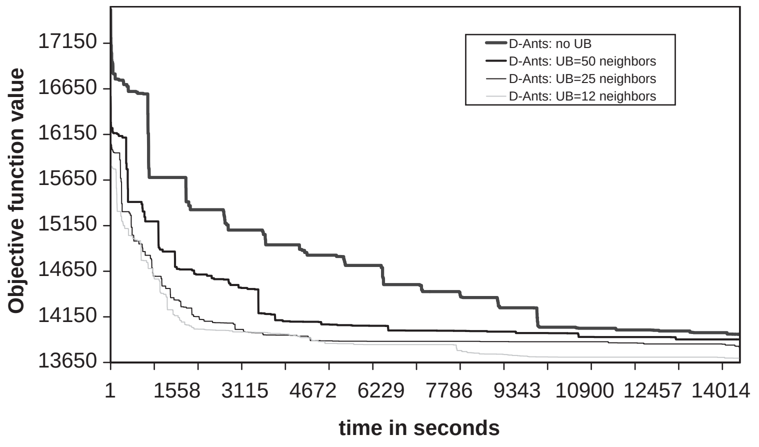

Fig. 4. Influence of the size of the neighborhood on the evolution of solution quality for large scale problem instance 4.

obtained with the different settings can be observed. This gap widens as the run progresses. Only after a very long time, it starts to narrow. However, even after 4 h the gap has not disappeared completely. While in the very long run it should eventually disappear the results suggest that for short to moderate runtime reducing the number of ants to $n/8$ ants is favorable. However a further reduction to $n/16$ ants leads to no further improvement. Moreover, we compared the results obtained with $n/8$ ants to the results reported by Toth and Vigo and Prins to again check the efficiency of the D-Ants. Toth and Vigo report a solution quality of 14,910.62 after 15.13 min. Our D-Ants with $n/8$ ants on average find a solution quality of 15,763.98 after an equivalent amount of time. On the other hand, Prins reports 13,728.80 after 187.60 min, where our D-Ants find on average 13,784.55 after an equivalent amount of time. These results suggest that in the very short run. the GTS of Toth and Vigo still shows the best performance. However, we cannot reduce the ants further, as we then reduce exploration even more and the ants will converge to poorer solutions as shown for the case with $n/16$ ants.

Thus, we have to look at the second effect described above namely the size of the neighborhood. As discussed above, the default setting for the neighborhood size is $n/4$. As the tests in [17] have shown, this setting is preferable to larger or smaller neighborhoods. The problem with this setting is, that as the instances get large these neighborhoods get large and thus the selection pressure in the roulette wheel selection decreases. To overcome this issue, we introduce an upper bound for the number of neighbors. While this leaves the algorithm unchanged for small problems, it should help for large problems.

More specifically, Fig. 4 reports the results for the D-Ants without an upper bound (i.e. the results already presented above), as well as for upper bounds (UB) of $\mathrm{UB} = 50, 25$ and 12. We performed 10 runs of 240 min on instance 4 with these four settings.

The results show the same qualitative features as the ones for different numbers of ants. Again, a reduction in the neighborhood size yields much faster evolution of the solution quality. However, for $\mathrm{UB} = 25$ and 12 there is no real difference early in the run, while in the end the average behavior for $\mathrm{UB} = 12$ neighbors is better than for $\mathrm{UB} = 25$. It seems that the smaller the neighborhood gets, the more influence comes from the master such that there will be improvements in both master problem

Table 11
Status of the best known solutions for the large scale benchmark problem instances

Instance	Previous BKS	D-Ants: best
1 (240)	5646.43	**5644.02**
2 (320)	8447.92	8449.12
3 (400)	11,036.22	11,036.22
4 (480)	13,624.52	13,699.11
5 (200)	6460.98	6460.98
6 (280)	8412.80	8412.90
7 (360)	10,195.59	10,195.59
8 (440)	11,828.78	11,828.78
9 (255)	587.09	**586.87**
10 (323)	746.56	750.77
11 (399)	932.68	**927.27**
12 (483)	1133.79	1140.87
13 (252)	868.80	**865.07**
14 (320)	1086.24	1093.77
15 (396)	1363.34	**1358.21**
16 (480)	1650.42	**1635.16**
17 (240)	709.09	**708.76**
18 (300)	1014.80	**998.83**
19 (360)	1376.49	**1367.20**
20 (420)	1846.55	**1822.94**

iterations and subproblem iterations leading to a better exploitation of the memory. However, as can be seen from the fact that there is little difference between UB = 25 and 12 in the beginning of the run, reducing the size of the neighborhood leads to reduced exploration and the algorithm has too little alternatives to work with. Thus, again a further reduction of the size of the neighborhood is detrimental to solution quality.

Finally, we also compared the results of the D-Ants with $n/8$ ants to those of the D-Ants with n ants and an upper bound on the size of the neighborhood of UB = 12. Between those results, there was no difference visible. Both solution evolution and long run solution quality are basically the same.

Given these results, it seems that a further reduction in both the number of ants and the upper bound on the size of the neighborhood is not advantageous. Alternatively, a combination of both measures could be reasonable. However, the disadvantage of this approach is that both measures lead to a reduction in the exploration of the search space. In fact, the choice of the number of ants and the size of the neighborhood reflects the tradeoff between solution quality in the short run and solution quality in the long run. The measures are to some extent substitutes as is shown by the fact that there is virtually no difference between reducing the number of ants to $n/8$ and fixing the upper bound on the size of the neighborhood to UB = 12. Taking these issues into account we performed our final runs with the number of ants fixed at $n/4$ and the upper bound on the size of the neighborhood set to UB = 25.

Appendix C. New best found solutions

In Table 11 we provide information about the previously best known results (Previous BKS) together with the best results (D-Ants: best) we obtained in our numerical experiments. Bold faced entries indicate that we found a new best solution.

References

[1] Christofides N. Vehicle routing. In: Lawler EL et al., editors. The traveling salesman problem. Chicester: Wiley, 1985.

[2] Toth P, Vigo D, editors. The vehicle routing problem. Philadelphia: Siam, 2002.

[3] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP completeness. New York: W.H. Freeman & Co, 1979.

[4] Clarke G, Wright JW. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 1964;12:568–81.

[5] Croes GA. A method for solving traveling salesman problems. Operations Research 1958;6:791–801.

[6] Reimann M, Stummer M, Doerner K. A savings based ant system for the vehicle routing problem. In: Langdon WB et al., editors. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002). San Francisco: Morgan Kaufmann, 2002.

[7] Taillard ED. Parallel iterative search methods for vehicle routing problems. Networks 1993;23:661–73.

[8] Toth P, Vigo D. The granular tabu search and its application to the vehicle routing problem. INFORMS Journal on Computing, 2002, in press.

[9] Cordeau JF, Gendreau M, Laporte G, Potvin JY, Semet F. A guide to vehicle routing heuristics. Journal of Operational Research Society 2002;53(5):512–22.

[10] Gillett B, Miller L. A heuristic algorithm for the vehicle dispatch problem. Operations Research 1974;22:340–9.

[11] Glover F, Laguna M. Tabu search. Boston: Kluwer, 1997.

[12] Osman IH. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research 1993;41:421–51.

[13] Gendreau M, Hertz A, Laporte GA. A tabu search heuristic for the vehicle routing problem. Management Science 1994;40:1276–90.

[14] Rochat Y, Taillard ED. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1995;1:147–67.

[15] Rego C, Roucairol C. A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In: Osman IH, Kelly J, editors. Meta-heuristics: theory and applications. Boston: Kluwer, 1996.

[16] Cordeau JF, Laporte G. Tabu search heuristics for the vehicle routing problem. GERAD Technical report G-2002-15, University of Montreal, Canada, 2002.

[17] Doerner KF, Gronalt M, Hartl RF, Reimann M, Strauss C, Stummer M. Savings Ants for the vehicle routing problem. In: Cagnoni S et al., editors. Applications of evolutionary computing. Berlin: Springer, 2002.

[18] Colorni A, Dorigo M, Maniezzo V. Distributed optimization by ant colonies. In: Varela F, Bourgine P, editors. Proceedings of the European Conference on Artificial Life. Amsterdam: Elsevier, 1991.

[19] Gutjahr WJ. A graph-based ant system and its convergence. Future Generation Computing Systems 2000;16:873–88.

[20] Gutjahr WJ. ACO algorithms with guaranteed convergence to the optimal solution. Information Processing Letters 2002;82:145–53.

[21] Stuetzle T, Dorigo M. A short convergence proof for a class of ACO algorithms. IEEE Transactions on Evolutionary Computation 2002;6(4):358–65.

[22] Bullnheimer B, Hartl RF, Strauss Ch. A new rank based version of the ant system: a computational study. Central European Journal of Operations Research 1999;7(1):25–38.

[23] Dorigo M, Gambardella LM. Any colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary computation 1997;1(1):53–66.

[24] Dorigo M, Di Caro G. The ant colony optimization meta-heuristic. In: Corne D et al., editors. New ideas in optimization. London: McGraw-hill, 1999.
[25] Stuetzle T, Hoos H. Improvements on the ant system: introducing the max–min ant system. In: Smith GD et al., editors. Proceedings of Artificial Neural Nets and Genetic Algorithms 1997. Wien: Springer, 1998.
[26] Stuetzle T, Hoos H. The max–min ant system and local search for combinatorial optimization problems. In: Voss S et al., editors. Meta-heuristics: advances and trends in local search paradigms for optimization. Boston: Kluwer, 1998.
[27] Bonabeau E, Dorigo M, Theraulaz G. Swarm intelligence. New York: Oxford University Press, 1999.
[28] Bullnheimer B, Hartl RF, Strauss Ch. Applying the ant system to the vehicle routing problem. In: Voss S et al., editors. Meta-heuristics: advances and trends in local search paradigms for optimization. Boston: Kluwer, 1999.
[29] Bullnheimer B, Hartl RF, Strauss Ch. An improved ant system algorithm for the vehicle routing problem. Annals of Operations Research 1999;89:319–28.
[30] Spaeth H. Cluster-Analyse-Algorithmen zur Objektklassifizierung und Datenreduktion. Muenchen: Oldenbourg, 1977.
[31] Christofides N, Mingozzi A, Toth P. The vehicle routing problem. In: Christofides N et al., editors. Combinatorial optimization. Chicester: Wiley, 1979.
[32] Golden BL, Wasil EA, Kelley JP, Chao KM. The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In: Crainic TG, Laporte G, editors. Fleet management and logistics. Norwell: Kluwer, 1998.
[33] Prins C. A simple and effective evolutionary algorithm for the vehicle routing problem. Research Report, University of Technology of Troyes, France, 2001.
[34] Rego C. A subpath ejection method for the vehicle routing problem. Management Science 1998;44:1447–59.