
UNIVERSITÉ LIBRE DE BRUXELLES
École Polytechnique de Bruxelles
IRIDIA - Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

# Coupling Ant Colony System with Local Search

**Luca Maria Gambardella**

Promoteur de Thèse:
Prof. **Marco DORIGO**

Thèse présentée en vue de l'obtention du titre de
Docteur en Sciences de l'Ingénieur

Année académique 2014-2015

# Summary

In the last decades there has been a lot of interest in computational models and algorithms inspired by the observation of natural systems. Nature seems to be very good at inventing processes that have many of the qualities we would like to be able to put into artificial systems. Consider for example genetic algorithms and evolutionary computation (Holland 1975): by reproducing in a simplified way the approach taken by Nature to develop living beings, researchers were able to develop both control systems for learning autonomous agents, and robust optimization algorithms for function optimization or combinatorial optimization. We can also think of work in reinforcement learning (Kaelbling et al. 1996): again, the observation of how animals learn to accomplish some simple tasks has suggested a useful and productive set of reinforcement-based learning algorithms. There are many such examples, some already very widespread, like simulated annealing (Kirkpatrick et al. 1983), genetic algorithms (Holland 1975), neural networks (Rumelhart et al. 1986), and others that still exist mostly in research labs, like immune networks (Bersini & F.Varela 1993) and classifier systems (Holland & Reitman 1978).

In this thesis we present another ethological approach to combinatorial optimization, which has been inspired by the observation of ant colonies: the Ant Colony System (ACS, Gambardella & Dorigo 1996; Dorigo & Gambardella 1997).

ACS finds its ground in Ant System (AS, Dorigo 1992; Colorni et al. 1992, 1995) and in Ant-Q algorithm (Gambardella & Dorigo 1995; Dorigo & Gambardella 1996), an extension of AS which integrates some ideas from Q-learning (Watkins 1989). ACS belongs to the class of Ant Colony Optimization (ACO) algorithms defined by Dorigo et al. (1999) as a general framework to describe the class of ant-inspired algorithms.

ACS (Chapter 2) was initially proposed to solve the symmetric and

asymmetric travelling salesman problems (TSP and ATSP), where it has been shown to be competitive with other metaheuristics (Gambardella & Dorigo 1996). Although this is an interesting and promising result, it was immediately clear that ACO, as well as other metaheuristics, in many cases cannot compete with specialized local search methods. An interesting trend (Mühlenbein et al. 1988; Ulder et al. 1991; Johnson & McGeoch 1997) is therefore to couple metaheuristics with a local optimizer, giving birth to so-called hybrid methods (Chapter 1). This is an interesting marriage since local optimizers often suffer from the initialization problem. That is, the performance of a local optimizer is often a function of the initial solution to which it is applied. Therefore, it becomes interesting to find good couplings between a metaheuristic and a local optimizer, where a coupling is good if the metaheuristic generates initial solutions that can be carried to very good local optima by the local optimizer.

In successive works we have shown that local search plays an important role in ACS. For example, Dorigo & Gambardella (1997) have applied ACS with a version of the 3-opt local search extended to symmetric and asymmetric TSPs, obtaining very good results (Chapter 2). Also, Gambardella et al. (1999) have proposed MACS-VRPTW (Multiple ACS for the Vehicle Routing Problem with Time Windows), a hierarchically organized ACS in which two different colonies successively optimize a multiple objective function, exploiting, among other things, local search. MACS-VRPTW (Chapter 3) has been shown to be competitive with other existing methods in terms of both solution quality and computational time and has been able to improve some of the best-known solutions for a number of problem instances in the literature. Starting from these experiences, the next proposed algorithm is HAS-SOP: Hybrid Ant System for the Sequential Ordering Problem (Gambardella & Dorigo 2000, Chapter 4). Sequential Ordering Problem (SOP). SOP is a version of the asymmetric travelling salesman problem where precedence constraints on vertices are imposed. In this work we define a new local search optimization algorithm called SOP-3-exchange for the SOP that extends a local search for the travelling salesman problem to handle multiple constraints directly without increasing computational complexity. The hybridization between ACS and SOP-3-exchange is investigated in depth (Chapter 4), and experimental evidence that the resulting algorithm is more effective than other methods is shown. In Gambardella

et al. (2012), directions for improving the original ACS framework when a strong local search routine is available have been identified. In particular, some modifications of the original ACS algorithm are presented. These modifications are able to speed up the method and to make it more competitive in case of large problem instances. The resulting framework, called Enhanced Ant Colony System (EACS, Gambardella et al. 2012, Chapter 5) is tested for the SOP. Many new best-known solutions are retrieved for the benchmarks available for these optimization problems. Lastly, Chapter 6 presents the application of ACO to solve academic and real-life vehicle routing problems (Rizzoli et al. 2007) where additional constraints and stochastic information are included.

This thesis is organized as follows:

Chapter 1 introduces combinatorial optimization problems and some heuristic techniques to solve them. In particular the chapter presents constructive and local search algorithms and their coupling with metaheuristics methods. This is the basis for ACS, which is described in Chapter 2. Also in Chapter 2, ACS-3-opt, the first coupling between ACS and local search, is presented. Chapter 3 describes MACS-VRPTW, the second application in which ACS is coupled with a local search, in particular to solve the vehicle routing problem with time windows. Chapter 4 presents HAS-SOP for the solution of sequential ordering problems. HAS-SOP also extends ACS with a specific local search designed explicitly for the SOP. Chapter 5 enhances ACS to explicitly deal with strong local searches, improving the original ACS algorithm with new effective features. Chapter 6 summarizes the work done in the application of ACS and local search to different types of vehicle routing problems including real-life industrial problems. Chapter 7 draws some conclusions and highlights possible extensions of the work.

# Acknowledgments

The research work reported in this thesis has been mainly done at IDSIA, Dalle Molle Institute for Artificial Intelligence in Manno, Lugano, Switzerland. IDSIA is affiliated to both the University of Applied Science and Art of Southern Switzerland (SUPSI) and the University of Lugano (USI). I offer my sincere thanks to these institutions, which have supported me during these years. I also thank the Swiss National Science Foundation, the Swiss Commission for Technology and Innovation, the European Commission, the spin-off company AntOptima sa, and many other companies and institutions which have funded my research during this period. At the personal level I would like to thank all my colleagues at IDSIA for the special environment they have created in these years, and all the people who have collaborated with me. Particular thanks go to all my co-authors of the research work presented in this thesis, Alberto Donati, Andrea Rizzoli, Davide Anghinolfi, Dennis Weyland, Derek H. Smith, Enzo Lucibello, Eric Taillard, Fabio De Luigi, Gianni Di Caro, Giovanni Agazzi, Marco Dorigo, Massimo Paolucci, Roberto Montemanni and Vittorio Maniezzo, without whom I would not have been able to succeed in my work. I am especially grateful to Andrea Rizzoli and Roberto Montemanni for their continuous support and to Marco Dorigo for his careful supervision along these years. Last but not least, special thoughts go to Daria, Matteo, and Marta (in order of appearance) for being always by my side.

# Original Contributions

This PhD thesis describes an original research carried out by the author. Nevertheless, some chapters of this document are based on papers prepared together with other co-authors, which have been published in the scientific literature. The rest of this section lists for each chapter the related scientific publications:

- Chapter 2: ACS: Ant Colony System
  - The first paper related to this chapter is Gambardella L.M. and Dorigo M. (1995), Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem, Twelfth International Conference on Machine Learning, A. Prieditis and S. Russell (Eds.), Morgan Kaufmann, pp. 252-260 (Gambardella & Dorigo 1995).
    Ant-Q has been developed since 1994 as an evolution of the Ant System. This article is the first conference paper to introduce this new algorithm and its main features. We present Ant-Q, a distributed algorithm which was inspired by both observation of ant colonies and work on reinforcement learning (RL) and in particular on Q-learning. We apply Ant-Q to both symmetric and asymmetric travelling salesman problems. The results show that Ant-Q has been able to find good solutions to these problems. In particular, it is particularly effective in finding solutions within a very small percentage of the optimum for difficult asymmetric travelling salesman problems.
  - The second paper related to this chapter is Gambardella L.M. and Dorigo M. (1996), Solving Symmetric and Asymmetric TSPs by Ant Colonies, ICEC96, Proceedings of the IEEE Conference on Evolutionary Computation, Nagoya, Japan, 20-22 May 1996 (Gambardella & Dorigo 1996).

This conference paper introduces the Ant Colony System (ACS), which was developed in 1995. The paper simplified Ant-Q by leaving out reinforcement learning concepts and presents the first results using ACS algorithms. ACS has been applied to both symmetric and asymmetric travelling salesman problems. The results show that ACS is able to find good solutions for these problems.

- The third paper related to this chapter is Dorigo M. and Gambardella L.M. (1997), Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, IEEE Transactions on Evolutionary Computation, 1 (1), 53-66 (Dorigo & Gambardella 1997).
  This journal paper describes in detail the Ant Colony System (ACS), a distributed algorithm that is applied to the travelling salesman problem (TSP). In ACS, a set of cooperating agents called ants cooperate to find good solutions to TSPs. Ants cooperate using an indirect form of communication mediated by pheromones that they deposit on the edges of the TSP graph while building solutions. We study ACS by running experiments to understand its operation. The results show that ACS has been able to outperform other nature-inspired algorithms such as simulated annealing and evolutionary computation, and we conclude by comparing ACS-3-opt, the first version of ACS coupled with a local search procedure, to some of the best-performing algorithms for symmetric and asymmetric TSPs. This paper is currently the second most cited paper ever published by IEEE Transactions on Evolutionary Computation.

- Chapter 3: MACS-VRPTW: A Multiple Ant Colony System for vehicle routing problems with time windows

  - The paper related to this chapter is Gambardella, L.M., Taillard, E.D., and Agazzi, G. (1999), MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows, in D. Corne, M. Dorigo, and F. Glover (Eds.), New Ideas in Optimization, McGraw Hill, London, UK, pp. 63-76 (Gambardella et al. 1999).
    This book chapter describes MACS-VRPTW. MACS-VRPTW

was developed and tested in 1997 as an extension of ACS to solve vehicle routing problems with time windows (VRPTW). MACS-VRPTW, which couples ACS with specialized local searches for the VRPTW, is organized with a hierarchy of artificial ant colonies designed to successively optimize a multiple objective function: the first colony minimizes the number of vehicles while the second colony minimizes the distances travelled. Cooperation between colonies is performed by exchanging information through pheromone updating. We show that MACS-VRPTW has been competitive with the best existing methods in terms of both solution quality and computation time.

- Chapter 4: HAS-SOP: An ant colony system hybridized with a new local search for the sequential ordering problem
  - The paper related to this chapter is Gambardella, L.M. and Dorigo, M. (2000). An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem, INFORMS Journal on Computing, 12 (3), 237-255 (Gambardella & Dorigo 2000).
    This journal paper presents HAS-SOP: Hybrid Ant Colony System for the Sequential Ordering Problem, another algorithm that couples ACS with local search. HAS-SOP was developed and tested in 1998. The paper also introduces a new sophisticated local search algorithm called SOP-3-exchange that is able to manage precedence constraints in constant time. In this paper we present the basic algorithms, the new local search we have introduced to explicitly manage the SOP problem, and experimental evidence the resulting algorithm has been more effective than existing methods for the problem.
- Chapter 5: EACS: Coupling ant colony systems with strong local searches
  - The paper related to this chapter is Gambardella, L.M., Montemanni, R., and Weyland, D. (2012), Coupling ant colony systems with strong local searches, European Journal of Operational Research, 220 (1), 831-843 (Gambardella et al. 2012).
    This journal paper presents EACS: Enhanced Ant Colony System. EACS was developed and tested in 2011. In particular,

in this paper the situation where a strong local search routine is available for an optimization problem is considered. It is shown how the original ACS framework can be enhanced to achieve a better integration between ACS and the local search. Experimental results for three optimization problems arising in transportation are discussed. The results show the effectiveness of the enhancements introduced. In this thesis only the application of EACS to SOP is presented.

- Chapter 6: Ant Colony Optimization for real-world vehicle routing problems: from theory to applications.
  - The paper related to this chapter is A.E. Rizzoli, R. Montemanni, E. Lucibello, and L.M. Gambardella (2007), Ant Colony Optimisation for Real World Vehicle Routing Problems: From Theory to Applications, Swarm Intelligence, 1 (2), 135-151, December (Rizzoli et al. 2007).
    In this journal paper we show the successful application of ACS to the Vehicle Routing Problem (VRP). First, we introduce VRP and its many variants, such as VRP with Time Windows, Time Dependent VRP, Dynamic VRP, and VRP with Pickup and Delivery. These variants have been formulated in order to bring the VRP as close as possible to the kinds of situations encountered in real-world distribution processes. In many of these real world applications ACS is successfully coupled with effective local search procedures.

## Algorithms

The following algorithms have been developed and tested by the author of this PhD thesis:

- Ant-Q: A Reinforcement Learning Approach to the Travelling Salesman Problem.
- ACS: Ant Colony System, including local searches for symmetric and asymmetric TSP problems.
- MACS-VRPTW: Multiple ACS for the Vehicle Routing Problem with Time Windows. This includes the hierarchical objective func-

tion optimization with two colonies and a sophisticated local search to manage vehicle routing problems with time windows.

- HAS-SOP: Hybrid Ant Colony System for the Sequential Ordering Problem (SOP). This includes an extension of ACS for SOP problems and SOP-3-Exchange, a new local search for the SOP.
- EACS: Enhanced Ant Colony System. This includes the basic EACS algorithm and the direct application to SOP problems.

# List of Algorithms

# Contents

# Chapter 1

# Metaheuristics and Local Search

## 1.1 Introduction

Combinatorial optimization problems can be found in a variety of situations like manufacturing scheduling, vehicle routing, distribution management, network routing, and crew rostering as well as in business and economics. Combinatorial optimization problems are interesting because their definition is usually simple but their solution is usually complex. The main difficulty in solving combinatorial optimization problems is that the optimal solution has to be chosen from the exponential set of all possible solutions. In fact, many of these problems are *NP-hard*; that is, it is not known whether they can be solved to optimality in polynomial computation time. To be more formal, according to Papadimitriou & Steiglitz (1982), an *optimization problem* is defined as the set of instances of an optimization problem. An instance of an optimization problem is a pair $(S, f)$ where $S$ is the domain of feasible solutions and $f$ is the cost function, a mapping $f : S \to \mathbb{R}$ that assigns a positive cost value to each of the solutions $s \in S$. The problem is to find a feasible solution of minimal cost value $s \in S$ for which $f(s) \leq f(y)\ \forall y \in S$. Such point $s$ is called a *globally optimal* solution to the given instance, or simply an *optimal* solution. We have been careful to distinguish between a *problem* and an *instance* of a problem. Informally, in an *instance* we are given the "input data" and have enough information to obtain a solution; a *problem* is a collection of instances, usually all generated in similar way.

Computational methods that are used to solve combinatorial optimization problems are usually grouped into exact algorithms and approximation algorithms (heuristics). In the case of an exact algorithm it is possible to ensure and to prove that the optimal solution is computed. On the other hand the computation of the optimal solution requires in the worst case the enumeration of all possible problem solutions. So, due to the combinatorial nature of the problem, already for instances of a few tens of nodes these methods become infeasible. Approximation methods aim to relax the constraint of optimality and to make a compromise between the quality of the solution and the required computational time.
These approximation methods are divided into three groups: construction techniques, local search techniques, and metaheuristics techniques (Johnson & McGeoch 1997).
Construction methods are the most simple but the least effective; these techniques are in general based on the construction of a feasible solution starting from scratch. This initial set is usually created empty or with one element often randomly chosen. After it is iteratively expanded by adding other elements according to a deterministic rule which considers only local conditions. The procedure ends when a feasible solution which contains all the elements is produced.
Local search methods are more sophisticated. Local search methods start from a complete feasible solution usually generated by a constructive procedure or by a random process. Thanks to a neighbourhood exploration, the procedure seeks to find ever better solutions by systematically exploring the neighbourhood of the current solutions. Given a solution $s$, the neighbourhood of $s, N(s)$ is the finite set of solutions $N : S \rightarrow 2^s$ that includes all solutions reachable from $s$ in one step. $N(s)$ is therefore the set of solutions to which $s$ could be immediately transformed in one move. The local search process searches for the best possible solution $next$ in $N(s)$. If $next$ improves $s$, the search is continued from $next$. The process stops when there are no more solutions that improve $s$ in the neighbourhood of the current solution $N(s)$. Each local search algorithm provides the rule to deterministically choose the best neighbourhood solution in the set of all possible neighbourhood solutions. The problem with these methods is that they are not able to escape from local minima. If the algorithm is no longer able to find an improved solution, the procedure stops.
One way to overcome this limitation is metaheuristic procedures. Meta-

heuristic procedures are also driven by a search process based on the notion of neighbourhood. However there are two main differences between local search and metaheuristics: (i) in contrast with local search procedures, metaheuristics methods typically use stochastic components in their searching process, and (ii) metaheuristics algorithms choose next solutions which do not necessary improve the current solution. These approaches have the advantage of being able to escape from local minima, and if they are well designed, they are able to find solutions that are often within a small percentage of the real optimal solution to the problem. Examples of metaheuristics include simulated annealing, tabu search, iterated local search, genetic algorithms, and Ant Colony Optimization (ACO). In many cases, however, these metaheuristic algorithms are slow and the time required to achieve good performance, mainly in the case of large instances, is prohibitive.
To deal with this limitation, one possibility (Mühlenbein et al. 1988; Ulder et al. 1991; Johnson & McGeoch 1997) is to couple metaheuristic methods with local search methods, giving birth to so-called *hybrid methods*. The main idea is to use metaheuristic algorithms to generate new feasible solutions and to improve these solutions during the search using a local search procedure. The use of hybrid metaheuristic algorithms currently allows combinatorial optimization problems to be solved efficiently in reasonable time.
The rest of this chapter is organized as follows: Section 1.2 introduces constructive algorithms starting from the travelling salesman problem (TSP), a classical combinatorial optimization problem. Section 1.3 is dedicated to local search procedures and Section 1.4 to metaheuristics. The last Section 1.5 introduces hybrid methods.

## 1.2 Constructive Methods

Constructive, local search, and metaheuristic algorithms are presented by their application to the TSP or to the more general asymmetric travelling salesman problem (ATSP). They are defined as follows.

**TSP** Let $V = \{v_1, \ldots, v_n\}$ be a set of cities (or nodes), $A = ((i,j) : i,j \in V)$ the edge set, and $d_{ij} = d_{ji}$ a cost measure associated with edge $(i,j) \in A$. The TSP is the problem of finding a closed tour of minimal

length that visits each city once (the minimal Hamiltonian cycle). When cities $v_i \in V$ are given by their coordinates $(x_i, y_i)$ and $d_{ij}$ is the Euclidean distance between $i$ and $j$ then we have a Euclidean TSP.

**ATSP** If $d_{ij} \neq d_{ji}$ for at least one $(i, j)$ then the TSP becomes an ATSP.

A constructive procedure usually starts by selecting an empty solution or by defining a partial solution which contains a single random chosen node. Starting from this set, the algorithm applies a constructive and usually deterministic rule to build a complete feasible solution step by step. To some extent a construction heuristic can be considered a specific class of local search algorithm. In this case the neighbourhood of the partial solution is given by the set of possible feasible nodes which could extend the current partial solution. These nodes are often ranked according to a greedy evaluation function.
The most popular constructive procedure for TSP is the Nearest Neighbour Heuristic ($NN$, Flood 1956; Reinelt 1994, Algorithm 1). $NN$ works as follows: the algorithm starts from a randomly chosen node, which becomes the last node in the sequence. The algorithm iteratively adds to the last node the closest feasible node, that is, the closest node in terms of distance that has not already been inserted in the sequence, until no more nodes are available. The standard procedure runs in time $O(n^2)$.

**Algorithm 1** Nearest Neighbour Heuristic for the TSP

**Procedure** *NearestNeighbour*
$W$ is the initial set of $n$ nodes
(1) Select an arbitrary node $j$, and set $l \leftarrow j$ and $W \leftarrow \{1, 2, ..., n\} \backslash \{j\}$
**while** $W \neq 0$ **do**
  Let $j \in W$ such that $d_{lj} = \min \{d_{li} | i \in W\}$
  Connect $l$ to $j$ and set $W \leftarrow W \backslash \{j\}$ and $l \leftarrow j$.
**end while**
Connect $l$ to the node selected in Step (1) to form a Hamiltonian cycle

Although solutions produced by $NN$ are usually not particularly good, they are often used as a starting solution for more effective meth-

ods like local search procedures (Reinelt 1994). Before discussing local search methods, we briefly mention that for the construction of feasible TSP solutions, possible alternative constructive methods are the insertion heuristics (Reinelt 1994), heuristics based on spanning trees (Prim 1957), and the savings methods (Clarke & Wright 1964).

To assess the average quality of the heuristic algorithm, a set of instances $I = (i_1, i_2, ..., i_k)$ is taken whose optimal (or lower bound) solution $opt_i\,(\forall i \in I)$ is known. The heuristic algorithm is then executed on each instance $i$ by running many experiments $exp_{ie}$. For each experiment $exp_{ie}$, the final objective function value $sol_{ie}$ is computed. The relative error for each experiment is given by $re_{ie} = ((sol_{ie} - opt_i)/opt_i)$ and the final average quality $avq$ of the heuristic algorithm is the average relative error over all the experiments: $avq = Avg(re_{ei}\,(\forall e,\, i)$.

In the case of constructive heuristics, a set of experiments has been executed by Bentley (1992). The paper reports that for $NN$ the average quality is 24.2% over the lower bound. In the same paper (Bentley 1992), in the case of the farthest insertion heuristic (i.e. the best insertion heuristic), the average quality is 13.0% over the lower bound, and in the case of the multiple fragment heuristic based on spanning trees, the average quality is 15.7% over the lower bound.
From these results we conclude that these constructive algorithms are not very effective. However they have the advantage of being very fast as the computation time to build a solution depends linearly to the number of cities. For this reason, these construction techniques are generally used as methods to generate initial solutions that are processed later on by local search techniques or by metaheuristics.

## 1.3 Local Search Methods

The idea of local search algorithms is to use procedures, which only locally change the structure of a given solution. Also, in the case of local searches, these procedures are based on the notion of neighbourhoods. The general, basic algorithm for a local search algorithm is presented in Algorithm 2. In the case of a minimization problem the local search procedure starts from a feasible solution $s$. In each iteration the neighbourhood $N(s)$ of $s$ is systematically explored, searching for the minimum

**Algorithm 2** Local Search Algorithm

```
Procedure Local_Search()
/* Initialize solution */
s ← an initial feasible solution.
while terminal_condition = false do
  generate N(s) : S → 2^s the set of feasible neighbourhood solutions of s
  s' ← the minimum solution in N(s)
  if f(s') < f(s) then
    s ← s'
  else
    terminal_condition = true
  end if
end while
return s
```

$s'$ such that $s' \in N(s)$. If $f(s') < f(s)$, the algorithm continues its search by setting $s = s'$. If $f(s') \geq f(s)$, the algorithm stops and $s$ is returned as the final solution. Solution $s$ is a local optimum for $f$, since $s$ is optimal with respect to the neighbourhood function but it is not guaranteed that $s$ is the global optimum solution for the objective function $f$.
There are many ways to define the neighbourhood structure $N(s)$. In the case of small $N(s)$, the exploration takes less time but the number of consecutive improving iterations is typically small. On the contrary, large neighbourhoods require more computational time leading to low quality solutions. In general there is a compromise between time and quality and the choice of the best neighbour structure often requires problem-specific knowledge.

Many efforts have been devoted to defining ad-hoc TSP local search procedures (see Johnson & McGeoch 1997 for an overview). These local search procedures are usually called *edge-exchange* heuristics. An *edge-exchange* heuristic starts from a given feasible solution and attempts to reduce its length by exchanging $k$ edges with another set of $k$ edges chosen according to some heuristic rule. This operation is usually called a *k-exchange* and is iteratively executed until no additional improving *k-exchange* using the heuristic rule is possible (in this situation we say that a local optimum has been found). When this is the case, the final solution is said to be *k-optimal*; the verification of *k-optimality* requires $O(n^k)$ time. It has been shown that increasing $k$ produces solutions of in-

creasing quality, but the computational effort required to test completely the *k-exchange* set for a given solution usually restricts our attention to *k-exchange* with $k \leq 3$. In fact, the most widely used edge-exchange procedures set $k$ to 2 or 3 (*2-opt* or *2-exchange*), (*3-opt* or *3-exchange*), (Lin 1965) or to a variable value (Lin & Kernighan 1973), in which case a variable-depth edge-exchange search is performed.

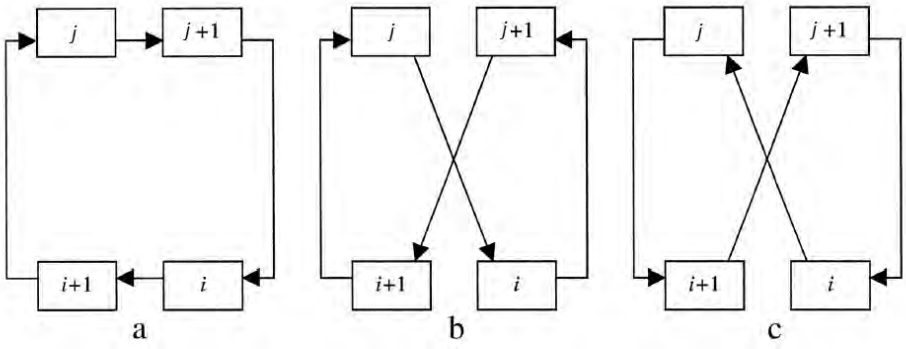


Figure 1.1: 2-exchange procecudere for TSP/ATSP

Consider a *2-exchange* procedure (Algorithm 3, Figure 1.1). This procedure receives as input a feasible solution *solution* and the number of cities $n$ of the original problem. The procedure explores all possible pairs of edges $((i, i+1), (j, j+1))$ starting from every possible couple of nodes $i$ and $j$ in *solution*. For each pair of edges $((i, i+1), (j, j+1))$ the procedure computes *gain*, that is, the numerical gain obtained removing from *solution* edges $(i, i+1)$ and $(j, j+1)$ and inserting in *solution* edges $(j, i)$ and $(j+1, i+1)$. The procedure selects the most profitable gain among all possible gains. If the gain is negative, that is, the solution is improved by the selected exchange, *solution* is modified accordingly and the process is iterated again on the new solution. The procedure stops when no negative improving gain is found so the last computed *solution* is returned. As in the case of *2-exchange*, any local search procedure is considered efficient only if the gain computation is executed in constant time.

In general, local search procedures produce better quality solutions than constructive algorithms (Reinelt 1994). On the other hand, a constructive procedure is needed to build a starting solution for local search algorithms. It is important to notice, anyway, that the general quality

**Algorithm 3** 2-exchange for TSP

```
Procedure 2_exchange_tsp(solution, n)
Initialization
while best_gain ≠ 0 do
  best_gain ← 0
  for i = 1 to n do
    for j = 1 to n do
      gain ← compute_gain(solution, i, j)
      if (gain < best_gain)) then
        best_gain ← gain, best_i ← i, best_j ← j
      end if
    end for
  end for
  if (best_gain < 0 ) then
    solution ← edge_exchange(solution, best_i, best_j)
  end if
end while
return solution
```

of local search procedures is not sufficient to provide solutions close to the optimal solution. In order to improve local search algorithms, an immediate option is to rerun the local search starting many times from different solutions (randomly generated in many cases). For this random-restart policy, it is necessary to keep the best solution among the many final solutions generated by multiple runs of the local search algorithm. By using this approach, the solution quality is better than the solution provided by a single run of local search. Anyway, also in this case each run is reaching a local minimum and the search process is inefficient. A way to overcome these problems is to use metaheuristic algorithms which provide an effective methodology to escape from local minima and to profit from the knowledge accumulated in previous iterations of the algorithm.

## 1.4 Metaheuristics

Metaheuristics have been proposed to solve combinatorial optimization problems when exact methods algorithms are inefficient. They overcome previous approaches which defined a specialized heuristic for each

problem. In fact, metaheuristics are rather general algorithmic frameworks which can be applied to several different optimization problems with few modifications. Examples of metaheuristics are simulated annealing (Kirkpatrick et al. 1983), genetic algorithms (Holland 1975), and tabu search (Glover 1989). Metaheuristics are often inspired by natural processes. In particular, the above-cited metaheuristics are inspired, respectively, by the physical annealing process, the Darwinian evolutionary process, and the clever management of memory structures. One of the most effective nature-inspired metaheuristics is Ant Colony Optimization (Dorigo et al. 1999), where the inspiring natural process is the foraging behaviour of ants.
The structure of the metaheuristic search has many principles in common with other search methods presented in previous paragraphs. As in the case of local search algorithms, metaheuristic algorithms (Algorithm 4) are also strongly based on the notion of neighbourhood. Also here the basic principle is to define a search method that starts from a solution or from a set of solutions (*current*). In contrast with local search, the majority of metaheuristic algorithms do not systematically explore the neighbours of the current solution looking for the best possible next improved solution. On the contrary, given the neighbourhood $N(current)$ of *current*, a candidate solution(s) (or a set of candidate solutions) *next* is stochastically selected and then evaluated. This evaluation process follows different criteria and is usually specific for each metaheuristic. In general, the evaluation includes the calculation of the value of the new solution(s) *next* (by using the objective function $f$) and the comparison of $f(next)$ with the current solution(s) $f(current)$. The result of this process is that *next* may be accepted or rejected. If the solution(s) *next* is accepted, the search continues by setting $current \leftarrow next$. We notice that in general *next* is always accepted if its value improves the value of the *current* solution(s). If *next* is rejected the search does not stop and the process is iterated by starting again from the same *current* solution(s). Following these search principles, metaheuristics do not stop their search when a local minimum is reached. The stopping criterion is usually provided by setting a given amount of search time or a given number of search iterations. These simple search mechanisms have proved to be very efficient as they are able both to escape from local minima and to produce solutions of very good quality.

In order to better understand the main principles and the most im-

**Algorithm 4** Metaheuristic Search

**Metaheuristic Search**
/* Inputs */
initial solution $s_{start}$ (or an initial set of solutions)
objective function $f$
neighbourhood function $N$
$current \leftarrow s_{start}$ (current should also be a set of solutions)
**while** terminal condition is not met **do**
  stochastically compute a solution (or a set of solutions) $next \in N(current)$
  Following a criterion, decide whether or not to continue the search from $next$ by setting $current \leftarrow next$
**end while**
**return** *current*

portant advantages of metaheuristics, in the next sections four metaheuristics, namely simulated annealing, tabu search, genetic algorithms, and ACO are presented.

### 1.4.1 Simulated Annealing

Simulated annealing (Kirkpatrick et al. 1983) is a metaheuristic technique inspired by the physical cooling of metals. The algorithm emulates the process where a solid is first melted and later slowly cooled in order to form large crystals at the minimum energy configuration. For this reason the method is based on a temperature parameter which is lowered during the search. Simulated annealing is largely used for finding solutions to continuous and discrete optimization problems in the case of a search space with many local minima. Simulated annealing, following the metaheuristic principle presented in Algorithm 4, it explores the neighbourood in a random order moving from solution to solution according to an acceptance criterion test. A new candidate solution is always accepted if its value improves on the value of the current solution. Otherwise a probabilistic acceptance test is executed based on both the temperature and the difference between the value of the current solution and the value of the next candidate solution. The larger the difference between the two values, the smaller the probability of accepting the new, worst, candidate solution. In addition, the acceptance probability is based on the

**Algorithm 5** Simulated Annealing

```
Simulated Annealing
/* Inputs */
neighbourhood function N
objective function f
/* Execution */
T ← determine a starting temperature
current ← generate an initial solution
best ← current
while not yet frozen do
  while not yet at equilibrium for this temperature do
    next ← a random solution selected from N(current)
    ΔE ← f(next) − f(current)
    if ΔE < 0 then
      current ← next
      if f(next) < f(best) then
        best ← next
      end if
    else
      choose a random number r uniformly from [0.1]
      if r < e^(−ΔE/T) then
        current ← next
      end if
    end if
  end while
  lower the temperature T
end while
return best
```

temperature. The temperature is decreased during the search process, starting from an initial value and moving down towards zero. The higher the temperature, the higher the probability of accepting a worse solution as the new solution. So at the beginning of the search, worse solutions have a larger probability of being accepted, while towards the end of the process this probability decreases and the method becomes almost a simple local search procedure. The original idea of this probabilistic test is taken from the Metropolis algorithm of Metropolis et al. (1953), which was defined to simulate the behaviour of atoms in the case of a heat bath.

In practice, the simulated annealing algorithms (Algorithm 5) work as follows: the algorithm receives as input a neighbourhood function $N$ and an objective function $f$. In the case of a TSP problem, the objective function $f$ is the length of the tour and the neighbourhood function $N$ is usually one step of the *2-opt* edge exchange procedure. The algorithm uses a starting temperature $T$ and an initial solution *current*. If *current* is computed by the algorithm, it is usually randomly generated. The algorithm is an iterative procedure composed of an inner and an outer loop. The outer loop executes the cooling process where the temperature is decreased step by step from the initial value towards zero and then the search is stopped. At each given temperature the inner loop simulates the thermal equilibrium process. Inside the inner loop, at each iteration a random solution *next* is selected from $N(current)$ and $\Delta E \leftarrow f(next) - f(current)$ is computed. If $f(next)$ is better than $f(current)$, *current* is replaced by *next* and the algorithm iterates. On the contrary, the probabilistic choice is activated by computing a random number $(0 < r < 1)$ and $e^{(-\Delta E/T)}$. If $r < e^{(-\Delta E/T)}$, *next* is accepted as the new *current* solution. In this formula higher temperature means higher probability of accepting the new solution. $T$ is high at the beginning of the algorithm and it usually decreased by using a cooling rate $cr < 1.0$ such that $T \leftarrow T * cr$. A good value for $cr$ is a number in the range $0.93 < cr < 0.99$. Following this iterative procedure, the algorithm generates and evaluates a set of candidate solutions. At the end of the process, the *best* solution computed from the beginning of the algorithm is returned.

Simulated annealing has been used to solve complex discrete and continuous optimization problems. The main advantage is that the method is simple, easy to implement, and robust since the final result does not

usually depend on the initial solution. For this reason, simulated annealing has been successfully applied to solve TSPs. In particular, Johnson & McGeoch (1997) report, for both random and structured problems, an average quality result in the range $3\% < avg < 4.5\%$. In general these are good results but, as discussed before, the required running time is usually too large. This is due to the fact that, in order to find good solutions, the method requires a high initial temperature and a slow cooling rate. For this reason, simulated annealing, as in general metaheuristic algorithms, is often coupled with local search algorithms in order to speed up the search and to produce high quality solutions (Section 1.5).

### 1.4.2 Tabu Search

A fundamental approach to escape from local optima is to use aspects of the search history. Tabu Search (TS) is a metaheuristic method which systematically utilizes memory for guiding the search process (Glover 1989). In the simplest and most widely applied version of TS, a greedy local search algorithm is enhanced with a form of short-term memory which enables it to escape from local optima. TS algorithms are among the most successful local search-based methods to date. As it happens for simulated annealing, also in the case of TS the algorithm moves from one solution to another. The next solution is always chosen from among the neighbouring solutions of the current solution, that is, solutions that are reachable from the current solution with a single move. It is precisely the notion of move and memory of these moves that guides the TS algorithm. The principle is to help the algorithm not to invert the moves made in the recent past in order to avoid forming cycles that do not allow an efficient exploration of the search space. For this reason, the algorithm uses the tabu-list, a fixed-length data structure that keeps the most recent moves in memory. Every time a new move is executed, it is inserted in the tabu-list and the oldest move is then removed from the list. If a new solution is selected, the tabu-list is investigated to decide whether to accept it or not.
As reported in Algorithm 6, TS, like many other metaheuristic algorithms, also begins its exploration with a randomly generated solution or a solution generated by a constructive heuristic. However, the main difference from other metaheuristic techniques is that TS is strongly based on a local search mechanism. In fact, at each iteration, the *next* solution is the best solution in the neighbourhood of the *current* solution

**Algorithm 6** Tabu Search

```
Tabu Search
current ← generate an initial solution
best ← current
while a terminal condition is met do
  ChooseBest: next ← the next best solution selected from N(current)
  if f(next) < f(best) then
    best ← next /* aspiration criteria */
    next while
  end if
  if the move that transforms current into next is forbidden in the tabu-list
  then
    goto ChooseBest
  else
    current ← next
    insert the inverted move in the tabu-list
    remove the last tabu move from the tabu-list
  end if
end while
```

and is not chosen stochastically. This behaviour is exactly the same as that of a local search algorithm of Section 1.3. The following step is the comparison between the value of *next* and the value of the best solution *best* computed so far. If $f(next) < f(best)$, *next* is always accepted as the new *current* solution regardless of any consideration about the past moves and the values in the tabu-list. This is called the aspiration criterion and it is one of the main components of the TS algorithm since it allows the memory constraints to be violated in the case of a successful solution. If the aspiration criterion is not verified, the algorithm proceeds with the verification of the validity of the move used to transform *current* into *next*. If the move is tabu (i.e. it belongs to the current tabu-list), the solution *next* is discarded. At this point, the algorithm selects as *next* the next best solution in the neighbourhood of *current* and the comparison with the tabu-list moves is repeated. When the *next* non-tabu solution is selected, it becomes the next *current* solution even if its value is worse than the value of the current *current* solution. This is a typical component of all metaheuristics which is also found in TS.

TS is an algorithm that does not use many parameters. In particular there are three components which must be specified to have a correct implementation of the algorithm. The first is the neighbourhood function. Still referring to the TSP problem, it can be said that a typical choice is to use 2-opt or 3-opt (Section 1.3). The second choice concerns the length of the tabu-list (tabu tenure). If the size of the tabu tenure is too small, we run the risk of entering into a cycle and not escaping from local minima. If the tabu tenure is too long, this prevents the efficient exploration of the search space. In the case of TSP, Glover (1989) reports that 6 is the ideal tabu tenure and a tabu tenure of less than 4 can be critical. More detailed studies of Tsubakitani & Evans (1998) tend to demonstrate that for the TSP the tabu tenure depends on the size $n$ of the problem and the selected neighbourhood function. For example Tsubakitani & Evans (1998) experimentally compute that the optimal choice for the tabu tenure is $n/4$ in the case of 2-opt and between $n/8$ and $n/16$ in the case of 3-opt. In the same study it is reported that a tabu tenure longer than $n/2$ drastically worsens the algorithm performance. In addition, a proposal has been made to dynamically vary the tabu tenure during the search or to adapt (increase or decrease) the tabu tenure according to search information and the presence of cycles (Reactive Tabu search, Battiti & Tecchiolli 1994a).
The last TS component is the tabu-list structure. In case of the TSP and the 2-opt neighbourhood function the basic tabu-list move is composed of pairs of edges (Glover 1989), each pair being the two edges removed during the 2-opt operation. Glover (1989) also suggests using as move the shortest of the two edges deleted during the 2-opt operation. Malek et al. (1989) propose to use the endpoints of the removed edges instead of the entire edge.
The first application of TS to the TSP is reported in Malek et al. (1989). In this paper it is experimentally shown that TS is superior to simulated annealing (Section 1.4.1) and that it is possible to take advantage of a parallel implementation.

### 1.4.3 Genetic Algorithms

A different approach to solve optimization problems is provided by genetic algorithms. Genetic algorithms (Holland 1975) are no longer based on the modification of a single solution but require the modification and combination of a population of solutions. Genetic algorithms are in-

**Algorithm 7** Genetic Algorithms

**Genetic Algorithms**
/* Inputs */
fitness function $fitness$ to be maximized and other parameters
/* Execution */
$t \leftarrow 1$, the number of the current generation
determine a random initial population $M(t)$ of $m$ individuals
determine the population fitness $fitness(t)$
**while** terminal condition is met **do**
1. define the selection probabilities $P(t)$
2. select parents for mating according to $P(t)$
3. perform crossover and mutation on the selected parents, creating new offspring $C(t)$
4. some individuals between $M(t)$ and $C(t)$ die and the new population $M(t+1)$ is defined
5. determine the population fitness $fitness(t+1)$
$t \leftarrow t+1$
**end while**
**return** the best solution from the beginning

spired by Darwin's theory of evolution including mechanisms such as inheritance, generations, natural selection, and reproduction. In genetic algorithms the selection mechanism determines which individuals are chosen for mating and how many children each selected couple will produce. Natural selection implies that individuals who are best suited to the environment are more likely to reproduce and to generate new offspring similar to themselves. Reproduction is based on the principle of survival of individuals with better fitness. Due to the recombination of genetic materials of the best parents, the evolution of the population is much faster and more efficient. From this point of view, individuals can be considered excellent problem solvers because they are able to survive in their environment and to develop skills and behaviours that are the result of natural evolution. Genetic algorithms implement this idea, starting from a set of solutions that correspond to a natural population. This population evolves generation after generation following the Darwinian mechanisms.

Genetic algorithms are described in Algorithm 7. The algorithm receives as input the fitness function $fitness$ to be maximized and other

parameters such as the population size $m$. The algorithm iterates step by step from generation to generation and in each generation it produces a new population. The number of the current generation is stored in the parameter $t$, which is initially set to 1. The first population of $m$ individuals is usually randomly created and for each individual the fitness function is computed. These individuals are solutions of the optimization problem and their structure is the counterpart of the genetic chromosome. An iterative process is then started with the terminal condition defined by setting the total running time or the total number of iterations. In each iteration the following steps are repeated:
1. For each individual the reproduction probability is computed. According to the Darwinian theory of evolution, the best individuals survive and therefore should have the highest probability of creating new offspring. In genetic algorithms each individual is associated with a selection probability proportional to the value of its fitness. The better the fitness, the higher the probability that the individual will be selected for reproduction.
2. Based on these probabilities, the selection process is started and a pair of parents is formed.
3. Each pair generates new offspring (usually a pair of individuals) through crossover and mutation mechanisms. Crossover is used to generate new offspring where one part of the chromosome is taken from one parent and the other part is taken from the second parent. This is very important because new individuals are not randomly generated but their structure (chromosome) depends critically on the structure of their parents. As in natural breeding, the creation of new offspring may suffer from some genetic mutation. Mutation means that the chromosomes of new individuals are perhaps a little different from the chromosomes of their parents due to errors in copying genes that happen in nature.
4. After the reproduction process, a set of new offspring is finally created. At this point the population for the next generation is calculated by selecting part of the individuals from the current population and part of the individuals from the new set of offspring. In general, the dimension $m$ of the population is constant and at least the best parent in the current generation is kept in the next generation (elitism). The proportion of elderly parents and new offspring in the new population is defined by external parameters, but it is not unusual for the new generation to be entirely composed of the new offspring.

5. The fitness of the new population is computed and the next iteration $t + 1$ is started.

Genetic algorithms have been successfully applied to the solution of TSPs (see for example Whitley et al. 1989). In general, to implement genetic algorithms for TSP, each individual is defined as a feasible TSP tour by providing the matrix representation or the cycle notation (Michalewicz 1994). Here, the main problem is the definition of effective crossover and mutation operations which prevent the generation of unfeasible solutions. In fact, several ad hoc crossover methods have been developed for the TSP. In particular, we can mention matrix crossover (MX, Goldberg 1989; Michalewicz 1994), which uses matrix representation, and cycle crossover (CX), which uses cycle notation (Goldberg 1989; Michalewicz 1994). Computation results of using the genetic algorithm to solve TSP problems are reported in Freisleben & Merz (1996a) with many solutions close to the best known results. Anyway, the algorithm is slow so it is not applicable for large instances in acceptable time. Therefore, as mentioned in the previous section on simulated annealing, also in the case of genetic algorithms the best way to proceed is to couple the basic algorithm with a local search procedure (Section 1.5).

### 1.4.4 ACO: Ant Colony Optimization Algorithms

ACO algorithms (Dorigo et al. 1999) pertain to the study of computational systems in which computation is carried out by simple artificial agents which mimic the behaviour of ants. Their basic principle is based on the way in which ants search for food and find their way back to the nest. Real ants are capable of finding the shortest path from a food source to the nest (Beckers et al. 1993; Goss et al. 1989) without using visual cues. Initially, ants explore the area surrounding their nest in a random manner. As soon as an ant finds a source of food it evaluates the quantity and quality of the food and carries some of this food to the nest. During the return trip, the ant leaves a chemical pheromone trail on the ground. The role of this pheromone trail is to guide other ants toward the source of food, and the quantity of pheromone left by an ant depends on the amount of food found. After a while, the path to the food source will be indicated by a strong pheromone trail and the more ants reach the source of food, the stronger the pheromone trail that is left. Ants exploit these pheromone trails as a means of finding their way from the nest to the food source and back. Also, they are capable of adapting to changes

**Algorithm 8** ACO, Ant Colony Optimization

```
ACO, Ant Colony Optimization
while termination conditions are not met do do
  schedule activities
    ants_generation_and_activity();
    pheromone_evaporation();
    daemon_actions(); {optional}
  end schedule activities
end while
```

in the environment, for example finding a new shortest path once the old one is no longer feasible due to a new obstacle (Beckers et al. 1993; Goss et al. 1989). In a sense, this behaviour is an emergent property of the ant colony. It is also interesting to note that ants can perform this specific behaviour using a simple form of indirect communication mediated by pheromone laying, known as stigmergy (Grassé 1959).
The transposition of real ants food searching behaviour into a framework for knowledge representation and problem solving is done in the following way: first the search space for ant-based algorithms is defined as a graph. Each edge of the graph has two associated measures: the cost and the artificial pheromone trail. Cost is a static value; that is, it never changes for a given problem, while an artificial pheromone trail is a dynamic value changed during runtime by ants. Second, real ants are replaced by artificial ants that move in the graph following and modifying the artificial pheromone trail. Each ant moves from node to node and chooses to move to the next node using a probabilistic rule which favours nodes that are close and connected by edges with a high pheromone trail value. The guiding principle is to increase the pheromone trail on those edges visited by those ants that have found a better solution. The pheromone trail also evaporates so that memory of the past is gradually lost.

Artificial ants have a twofold nature. On the one hand, they are an abstraction of those behavioural traits of real ants which seem to be at the heart of the shortest path-finding behaviour observed in real ant colonies. On the other hand, they have been enriched with some capabilities which do not find a natural counterpart. In fact, ant colony optimization is intended to be an engineering approach to the design and implementation of software systems for the solution of difficult optimization problems. It is therefore reasonable to give artificial ants some capabilities that,

although they do not correspond to any capacity of their real ant counterparts, make them more effective and efficient.
As presented in Dorigo et al. (1999) and in Algorithm 8, in ACO a finite-size colony of artificial ants with the above described characteristics collectively searches for good quality solutions to the optimization problem under consideration. Each ant builds a solution, or a component of it, starting from an initial state selected according to some problem-dependent criteria. While building its own solution, each ant collects information on the problem characteristics and on its own performance and uses this information to modify the representation of the problem, as seen by the other ants. Ants can act concurrently and independently, showing a cooperative behaviour. They do not use direct communication: it is the stigmergy paradigm that governs the information exchange among the ants.
An incremental constructive approach is used by the ants to search for a feasible solution. A solution is expressed as a minimum cost (shortest) path through the states of the problem in accordance with the problems constraints. Each single ant is able to find a (probably poor quality) solution. However, high quality solutions are found as the emergent result of the global cooperation among ants.
According to the assigned notion of the neighbourhood (problem-dependent), each ant builds a solution by moving through a (finite) sequence of neighbouring states. Moves are selected by applying a stochastic local search policy directed by (i) ants private information (the ants internal state or memory) and (ii) the publicly available pheromone trail and a priori problem-specific local information.
The ants internal state stores information about the ants past history. It can be used to carry useful information to compute the value/goodness of the generated solution and/or the contribution of each executed move. Moreover it can play a fundamental role in managing the feasibility of the solutions. In some problems, in fact, typically in combinatorial optimization, some of the moves available to an ant in a state can take the ant to an infeasible state. This can be avoided by exploiting the ants memory. Ants can therefore build feasible solutions using only knowledge about their local state.
This knowledge comprises both problem-specific heuristic information and knowledge accumulated in the pheromone trails. This pheromone knowledge is a shared local long-term memory that influences the ants

decisions. The decisions about when the ants should release pheromones into the environment and how much pheromone should be deposited depend on the characteristics of the problem and on the design of the implementation. Ants can release pheromone while building the solution (online step-by-step), or after a solution has been built, moving back to all the visited states (online delayed), or both. Autocatalysis plays an important role in ACO algorithms functioning: the more ants choose a move, the more the move is rewarded (by adding pheromone) and the more interesting it becomes to the next ants. In general, the amount of pheromone deposited is made proportional to the goodness of the solution an ant has built (or is building). In this way, if a move contributes to generating a high-quality solution, its goodness will be increased proportionally to its contribution. Once an ant has accomplished its task, consisting of building a solution and depositing pheromone information, the ant dies; that is, it is deleted from the system. The overall ACO meta-heuristic, beside the two above-described components acting from a local perspective (that is, *ants_generation_and_activity*() and *pheromone_evaporation*()), can also comprise some extra components which use global information and that go under the name of *daemon_actions*() in the algorithm reported in Algorithm 8. For example, a daemon can be allowed to observe the ants behaviour and collect useful global information to deposit additional pheromone information, biasing, in this way, the ant search process from a non-local perspective. Or, it could, on the basis of the observation of all the solutions generated by the ants, apply problem-specific local optimization procedures and deposit additional pheromone offline in addition to the pheromone the ants deposited online.

The ACO framework has been proposed in Dorigo et al. (1999), which initially includes not only the solution of combinatorial optimization problems but also the solution of routing in telecommunication networks (Di Caro & Dorigo 1998). Anyway, the first algorithm in this framework was Ant System (AS, Dorigo 1992; Colorni et al. 1991, 1992, 1995), which introduced the idea of ant-based optimization and the use of the pheromone trail. The second ACO algorithm was Ant-Q (Gambardella & Dorigo 1995; Dorigo & Gambardella 1996), which takes inspiration from reinforcement learning (Kaelbling et al. 1996). Ant-Q proposes a more efficient algorithm with a new methodology to build feasible solutions and to update the pheromone trail both locally and globally. Starting

from Ant-Q, the third algorithm was ACS of Gambardella & Dorigo (1996) (Chapter 2), which consolidated the ideas proposed in Ant-Q and introduced the first coupling between ACO and local search (Dorigo & Gambardella 1997).
Dorigo & Stützle (2004) distinguish between variants and extensions of AS. The variants differ from the original algorithm mainly in the pheromone update rule. Among variants we find: elitist ant system (Dorigo 1992); rank-based ant system (Bullnheimer et al. 1999a); and $\mathcal{MAX}$-$\mathcal{MIN}$ ant system (Stützle & Hoos 2000), which uses bounding techniques to limit the possible values for pheromone on arcs. Extensions display more substantial changes in the algorithm structure. Among the extensions we find: approximate nondeterministic tree search ANTS, (Maniezzo & Carbonaro 2000; Maniezzo et al. 2004) exploiting the use of lower bounds in the computation of a solution; D-ants (Reimann et al. 2002, 2004), which makes use of the savings algorithm; the hyper-cube framework for ACO (Blum & Dorigo 2004), which rescales pheromone values between 0 and 1; beam-ACO (Blum 2005), which hybridizes ACO with beam search and the mentioned Ant Colony System on which we focus in the remainder of this thesis.

## 1.5 Coupling Metaheuristics with Local Search

In the preceding sections, several techniques for solving combinatorial optimization problems have been shown. On the one hand the construction techniques (Section 1.2) are able to generate initial feasible solutions to the problem. Starting from these solutions (or other randomly generated solutions), it is possible to activate local search methods (Section 1.3). These algorithms are able to improve a feasible solution with a systematic exploration of the neighbourhood space. They are fast and efficient but their disadvantage is that they are not able to escape from local minima. Finally we saw metaheuristic algorithms (Section 1.4) often inspired by natural systems. These methods are capable of exploring large search spaces without becoming trapped in local minima as they use stochastic moves and memory and they accept solutions that are worse than the previous solutions. Their defect is, however, that they require long computational times and they do not go deep into the search space quickly. To solve these problems it was thought (Johnson & McGeoch

1997) to class of algorithms which combine the advantages of these two approaches. On the one hand, the local search methods are seen as a technique to intensify the search for good solutions on and the other metaheuristic methods are used as a tool to diversify the search into new areas of the search space. The coupling of these two methods is called hybridization (Blum et al. 2008).

One of the most interesting hybridization approaches for improving the efficiency of metaheuristic methods is Iterated Local Search (ILS, Lourenço et al. 2003, Algorithm 9). The proposal is to define a resolution algorithm that combines two components: one is a fast and deterministic method able to bring solutions to their local minimum quickly (intensification); the other is a stochastic metaheuristic method which can effectively explore the space of all possible solutions (diversification). The principle is to avoid repeating many calls to the local search procedure starting from many randomly generated solutions (see the random-restart policy in Section 1.3) and to embed successive calls of the local search in a metaheuristic-like method able to iteratively profit from previously generated solutions.
In Algorithm 9 the basic mechanism of ILS is presented. ILS starts by generating a solution $s_0$ that is immediately optimized with a local search procedure. Here, the iterative loop, which basically consists of three phases, begins. The first step is to perturb the current solution $s^*$, generating a new solution $s'$ (diversification). Local searches are usually deterministic and therefore the perturbation step must introduce a non-deterministic random component that allows the effective exploration of new areas of the search space. In this light, perturbation performs a kind of global random search in the solution space. The new solution $s'$ is immediately optimized with a call to the local search procedure which produces $s^{*\prime}$ (intensification). The next step is choosing whether to accept $s^{*\prime}$ as the new starting solution or to move back to the current solution $s^*$. Anyway, these two solutions are two local optimal solutions since both of them have been optimized by a local search procedure. The choice here may be greedy, in the sense that the better solution between $s^*$ and $s^{*\prime}$ is always used as a new starting point, or probabilistic, where the choice follows an approach similar to simulated annealing (Section 1.4.1, Martin et al. 1991). The algorithm then iterates, moving from one local minimum to the next.

In this class of algorithms the most delicate aspect to handle is the

**Algorithm 9** Iterated Local Search

**Iterated Local Search**
/* Inputs */
$s_0 \leftarrow$ generate an initial solution
$s^* \leftarrow Local\_search(s_0)$
**while** a terminal condition is met **do**
  $s' \leftarrow Perturbation(s^*, history)$  /* Diversification */
  $s^{*\prime} \leftarrow Local\_search(s')$  /* Intensification */
  $s^* \leftarrow Acceptance\_Criterion(s^*, s^{*'}, history)$
**end while**

balance between perturbation and local search. This balance has to be made on a case by case basis by analysing in depth the characteristics of these two components. The perturbation step should be strong enough to make it possible to escape from the local minima produced by the local search. On the contrary, if the perturbation is too strong, the process becomes very similar to the random-restart policy, since the algorithm does not exploit the knowledge accumulated in previous iterations of the algorithm.

One of the first applications of ILS to combinatorial optimization problems was the TSP (Martin et al. 1991). The authors proposed a method initially called Large-Step Markov Chains that combines a local search procedure like *3-opt* (Section 1.3) with a particular form of perturbation called *double-bridge* (Figure 1.2). *Double-bridge* perturbation randomly removes four edges from the tour $((h, h+1)(r, r+1)(s, s+1)(t, t+1))$ and builds a new tour, adding four new edges $((h, s+1)(t, r+1)(s, h+1)(r, t+1))$ according to Figure 1.2. The goal is to precisely create the desired balance between perturbation and local search. In fact, the combination of an intensification algorithm like *3-opt* with a perturbation algorithm like *2-opt* is not effective. If a *2-opt* perturbation is applied to $s^*$, obtaining $s'$, and then a *3-opt* local search is executed on $s'$, obtaining $s^{*\prime}$, the final local search solution $s^{*\prime}$ has a great chance of being exactly the original starting solution $s^*$, cancelling out the effect of the perturbation. The use of a *double-bridge* perturbation combined with *3-opt* was instead proven to be very efficient (Martin et al. 1991) because *3-opt* is not able to easily reverse the effect of the perturbation. The new optimized solution is usually a different local minimum from the original solution

**Algorithm 10** Coupling Metaheuristics with Local Search

**Coupling Metaheuristics with Local Search**
/* Inputs */
initial solution $s_{start}$ (or an initial set of solutions)
Apply *Local_search* to $s_{start}$
objective function $f$
neighbourhood function $N$
$current \leftarrow s_{start}$ (current should also be a set of solutions)
**while** terminal condition is met **do**
  stochastically compute a solution (or a set of solutions) $next \in N(current)$ /* Diversification */
  $next \leftarrow Local\_search(next)$ /* Intensification */
  Following a criterion, decide whether or not to continue the search from $next$ by setting $current \leftarrow next$
**end while**
**return** *current*

and the final algorithm is therefore very fast and very efficient. With this method Martin et al. (1991) proved that optimal solutions could be computed for many instances of the TSPLIB: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

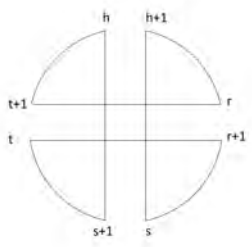


Figure 1.2: Double-bridge perturbation

The algorithm ILS is a clear example of successful coupling between metaheuristics and local search. One of the first proposals in this direction was made, however, by Brady (1985), who used genetic algorithms combined with *2-opt* as a local optimizer. As reported in the book of Johnson & McGeoch (1997), this is a new trend in solving combinatorial optimization problems. In particular, the idea is to see this coupling as

a new class of algorithm in which metaheristics and local search are fully integrated (Algorithm 10). In this perspective, the role of the metaheuristics component is to diversify the search and to allow the exploration of new areas of the search space, while the contribution of the local search is to intensify the exploration of these newly discovered areas. Following this schema, the need to balance the role of the two components is even more important, to avoid stagnation and premature convergence. In this direction, one of the most efficient couplings is therefore the combination of population-based metaheuristics, like genetic algorithms and ACO, with local searches. In fact, these methods are very efficient in the diversification phase and are very effective in generating and discovering promising new search areas.
This thesis shows a path in this direction by presenting the successful integration of ACO with different types of local search algorithms.
In particular we start by presenting *Ant Colony System* (ACS, Gambardella & Dorigo 1996, Chapter 2), the basic algorithm in our research, which is coupled with a specialized local search to solve TSPs (Dorigo & Gambardella 1997). ACS is then extended to deal with VRPs with time windows. The resulting *MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows* algorithm, (Gambardella et al. 1999, Chapter 3) couples a specialized local search for VRPs with an extension of ACS which includes two colonies working in parallel. ACS is then specialized to deal with the SOP, a special form of ATSP with precedence constraints. The resulting algorithm *HAS-SOP: An ant colony system hybridized with a new local search for the sequential ordering problem*, (Gambardella & Dorigo 2000, Chapter 4) includes a new and effective local search procedure *SOP-3-exchange*. The following effort is the definition of a new algorithm called *EACS: Coupling ant colony systems with strong local searches* (Gambardella et al. 2012, Chapter 5), where ACS is definitively extended to explicitly deal with a strong local search routine. For all these algorithms and for the application of ACS to industrial routing problems (Rizzoli et al. 2007, Chapter 6), computational results and comparisons with other algorithms are presented and investigated in the thesis.

# Chapter 2

# ACS: Ant Colony System

## 2.1 Introduction

This chapter introduces Ant Colony System (ACS, Gambardella & Dorigo 1996; Dorigo & Gambardella 1997), a distributed algorithm belonging to the ACO framework (Section 1.4.4), which has been applied to the traveling salesman problem (TSP). In ACS, a set of cooperating agents called ants cooperate to find good solutions to TSPs. Ants cooperate using an indirect form of communication mediated by pheromone they deposit on the edges of the TSP graph while building solutions. We have studied ACS by running experiments to understand its operation. The results have shown that ACS has been able to outperform other nature-inspired algorithms such as simulated annealing and evolutionary computation, and we conclude comparing ACS-3-opt, a version of ACS augmented with a local search procedure, to some effective algorithms for symmetric and asymmetric TSPs.

The natural metaphor on which ant algorithms are based is that of ant colonies. Real ants are capable of finding the shortest path from a food source to their nest (Beckers et al. 1992; Goss et al. 1989), without using visual cues (Hölldobler & Wilson 1990) by exploiting pheromone information. While walking, ants deposit pheromone on the ground, and follow, in probability, pheromone previously deposited by other ants. A way ants exploit pheromone to find a shortest path between two points is shown in Figure 2.1.

Consider Figure 2.1A: Ants arrive at a decision point in which they have to decide whether to turn left or right. Since they have no clue about

which is the best choice, they choose randomly. It can be expected that, on average, half of the ants decide to turn left and the other half to turn right. This happens both to ants moving from left to right (those whose name begins with an L) and to those moving from right to left (name begins with a R). Figure 2.1B and Figure 2.1C show what happens in the immediately following instants, supposing all ants walk at approximately the same speed. The number of dashed lines is roughly proportional to the amount of pheromone that the ants have deposited on the ground. Since the lower path is shorter than the upper one, more ants will visit it on average, and therefore pheromone accumulates faster. After a short transitory period the difference in the amount of pheromone on the two path is sufficiently large so as to influence the decision of new ants coming into the system (this is shown by Figure 2.1D). From now on, new ants will prefer in probability to choose the lower path, since at the decision point they perceive a greater amount of pheromone on the lower path. This in turn increases, with a positive feedback effect, the number of ants choosing the lower, and shorter, path. Very soon all ants will be using the shorter path.

The above behavior of real ants has inspired AS, the first algorithm in the ACO framework (Section 1.4.4), where a set of artificial ants cooperate to the solution of a problem by exchanging information via pheromone deposited on graph edges. AS has been applied to combinatorial optimization problems such as the traveling salesman problem (TSP) (Dorigo 1992; Colorni et al. 1991, 1992; Dorigo et al. 1996), and the quadratic assignment problem (Maniezzo et al. 1994).

ACS, the algorithm presented in this chapter, builds on the previous AS in the direction of improving efficiency when applied to symmetric and asymmetric TSPs. The main idea is that of having a set of agents, called ants, search in parallel for good solutions to the TSP, and cooperate through pheromone-mediated indirect and global communication. Informally, each ant constructs a TSP solution in an iterative way: it adds new cities to a partial solution by exploiting both information gained from past experience and a greedy heuristic. Memory takes the form of pheromone deposited by ants on TSP edges, while heuristic information is simply given by the edge's length.

The main novel idea introduced by ant algorithms, which will be discussed in the remainder of the chapter, is the synergistic use of cooperation among many relatively simple agents which communicate by

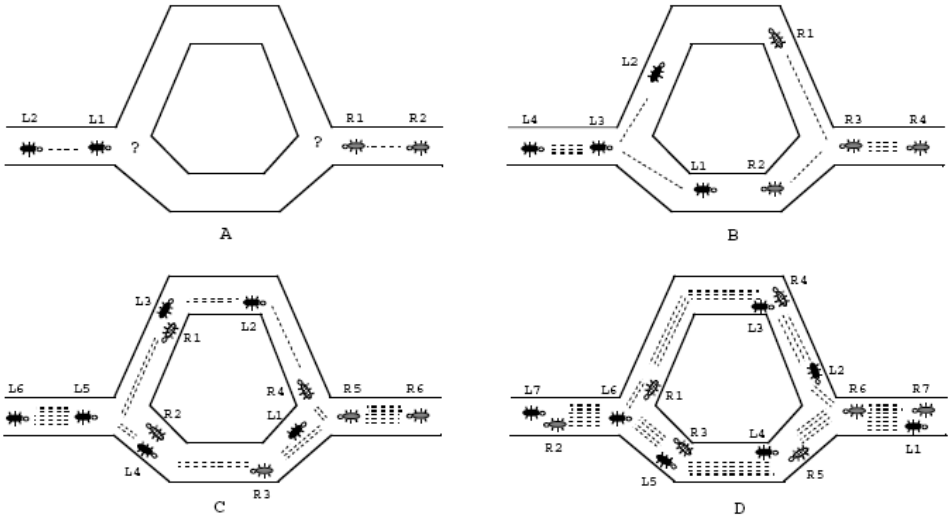


Figure 2.1: How real ants find a shortest path. A) Ants arrive at a decision point. B) Some ants choose the upper path and some the lower path. The choice is random. C) Since ants move at approximately constant speed, the ants which choose the lower, shorter, path reach the opposite decision point faster than those which choose the upper, longer, path. D) Pheromone accumulates at a higher rate on the shorter path. The number of dashed lines is approximately proportional to the amount of pheromone deposited by ants.

distributed memory implemented as pheromone deposited on edges of a graph.

The chapter is organized as follows. Section 2.2 puts ACS in context by describing AS, the progenitor of ACS. Section 2.3 introduces ACS. Section 2.4 is dedicated to the study of some characteristics of ACS: We study how pheromone changes at run time, estimate the optimal number of ants to be used, observe the effects of pheromone-mediated cooperation, and evaluate the role that pheromone and the greedy heuristic have in ACS performance. Section 2.6 provides an overview of results on a set of standard test problems and comparisons of ACS with well-known general purpose algorithms like evolutionary computation and simulated annealing. In Section 2.7 we add local optimization to ACS, obtaining an algorithm called ACS-3-opt. This algorithm is compared favorably with the winner of the First International Contest on Evolutionary Optimization (Bersini et al. 1996) on ATSP problems, while it yields a slightly worse performance on TSP problems. Section 2.8 is dedicated to the discussion of the main characteristics of ACS and presents some conclusions.

## 2.2 Background

AS (Dorigo 1992) is the progenitor of all our research efforts with ACO ant algorithms (Section 1.4.4), and was first applied to the traveling salesman problem (TSP), which is defined in Section 1.2.

AS utilizes a graph representation which is the same as that defined in case of TSP. In contrast with TSP graph, in AS each edge $(i, j)$ of the graph has two associated measures: the heuristic desirability $\mu_{ij}$ (e.g. the inverse of the edge length in case of TSP) and the pheromone trail $\tau_{ij}$. The heuristic desirability is fixed during the search while the pheromone trail is modified at runtime by ants. When AS is applied to symmetric instances of the TSP, $\tau_{ij} = \tau_{ji}$, while when it is applied to asymmetric instances it is possible that $\tau_{ij} \neq \tau_{ji}$.

Informally, AS works as follows. Ants are initially randomly distributed on cities. Each ant generates a complete tour by choosing the cities according to a probabilistic state transition rule: Ants prefer to move to cities which are connected by short edges with a high amount of pheromone. Once all ants have completed their tours a global pheromone updating rule (global updating rule, for short) is applied: A fraction of

the pheromone evaporates on all edges (edges that are not refreshed become less desirable), and then each ant deposits an amount of pheromone on edges which belong to its tour in proportion to how short its tour was (in other words, edges which belong to many short tours are the edges which receive the greater amount of pheromone). The process is then iterated.

The state transition rule used by AS, called a *random-proportional* rule, is given by Equation 2.1, which gives the probability with which ant $k$ in city $i$ chooses to move to the city $j$.

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]\cdot[\mu_{ij}]^\beta}{\sum_{u\in J_r^k}[\tau_{ij}]\cdot[\mu_{ij}]^\beta} & \text{if} \quad j \in J_r^k \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where $\tau$ is the pheromone, $\mu = \frac{1}{d}$ is the inverse of the distance $d_{ij}$, $J_r^k$ is the set of cities that remain to be visited by ant $k$ positioned on city $i$ (to make the solution feasible), and $\beta$ is a parameter which determines the relative importance of pheromone versus distance ($\beta > 0$). In Equation 2.1 we multiply the pheromone $\tau_{ij}$ on edge $(i, j)$ by the corresponding heuristic value $\mu_{ij}$. In this way we favor the choice of edges which are shorter and which have a greater amount of pheromone. In AS, the global updating rule is implemented as follows. Once all ants have built their tours, pheromone is updated on all edges according to

$$\tau_{ij} \leftarrow (1-\alpha)\cdot\tau_{ij} + \sum_{k=1}^{m} \triangle\tau_{ij}^k \tag{2.2}$$

where

$$\triangle\tau_{ij}^k = \begin{cases} \frac{1}{L_k} & \text{if } (i,j) \in \text{tour done by ant } k \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

$0 < \alpha < 1$ is a pheromone decay parameter, $L_k$ is the length of the tour performed by ant $k$, and $m$ is the number of ants.

Pheromone updating is intended to allocate a greater amount of pheromone to shorter tours. In a sense, this is similar to a reinforcement learning scheme (Sutton & Barto 1998; Kaelbling et al. 1996) in which better solutions get a higher reinforcement (as happens, for example, in genetic algorithms under proportional selection). The pheromone updating formula was meant to simulate the change in the amount of pheromone due to both the addition of new pheromone deposited by ants on the visited edges, and to pheromone evaporation.

**Algorithm 11** Ant Colony System (ACS). High Level Definition

```
Initialize
repeat {at this level each loop is called an iteration}
  Each ant is positioned on a starting node
  repeat {at this level each loop is called a step}
    Each ant applies a state transition rule to incrementally build a solution
    and a local pheromone updating rule
  until all ants have built a complete solution
  A global pheromone updating rule is applied
until End condition
```

Pheromone placed on the edges plays the role of a distributed long term memory: This memory is not stored locally within the individual ants, but is distributed on the edges of the graph. This allows an indirect form of communication called stigmergy (Grassé 1959; Deneubourg 1977). The interested reader will find a full description of AS, of its biological motivations, and computational results in Dorigo et al. (1996).

## 2.3 ACS: Ant Colony System

Although AS was useful for discovering good or optimal solutions for small TSPs (up to 30 cities), the time required to find such results made it unfeasible for larger problems. Three main changes were devised to improve its performance which led to the definition of ACS, presented in the this section.

ACS differs from the previous AS because of three main aspects:

- (*i*) the state transition rule provides a direct way to balance between exploration of new edges and exploitation of *a priori* and accumulated knowledge about the problem,
- (*ii*) the global updating rule is applied only to edges which belong to the best ant tour, and
- (*iii*) while ants construct a solution a *local pheromone updating rule* (local updating rule, for short) is applied.

Informally, ACS works as follows (Algorithm 11, 12): $m$ ants are initially positioned on $n$ cities chosen according to some initialization rule (e.g., randomly). Each ant builds a tour (i.e., a feasible solution

Table 2.1: A comparison of action choice rules. Type of delayed reinforcement: iteration-best. 50-city problems were stopped after 500 iterations. Oliver30 was stopped after 200 iterations and ry48p after 600 iterations. Averaged over 15 trials. Results in bold are the best in the Table.

| | *Pseudo-random* | | | *Pseudo-random-proportional* | | | *Random-proportional* | | |
|---|---|---|---|---|---|---|---|---|---|
| | average | std dev | best | average | std dev | best | average | std dev | best |
| City Set 1 | 6.18 | 0.06 | 6.03 | **5.87** | 0.05 | 5.84 | 7.85 | 0.25 | 7.40 |
| City Set 2 | 6.26 | 0.04 | 6.20 | **6.06** | 0.05 | 5.99 | 7.77 | 0.30 | 7.43 |
| City Set 3 | 5.69 | 0.07 | 5.61 | **5.57** | 0.00 | 5.57 | 7.89 | 0.17 | 7.75 |
| City Set 4 | 5.92 | 0.05 | 5.84 | **5.76** | 0.03 | 5.70 | 7.95 | 0.10 | 7.85 |
| City Set 5 | 6.30 | 0.04 | 6.22 | **6.18** | 0.01 | 6.17 | 8.48 | 0.21 | 8.10 |
| Oliver30 | 425.02 | 1.22 | 424.69 | **424.44** | 0.46 | 423.74 | 515.19 | 10 | 493.20 |
| ry48p | 15602 | 440 | 14848 | **14690** | 175 | 14422 | 19495 | 797 | 17921 |

to the TSP) by repeatedly applying a stochastic greedy rule (the state transition rule). While constructing its tour, an ant also modifies the amount of pheromone on the visited edges by applying the local updating rule. Once all ants have terminated their tour, the amount of pheromone on edges is modified again (by applying the global updating rule).

As was the case in AS, ants are guided, in building their tours, by both heuristic information (they prefer to choose short edges), and by pheromone information: An edge with a high amount of pheromone is a very desirable choice. The pheromone updating rules are designed so that they tend to give more pheromone to edges which should be visited by ants. The ACS high level algorithm is reported in Algorithm 11 and the detailed code in Algorithm 12.

In the following we discuss the state transition rule, the global updating rule, and the local updating rule.

### 2.3.1 ACS State Transition Rule

In ACS the state transition rule is as follows: an ant positioned on node $i$ chooses the city $j$ to move to by applying the rule given by Equation 2.4

$$s = \begin{cases} \arg\max_{u \in J_r^k} \{[\tau_{ij}] \cdot [\mu_{ij}]^\beta\} & \text{if } q \leq q_0 \quad \text{(exploitation)} \\ S & \text{otherwise} \quad \text{(biased exploration)} \end{cases} \tag{2.4}$$

where $q$ is a random number uniformly distributed in $[0 \,..\, 1]$, $q_0$ is a parameter ($0 \leq q_0 \leq 1$), and $s$ is a random variable selected according to the probability distribution given in Equation 2.1. The state transition rule resulting from Equation 2.4 and Equation 2.1 is called *pseudo-random-proportional rule*. This state transition rule, favors transitions towards nodes connected by short edges and with a large amount of pheromone. The parameter $q_0$ determines the relative importance of exploitation versus exploration: Every time an ant in city $i$ has to choose a city $j$ to move to, it samples a random number $0 \leq q \leq 1$. If $q \leq q_0$ then the best edge (according to Equations 2.4) is chosen (exploitation), otherwise an edge is chosen according to Equation 2.1 (biased exploration).

The *pseudo-random-proportional* action choice rule is the best compromise between the *pseudo-random* action choice rule (with the *pseudo-random* rule the chosen action is the best one with probability $q_0$, and a random one with probability $(1-q_0)$, and the *random-proportional* action choice rule of AS (Equation 2.1). Results obtained running experiments (Table 2.1) on a set of five randomly generated 50-city TSPs Durbin & Willshaw (1987), on the Oliver30 symmetric TSP (Whitley et al. 1989), and the ry48p asymmetric TSP (Reinelt 1991) have shown that the *pseudo-random-proportional* action choice is by far the best choice for the state transition rule (Gambardella & Dorigo 1995).

### 2.3.2 ACS Global Updating Rule

In ACS only the globally best ant (i.e., the ant which constructed the shortest tour from the beginning of the trial) is allowed to deposit pheromone. This choice, together with the use of the pseudo-random-proportional rule, is intended to make the search more directed: Ants search in a neighborhood of the best tour found up to the current iteration of the algorithm. Global updating is performed after all ants have completed their tours. The pheromone level is updated by applying the global updating rule of Equation 2.5

$$\tau_{ij} \leftarrow (1 - \alpha) \cdot \tau_{ij} + \alpha \cdot \triangle\tau_{ij} \tag{2.5}$$

where

$$\triangle\tau_{ij} = \begin{cases} \frac{1}{L_{gb}} & \text{if } (i,j) \in \text{global best tour} \\ 0 & \text{otherwise} \end{cases}$$

where $0 < \alpha < 1$ is the pheromone decay parameter, and $L_{gb}$ is the length of the globally best tour from the beginning of the trial. This global updating is intended to provide a greater amount of pheromone to the best (shorter) tour.

Equation 2.5 dictates that only those edges belonging to the globally best tour will receive reinforcement. We also tested in Gambardella & Dorigo (1996) another type of global updating rule, called *iteration-best*, as opposed to the above called *global-best*, which instead used $L_{ib}$ (the length of the best tour in the current iteration of the trial), in Equation 2.5. Also, with iteration-best the edges which receive reinforcement are those belonging to the best tour of the current iteration. Experiments have shown that the difference between the two schemes is minimal, with a slight preference for global-best, which is therefore used in the following experiments.

### 2.3.3 ACS Local Updating Rule

While building a TSP solution (i.e., a tour), ants visit edges and change their pheromone level by applying the local updating rule of Equation 2.6

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \rho \cdot \triangle\tau_{ij} \tag{2.6}$$

where $0 < \rho < 1$ is a parameter.

We have experimented with three values for the term $\triangle\tau_{ij}$. The first choice was loosely inspired by Q-learning (Watkins 1989), an algorithm developed to solve reinforcement learning problems (Kaelbling et al. 1996). Such problems are faced by an agent that must learn the best action to perform in each possible state, using as the sole learning information a scalar number which represents an evaluation of the state entered after it has performed the chosen action. Q-learning is an algorithm which allows an agent to learn such an optimal policy by the recursive application of a rule similar to that in Equation 2.6, in which the term $\triangle\tau_{ij}$ is set to the discounted evaluation of the next state value. Since the problem our ants have to solve is similar to a reinforcement

learning problem (ants have to learn which city to move to as a function of their current location), we set

$$\triangle\tau_{ij} = \gamma \cdot \max_{z \in J_j^k} \tau_{jz} \tag{2.7}$$

which is exactly the same formula used in Q-learning where ($0 \leq \gamma \leq 1$) is a parameter). The other two choices were: we set $\triangle\tau_{ij} = 0$ and we set

$$\triangle\tau_{ij} = \tau_0 \tag{2.8}$$

where $\tau_0$ is the initial pheromone level that it is set to

$$\tau_0 = (n \cdot L_{nn}^{-1}) \tag{2.9}$$

where $n$ is the number of cities of the problem and $L_{nn}$ is the length of tour constructed with a nearest neighbour heuristic (Algorithm 1).
Since $\tau_0$ is usually very small in relation to the length of a tour an important effect in using Formula 2.8 and Formula 2.9 is that $\tau_{ij}$ never decreases below $\tau_0$ so that the values of the pheromone of each edge is bounded. In addition in case the edge was never updated with the global updating rule its pheromone level remains unchanged.

Finally, we also ran experiments in which local-updating was not applied (i.e., the local updating rule is not used, as was the case in AS).
Results obtained running experiments (Table 2.2) on a set of five randomly generated 50-city TSPs Durbin & Willshaw (1987), on the Oliver30 symmetric TSP (Whitley et al. 1989), and the ry48p asymmetric TSP (Reinelt 1991) essentially suggest that local-updating is definitely useful, and that the local updating rule with $\triangle\tau_{ij} = 0$ yields worse performance than local-updating with $\triangle\tau_{ij} = \tau_0$ or with Formula 2.7.

ACS with Formula 2.7 was called Ant-Q in Gambardella & Dorigo (1995) while ACS with $\triangle\tau_{ij} = \tau_0$ (Formula 2.10) was simply called ACS hereafter (Gambardella & Dorigo 1996; Dorigo & Gambardella 1997). ACS and Ant-Q resulted to be the two best performing algorithms, with a similar performance level. Since the ACS local updating rule requires less computation than Ant-Q, we chose to focus attention on ACS.

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0 \tag{2.10}$$

Table 2.2: A comparison of local updating rules. 50-city problems and Oliver30 were stopped after 2,500 iterations, while ry48p was halted after 10,000 iterations. Averages are over 25 trials. Results in bold are the best in the Table.

| | ACS | | | ANT-Q | | | ACS with $\triangle\tau_{ij} = 0$ | | | ACS without local updating | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | average | std dev | best | average | std dev | best | average | std dev | best | average | std dev | best |
| City Set 1 | **5.88** | 0.05 | **5.84** | **5.88** | 0.05 | **5.84** | 5.97 | 0.09 | 5.85 | 5.96 | 0.09 | **5.84** |
| City Set 2 | **6.05** | 0.03 | **5.99** | 6.07 | 0.07 | **5.99** | 6.13 | 0.08 | 6.05 | 6.15 | 0.09 | 6.05 |
| City Set 3 | **5.58** | 0.01 | **5.57** | 5.59 | 0.05 | **5.57** | 5.72 | 0.12 | **5.57** | 5.68 | 0.14 | **5.57** |
| City Set 4 | **5.74** | 0.03 | **5.70** | **5.75** | 0.04 | **5.70** | 5.83 | 0.12 | **5.70** | 5.79 | 0.05 | 5.71 |
| City Set 5 | **6.18** | 0.01 | **6.17** | **6.18** | 0.01 | **6.17** | 6.29 | 0.11 | **6.17** | 6.27 | 0.09 | **6.17** |
| Oliver30 | 424.74 | 2.83 | **423.74** | **424.70** | 2.00 | **423.74** | 427.52 | 5.21 | **423.74** | 427.31 | 3.63 | 423.91 |
| ry48p | **14,625** | 142 | **14,422** | 14,766 | 240 | **14,422** | 15,196 | 233 | 14,734 | 15,308 | 241 | 14,796 |

As will be discussed in Section 2.4, the role of ACS local updating rule is to shuffle the tours, so that the early cities in one ants tour may be explored later in other ants tours. In other words, the effect of local-updating is to make the desirability of edges change dynamically: every time an ant uses an edge this becomes slightly less desirable (since it loses some of its pheromone). In this way ants will make a better use of pheromone information: without local-updating all ants would search in a narrow neighborhood of the best previous tour.

### 2.3.4 ACS Parameter Settings

In all experiments of the following sections the numeric parameters, except when indicated differently, are set to the following values: $\beta = 2$, $q_0 = 0.9$, $\alpha = \rho = 0.1$, $\tau_0 = (n \cdot L_{nn}^{-1})$.

These values were obtained by a preliminary optimization phase, in which we found that the experimental optimal values of the parameters was largely independent of the problem, except for $\tau_0$ for which, as we said, $\tau_0 = (n \cdot L_{nn}^{-1})$. The number of ants used is $m = 10$ (this choice is explained in Section 2.4). Regarding their initial positioning, ants are placed randomly, with at most one ant in each city.

## 2.4 A Study of Some Characteristics of ACS

### 2.4.1 Pheromone Behavior and its Relation to Performance

To try to understand which mechanism ACS uses to direct the search we study how the pheromone-closeness product $[\tau_{ij}] \cdot [\mu_{ij}]^{\beta}$ changes at run

**Algorithm 12** Ant Colony System (ACS). Pseudo-code

**Procedure** ACS: Ant Colony System
**1.** /* Initialization phase */
**for each** pair $(i,j)$ **do** $\tau_{ij} \leftarrow \tau_0$
**for** $k := 1$ to $m$ **do**
  let $i_1^k$ be the starting city for ant $k$
  $J_{i_1^k}^k \leftarrow \{1,...,n\} - i_1^k$
    /* $J_{i_1^k}^k$ is the set of yet to be visited cities for ant $k$ in city $i_1^k$ */
  $i^k \leftarrow i_1^k$ /* $i^k$ is the city where ant $k$ is located */
**end for**
**2.** /* This is the phase in which ants build their tours. The tour of ant $k$ is stored in $Tour^k$. */
**for** $c := 1$ to $n$ **do**
  **if** $c < n$ **then**
    **for** $k := 1$ to $m$ **do**
      Choose the next city $j^k$ according to Equations 2.4 and 2.1
      $J_{j^k}^k \leftarrow J_{i^k}^k - j^k$
      $Tour^k(c) \leftarrow (i^k, j^k)$
    **end for**
  **else**
    **for** $k := 1$ to $m$ **do**
      /* In this cycle all the ants go back to the initial city $i_1^k$*/
      $j^k \leftarrow i_1^k$
      $Tour^k(c) \leftarrow (i^k, j^k)$
    **end for**
  **end if**
  /* In this phase local updating occurs and pheromone is updated using Equation 2.10*/
  **for** $k := 1$ to $m$ **do**
    $\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \rho \cdot \tau_0$
  **end for**
  $i^k \leftarrow j^k$ /* New city for ant $k$ */
**end for**
**3.** /* In this phase global updating occurs and pheromone is updated */
**for** $k := 1$ to $m$ **do**
  compute $L_k$ /* $L_k$ is the length of the tour done by ant $k$ */
**end for**
Compute $L_{gb}$, the best solution among all $L_k$ using a *global-best* rule
/* Update edges belonging to $L_{gb}$ using Equation 2.5 */
**for each** pair $(i,j) \in L_{gb}$ **do**
  $\tau_{ij} \leftarrow (1-\alpha) \cdot \tau_{ij} + \alpha \cdot \frac{1}{L_{gb}}$
**end for each**
**4. if** $(End\_condition = True)$ **then**
  Print shortest of $L_k$
**else goto** Phase 2

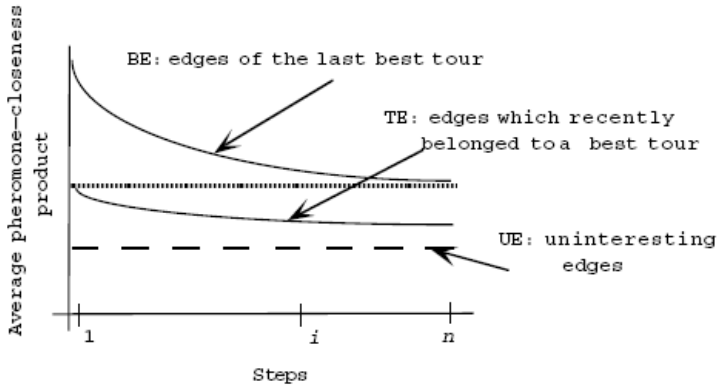


Figure 2.2: Families of edges classified according to different behavior with respect to the pheromone-closeness product. The average level of the pheromone-closeness product changes in each family during one iteration of the algorithm (i.e., during n steps)

time. Figure 2.2 shows how the pheromone-closeness product changes with the number of steps while ants are building a solution (steps refer to the inner loop in Algorithm 11: the abscissa goes therefore from 1 to $n$, where $n$ is the number of cities).

Let us consider three families of edges (Figure 2.2): (i) those belonging to the last best tour (BE, Best Edges), (ii) those which do not belong to the last best tour, but which did in one of the two preceding iterations (TE, Testable Edges), (iii) the remaining edges, that is, those that have never belonged to a best tour or have not in the last two iterations (UE, Uninteresting Edges). The average pheromone-closeness product is then computed as the average of pheromone-closeness values of all the edges within a family. The graph clearly shows that ACS favors exploitation of edges in BE (BE edges are chosen with probability $q_0 = 0.9$) and exploration of edges in TE (recall that, since Equations 2.4 and 2.1, edges with higher pheromone-closeness product have a higher probability of being explored).

An interesting aspect is that while edges are visited by ants, the application of the local updating rule, (Equation 2.10), makes their pheromone diminish, making them less and less attractive, and therefore favoring the exploration of edges not yet visited. Local updating has the effect of lowering the pheromone on visited edges so that these become less desirable and therefore will be chosen with a lower probability by the other ants in

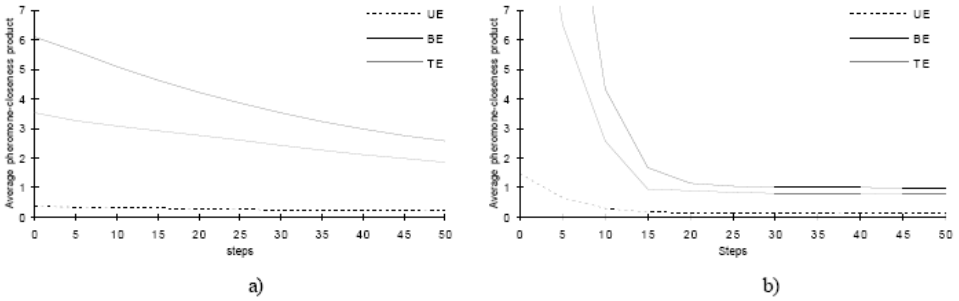


Figure 2.3: Families of edges classified according to different behavior with respect to the level of the pheromone-closeness product. Problem: Eil51 [14]. a) pheromone-closeness behavior when the system performance is good. Best solution found after 1,000 iterations: 426, $\alpha = \rho = 0.1$. b) pheromone-closeness behavior when the system performance is bad. Best solution found after 1,000 iterations: 465, $\alpha = \rho = 0.9$.

the remaining steps of an iteration of the algorithm. As a consequence, ants never converge to a common path. This fact, which was observed experimentally, is a desirable property given that if ants explore different paths then there is a higher probability that one of them will find an improving solution than there is in the case that they all converge to the same tour (which would make the use of $m$ ants pointless).

Experimental observation has shown that edges in BE, when ACS achieves a good performance, will be approximately downgraded to TE after an iteration of the algorithm (i.e., one external loop in Algorithm 12; see also Figure 2.2), and that edges in TE will soon be downgraded to UE, unless they happen to belong to a new shortest tour.

In Figure 2.3a and Figure 2.3b we report two typical behaviors of pheromone level when the system has a good or a bad performance respectively.

### 2.4.2 The Optimal Number of Ants

Consider Figure 2.4. Let $\varphi_2\tau_0$ be the average pheromone level on edges in BE just after they are updated by the global updating rule, and $\varphi_1\tau_0$ the average pheromone level on edges in BE just before they are updated by the global updating rule $\varphi_1\tau_0$ is also approximately the average pheromone level on edges in TE at the beginning of the inner loop of the

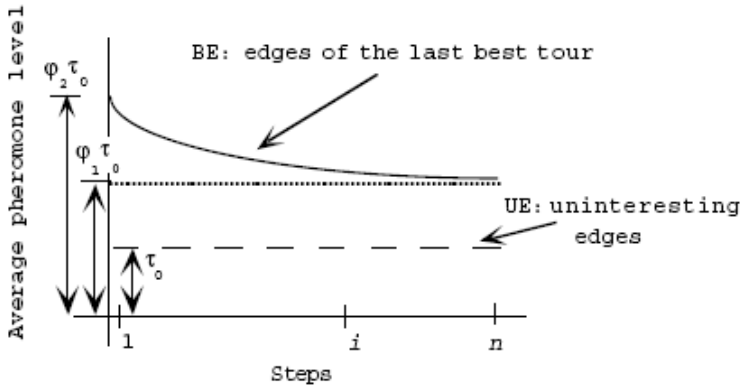


Figure 2.4: Change in average pheromone level during an algorithm iteration for edges in the BE family. The average pheromone level on edges in BE starts at $\varphi_2\tau_0$ and decreases each time an ant visits an edge in BE. After one algorithm iteration, each edge in BE has been visited on average $m\dot{q}_0$ times, and the final value of the pheromone level is $\varphi_1\tau_0$

algorithm). Under the hypothesis that the optimal values of $\varphi_1$ and $\varphi_2$ are known, an estimate of the optimal number of ants can be computed as follows. The local updating rule is a first-order linear recurrence relation of the form $T_z = T_{z-1}(1-\rho) + \tau_0\rho$ , which has closed form given by $T_z = T_0(1-\rho)^z - \tau_0(1-\rho)^z + \tau_0$. Knowing that just before global updating $T_0 = \varphi_2\tau_0$ (this corresponds to the start point of the BE curve in Figure 2.4), and that after all ants have built their tour and just before global updating, $T_z = \varphi_1\tau_0$ (this corresponds to the end point of the BE curve in Figure 2.4), we obtain $\varphi_1 = \varphi_2(1-\rho)^z - (1-\rho)^z + 1$.
Considering the fact that edges in BE are chosen by each ant with a probability $> q_0$, then a good approximation to the number $z$ of ants that locally update edges in BE is given by $z = m \cdot q_0$. Substituting in the above formula we obtain the following estimate of the optimal number of ants

$$m = \frac{\log(\varphi_1 - 1) - \log(\varphi_2 - 1)}{q_0 \cdot \log(1-\rho)} \tag{2.11}$$

This formula essentially shows that the optimal number of ants is a function of $\varphi_1$ and $\varphi_2$. Unfortunately, up to now, we have not been

able to identify the form of the functions $\varphi_1(n)$ and $\varphi_2(n)$, which would tell how $\varphi_1$ and $\varphi_2$ change as a function of the problem dimension. Still, experimental observation shows that ACS works well when the ratio $(\varphi_1 - 1)/(\varphi_2 - 1) \approx 0.4$, which gives $m = 10$.

## 2.5 Cooperation Among Ants

This section presents the results of two simple experiments which show that ACS effectively exploits pheromone-mediated cooperation. Since artificial ants cooperate by exchanging information via pheromone, to have noncooperating ants it is enough to make ants blind to pheromone. In practice this is obtained by deactivating Equation 2.5 and Equation 2.10, and setting the initial level of pheromone to $\tau_0 = 1$ on all edges. When comparing a colony of cooperating ants with a colony of noncooperating ants, to make the comparison fair, we use CPU time to compute performance indexes so as to discount for the higher complexity, due to pheromone updating, of the cooperative approach.

In the first experiment, the distribution of *first finishing times*, defined as the time elapsed until the first optimal solution is found, is used to compare the cooperative and the noncooperative approaches. The algorithm is run 10,000 times, and then we report on a graph the probability distribution (density of probability) of the CPU time needed to find the optimal value (e.g., if in 100 trials the optimum is found after exactly 220 iterations, then for the value 220 of the abscissa we will have P(220) = 100/10,000). Figure 2.5 shows that cooperation greatly improves the probability of finding quickly an optimal solution.

In the second experiment (Figure 2.6) the best solution found is plotted as a function of time (ms) for cooperating and noncooperating ants. The number of ants is fixed for both cases: $m = 4$. It is interesting to note that in the cooperative case, after 300 ms, ACS always found the optimal solution, while noncooperating ants where not able to find it after 800 ms. During the first 150 ms (i.e., before the two lines in Figure 2.6 cross) noncooperating ants outperform cooperating ants. Good values of pheromone level are still being learned and therefore the overhead due to pheromone updating is not yet compensated by the advantages which pheromone can provide in terms of directing the search towards good solutions.

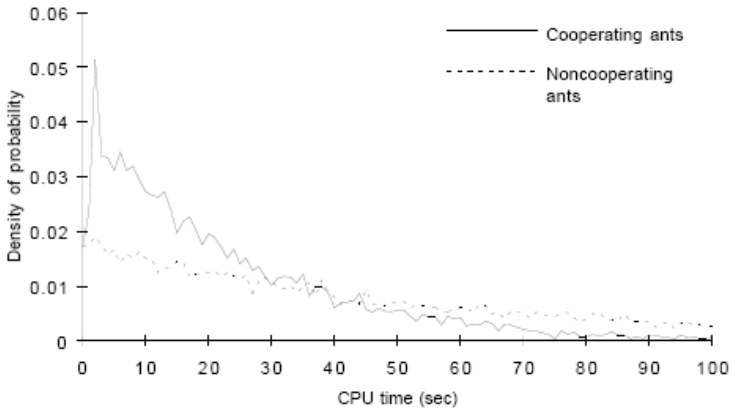


Figure 2.5: Cooperation changes the probability distribution of first finishing times: cooperating ants have a higher probability to find quickly an optimal solution. Test problem: CCAO Golden & Stewart (1985). The number of ants was set to $m = 4$

### 2.5.1 The Importance of the Pheromone and the Heuristic Function

Experimental results have shown that the heuristic function $\mu$ is fundamental in making the algorithm find good solutions in a reasonable time. In fact, when $\beta = 0$, ACS performance worsens significantly (see the ACS no heuristic graph in Figure 2.7). Figure 2.7 also shows the behavior of ACS in an experiment in which ants neither sense nor deposit pheromone (ACS no pheromone graph). The result is that not using pheromone also deteriorates performance. This is a further confirmation of the results on the role of cooperation presented in Section 2.4.

The reason ACS without the heuristic function performs better than ACS without pheromone is that in the first case, although not helped by heuristic information, ACS is still guided by reinforcement provided by the global updating rule in the form of pheromone, while in the second case ACS reduces to a stochastic multi-greedy algorithm.

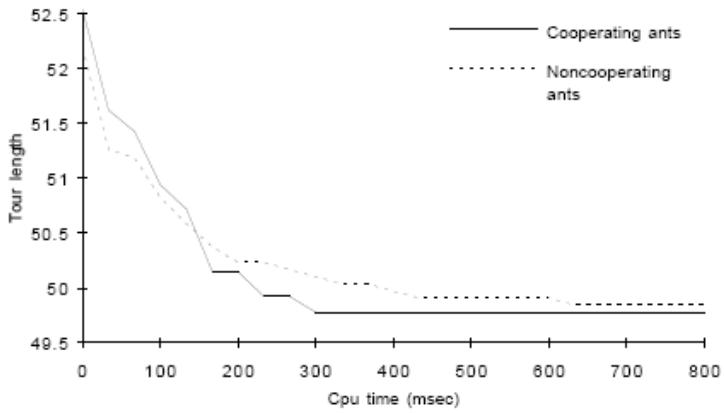


Figure 2.6: Cooperating ants find better solutions in a shorter time. Test problem: CCAO (Golden & Stewart 1985). Average on 25 runs. The number of ants was set to $m = 4$.

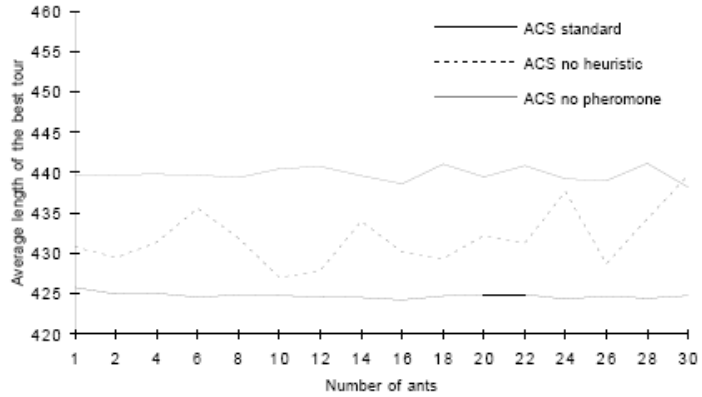


Figure 2.7: Comparison between ACS standard, ACS with no heuristic (i.e., we set $\beta = 0$), and ACS in which ants neither sense nor deposit pheromone. Problem: Oliver30. Averaged over 30 trials, 10,000/m iterations per trial

## 2.6 ACS: Some Computational Results

We report on two sets of experiments. The first set compares ACS with other heuristics. The choice of the test problems was dictated by published results found in the literature. The second set tests ACS on some larger problems. Here the comparison is performed only with respect to the optimal or the best known result. The behavior of ACS is excellent in both cases.

Most of the test problems can be found in TSPLIB: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/. When they are not available in this library we explicitly cite the reference where they can be found.

Given that during an iteration of the algorithm each ant produces a tour, in the reported results the total number of tours generated is given by the number of iterations multiplied by the number of ants. The result of each trial is given by the best tour generated by the ants. Each experiment consists of at least 15 trials.

### 2.6.1 Comparison with Other Heuristics

To compare ACS with other heuristics we consider two sets of TSP problems. The first set comprises five randomly generated 50-city problems, while the second set is composed of three geometric problems [1] of between 50 and 100 cities. It is important to test ACS on both random and geometric instances of the TSP because these two classes of problems have structural differences that can make them difficult for a particular algorithm and at the same time easy for another one.

Table 2.3 reports the results on the random instances. The heuristics with which we compare ACS are simulated annealing (SA), elastic net (EN), and self organizing map (SOM). Results on SA, EN, and SOM are from Durbin & Willshaw (1987) and Potvin (1993). ACS was run for 2,500 iterations using 10 ants (this amounts to approximately the same number of tour searched by the heuristics with which we compare our results). ACS results are averaged over 25 trials. The best average tour length for each problem is in boldface: ACS almost always offers the best performance.

[1] (Geometric problems are problems taken from the real world (for example, they are generated choosing real cities and real distances))

Table 2.3: Comparison of ACS with other heuristics on random instances of the symmetric TSP. Comparisons on average tour length obtained on five 50-city problems.

| Problem name | ACS (average) | SA (average) | EN (average) | SOM (average) |
|---|---|---|---|---|
| City Set 1 | **5.88** | **5.88** | 5.98 | 6.06 |
| City Set 2 | 6.05 | **6.01** | 6.03 | 6.25 |
| City Set 3 | **5.58** | 5.65 | 5.70 | 5.83 |
| City Set 4 | **5.74** | 5.81 | 5.86 | 5.87 |
| City Set 5 | **6.18** | 6.33 | 6.49 | 6.70 |

Table 2.4 reports the results on the geometric instances. The heuristics with which we compare ACS in this case are a genetic algorithm (GA), evolutionary programming (EP), and simulated annealing (SA). ACS is run for 1,250 iterations using 20 ants (this amounts to approximately the same number of tours searched by the heuristics with which we compare our results). ACS results are averaged over 15 trials. In this case comparison is performed on the best results, as opposed to average results as in previous Table 2.3, this choice was dictated by the availability of published results). The difference between integer and real tour length is that in the first case distances between cities are measured by integer numbers, while in the second case by floating point approximations of real numbers.

Results using EP are from Fogel (1993), and those using GA are from Whitley et al. (1989) for Eil50, and Eil75, and from Bersini et al. (1995) for KroA100. Results using SA are from Lin et al. (1993). Eil50, Eil75 are from Eilon et al. (1969) and are included in TSPLIB with an additional city as Eil51.tsp and Eil76.tsp. KroA100 is also in TSPLIB. The best result for each problem is in boldface. Again, ACS offers the best performance in nearly every case. Only for the Eil50 problem does it find a slightly worse solution using real-valued distance as compared with EP, but ACS only visits 1,830 tours, while EP used 100,000 such evaluations (although it is possible that EP found its best solution earlier in the run, this is not specified in the paper of Fogel (1993).

Table 2.4: Comparison of ACS with other heuristics on geometric instances of the symmetric TSP. We report the best integer tour length, the best real tour length (in parentheses) and the number of tours required to find the best integer tour length (in square brackets). N/A means "not available." In the last column the optimal length is available only for integer tour lengths.

| Problem Name | ACS | GA | EP | SA | Optimum |
|---|---|---|---|---|---|
| EIL50 | **425** | 428 | 426 | 443 | **425** |
| (50-city problem) | (427.96) | (N/A) | **(427.86)** | (N/A) | (N/A) |
| | **[1'830]** | [25'000] | [100'000] | [68'512] | |
| EIL75 | **535** | 545 | 542 | 580 | **535** |
| (75-city problem) | **(542.37)** | (N/A) | (549.18) | (N/A) | (N/A) |
| | **[3'480]** | [80'000] | [325'000] | [173'250] | |
| Kroa100 | **21'282** | 21'761 | N/A | N/A | **21'282** |
| (100-city problem) | (21'285.44) | (N/A) | (N/A) | (N/A) | (N/A) |
| | **[4'820]** | [103'000] | [N/A] | [N/A] | |

### 2.6.2 ACS on Some Bigger Problems

When trying to solve big TSP problems it is common practice (Lawler et al. 1985; Reinelt 1991) to use a data structure known as *candidate list*. A candidate list is a list of preferred cities to be visited; it is a static data structure which contains, for a given city $i$, the $cl$ closest cities, ordered by increasing distances; $cl$ is a parameter that we set to $cl = 15$ in our experiments. We implemented therefore a version of ACS (Gambardella & Dorigo 1996) which incorporates a candidate list: An ant in this extended version of ACS first chooses the city to move to among those belonging to the candidate list. Only if none of the cities in the candidate list can be visited then it considers the rest of the cities. ACS with candidate list (see Table 2.5) was able to find good results for problems up to more than 1,500 cities. The time to generate a tour grows only slightly more than linearly with the number of cities (this is much better than the quadratic growth obtained without the candidate list): On a Sun Sparcserver (50 MHz) it took approximately 0.02 sec of CPU time to generate a tour for the d198 problem, 0.05 sec for the pcb442, 0.07 sec for the att532, 0.13 sec for the rat783, and 0.48 sec for the fl1577 (the reason for the more than linear increase in time is that the number of failures, that is, the number of times an ant has to choose the next

Table 2.5: ACS performance for some bigger geometric problems (over 15 trials). We report the integer length of the shortest tour found, the number of tours required to find it, the average integer length, the standard deviation, the optimal solution (for fl1577 we give, in square brackets, the known lower and upper bounds, given that the optimal solution is not known), and the relative error of ACS.

| Problem name | ACS best integer length (1) | ACS number of tours generated to best | ACS average integer length | Standard deviation | Optimum (2) | Relative error $\frac{(1)-(2)}{(2)} \cdot 100$ |
|---|---|---|---|---|---|---|
| d198 (198-city problem) | 15,888 | 585,000 | 16,054 | 71 | 15,780 | 0.68% |
| pcb442 (442-city problem) | 51,268 | 595,000 | 51,690 | 188 | 50,779 | 0.96% |
| att532 (532-city problem) | 28,147 | 830,658 | 28,523 | 275 | 27,686 | 1.67% |
| rat783 (783-city problem) | 9,015 | 991,276 | 9,066 | 28 | 8,806 | 2.37% |
| fl1577 (1577-city problem) | 22,977 | 942,000 | 23,163 | 116 | $[22,204 - 22,249]$ | $3.27 \div 3.48\%$ |

city outside of the candidate list, increases with the problem dimension).

## 2.7 ACS Plus Local Search

In Section 2.3 we have shown that ACS is competitive with other nature-inspired algorithms on some relatively simple problems. On the other hand, in the past years a lot of work has been done to define ad-hoc tour improvement heuristics, (see Johnson & McGeoch 2002 for an overview), to solve the TSP. Tour improvement heuristics (Section 1.3) start from a given tour and attempt to reduce its length by exchanging edges chosen according to some heuristic rule until a local optimum is found. The most used and well-known tour improvement heuristics are 2-opt (Algorithm 3) and 3-opt (Lin 1965), and Lin-Kernighan (Lin & Kernighan 1973) in which respectively two, three, and a variable number of edges are exchanged

It has been shown (Johnson & McGeoch 1997, Section 1.5) that it is very effective to couple an improvement heuristic with mutations of the last (or of the best) solution produced, rather than iteratively executing a tour improvement heuristic starting from solutions generated randomly

**Algorithm 13** Ant Colony System (ACS) Coupled with a Local Search Procedure

Initialize
**repeat** {*at this level each loop is called an iteration*}
  Each ant is positioned on a starting node
  **repeat** {*at this level each loop is called a step*}
    Each ant applies a state transition rule to incrementally build a solution and a local pheromone updating rule
  **until** all ants have built a complete solution
  Each ant is brought to a local minimum using a tour improvement heuristic based on 3-opt
  A global pheromone updating rule is applied
**until** End condition

or by a constructive heuristic. An example of successful application of the above alternate strategy is TSP-GA (Freisleben & Merz 1996a; Merz & Freisleben 1997) in which a genetic algorithm is used to generate new solutions to be locally optimized by a tour improvement heuristic.

ACS is a tour construction heuristic which, like Freisleben and Merz's genetic algorithm, after each iteration produces a set of feasible solutions which are in some sense a mutation of the previous best solution. It is therefore a reasonable guess that adding a tour improvement heuristic to ACS could make it competitive with the best algorithms.

We have therefore added a tour improvement heuristic to ACS (Algorithm 13). In order to maintain ACS ability to solve both TSP and ATSP problems we have decided to base the local optimization heuristic on a *restricted 3-opt* procedure (Johnson & McGeoch 1997; Kanellakis & Papadimitriou 1980), that, while inserting/removing three edges on the path, considers only 3-opt moves that do not revert the order in which the cities are visited. The resulting hybrid algorithm is called ACS-3-opt (Algorithm 13). In this way the same procedure can be applied to symmetric and asymmetric TSPs, avoiding unpredictable tour length changes. In addition, when a candidate edge $(i, j)$ to be removed is selected, the *restricted 3-opt* procedure restricts the search for the other two edges to those nodes $p$ belonging to edge $(p, q)$ such as $d_{iq} < d_{ij}$.

The implementation of the *restricted 3-opt* includes some typical tricks which accelerate its use for TSP/ATSP problems. First, search for the candidate nodes during the *restricted 3-opt* procedure is only made inside

the candidate list (Johnson & McGeoch 1997). Second, the procedure uses a data structure called *don't look bit* (Bentley 1992) in which each bit is associated to a node of the tour. At the beginning of the local optimization procedure all the bits are turned off and the bit associated to node $i$ is turned on when a search for an improving move starting from $i$ fails. The bit associated to node $i$ is turned off again when a move involving $i$ is performed. Third, only in the case of symmetric TSPs, while searching for 3-opt moves starting from a node $i$ the procedure also considers possible 2-opt moves with $i$ as first node: the move executed is the best one among those proposed by 3-opt and those proposed by 2-opt. Last, a traditional array data structure to represent candidate lists and tours is used (Fredman et al. 1995) for more sophisticated data structures).

ACS-3-opt also uses candidate lists in its constructive part; if there is no feasible node in the candidate list it chooses the closest node out of the candidate list (this is different from what happens in ACS where, in case the candidate list contains no feasible nodes, then any of the remaining feasible cities can be chosen with a probability which is given by the normalized product of pheromone and closeness). This is a reasonable choice since most of the search performed by both ACS and the local optimization procedure is made using edges belonging to the candidate lists. It is therefore pointless to direct search by using pheromone levels which are updated only very rarely.

### 2.7.1 Experimental Results

The experiments on ATSP problems presented in this section have been executed on a SUN Ultra1 SPARC Station (167Mhz), while experiments on TSP problems on a SGI Challenge L server with eight 200 MHz CPU's, using only a single processor due to the sequential implementation of ACS-3-opt. For each test problem 10 trials have been executed. ACS-3-opt parameters were set to the following values (except if differently indicated): $m = 10$, $\beta = 2$, $q_0 = 0.98$, $\alpha = \rho = 0.1$, $\tau_0 = (n \cdot L_{nn}^{-1})$, $cl = 20$.

#### Asymmetric TSP problems

The results obtained with ACS-3-opt on ATSP problems are quite impressive. Experiments were run on the set of ATSP problems proposed

Table 2.6: Results obtained by ACS-3-opt on ATSP problems taken from the First International Contest on Evolutionary Optimization (Bersini et al. 1996). We report the length of the best tour found by ACS-3-opt, the CPU time used to find it, the average length of the best tour found and the average CPU time used to find it, the optimal length and the relative error of the average result with respect to the optimal solution.

| Problem Name (cities) | ACS-3-opt best result (lenght) | ACS-3-opt best result (sec) | ACS-3-opt average (lenght) | ACS-3-opt average (sec) | Optimum | % Error |
|---|---|---|---|---|---|---|
| p43 (43-city problem) | 2'810 | 1 | 2'810 | 2 | 2'810 | 0.00 % |
| ry48 (48-city problem) | 14'422 | 2 | 14'422 | 19 | 14'422 | 0.00 % |
| ft70 (70-city problem) | 38'673 | 3 | 38'679.8 | 6 | 38'673 | 0.02 % |
| kro124p (100-city problem) | 36'230 | 3 | 36'230 | 25 | 36'230 | 0.00 % |
| ftv170[2] (170-city problem) | 2'755 | 17 | 2'755 | 68 | 2'755 | 0.00 % |

in the First International Contest on Evolutionary Optimization (Bersini et al. 1996). For all the problems ACS-3-opt reached the optimal best known solution in a few seconds (see Table 2.6) in all the ten trials, except in the case of ft70, a problem considered relatively hard, where the optimum was reached 8 out of 10 times.

In Table 2.7 results obtained by ACS-3-opt are compared with those obtained by ATSP-GA (Freisleben & Merz 1996a), the winner of the ATSP competition. ATSP-GA is based on a genetic algorithm that starts its search from a set of individuals generated using a nearest neighbor heuristic. Individuals are strings of cities which represent feasible solutions. At each step two parents $x$ and $y$ are selected and their edges are recombined using a procedure called DPX-ATSP. DPX-ATSP first deletes all edges in $x$ that are not contained in $y$ and then reconnects the segments using a greedy heuristic based on a nearest neighbor choice. The new individuals are brought to the local optimum using a 3-opt procedure, and the new population is generated after the application of a mutation operation that randomly removes and reconnects some edges in the tour.

The 3-opt procedure used by ATSP-GA is very similar to our restricted 3-opt, which makes the comparison between the two approaches

Table 2.7: Comparison between ACS-3-opt and ATSP-GA on ATSP problems taken from the First International Contest on Evolutionary Optimization. We report the average length of the best tour found, the average CPU time used to find it, and the relative error with respect to the optimal solution for both approaches

| Problem Name (cities) | ACS-3-opt average (lenght) | ACS-3-opt average (sec) | ACS-3-opt %error | ATSP-GA average (lenght) | ATSP-GA average (sec) | ATSP-GA %error |
|---|---|---|---|---|---|---|
| p43 (43-city problem) | 2'810 | 2 | 0.00 % | 2'810 | 10 | 0.00 % |
| ry48 (48-city problem) | 14'422 | 19 | 0.00 % | 14'440 | 30 | 0.12 % |
| ft70 (70-city problem) | 38'679.8 | 6 | 0.02 % | 38'683.8 | 639 | 0.03 % |
| kro124p (100-city problem) | 36'230 | 25 | 0.00 % | 36'235.3 | 115 | 0.01 % |
| ftv170 (170-city problem) | 2'755 | 68 | 0.00 % | 2'766.1 | 211 | 0.40 % |

straightforward. ACS-3-opt outperforms ATSP-GA in terms of both closeness to the optimal solution and of CPU time used. Moreover, ATSP-GA experiments have been performed using a DEC Alpha Station (266 MHz), a machine faster than our SUN Ultra1 SPARC Station.

**Symmetric TSP problems**

If we now turn to symmetric TSP problems, it turns out that STSP-GA (STSP-GA experiments have been performed using a 175 MHz DEC Alpha Station), the algorithm that won the First International Contest on Evolutionary Optimization in the symmetric TSP category, outperforms ACS-3-opt (see Tables 2.8, 2.9). The results used for comparisons are those published in Freisleben & Merz (1996b), which are slightly better than those published in Freisleben & Merz (1996a). Our results are, on the other hand, comparable to those obtained by other algorithms considered to be very good. For example, on the lin318 problem ACS-3-opt has approximately the same performance as the large step Markov chain algorithm (Martin et al. 1992, Section 1.5). This algorithm is based on a simulated annealing mechanism that uses as improvement heuristic a restricted 3-opt heuristic very similar to ours (the only difference is that they do not consider 2-opt moves) and as mutation procedure a non-sequential-4-changes called double-bridge (Section 1.2). The double-

Table 2.8: Results obtained by ACS-3-opt on TSP problems taken from the First International Contest on Evolutionary Optimization. We report the length of the best tour found by ACS-3-opt, the CPU time used to find it, the average length of the best tour found and the average CPU time used to find it, the optimal length and the relative error of the average result with respect to the optimal solution.

| Problem Name (cities) | ACS-3-opt best result (lenght) | ACS-3-opt best result (sec) | ACS-3-opt average (lenght) | ACS-3-opt average (sec) | Optimum | % Error |
|---|---|---|---|---|---|---|
| d198 (198-city problem) | 15,780 | 16 | 15,781.7 | 238 | 15,780 | 0.01 % |
| lin318 [3] (318-city problem) | 42,029 | 101 | 42,029 | 537 | 42,029 | 0.00 % |
| att532 (532-city problem) | 27,693 | 133 | 27,718.2 | 810 | 27,686 | 0.11 % |
| rat783 (783-city problem) | 8,818 | 1,317 | 8,837.9 | 1,280 | 8,806 | 0.36 % |

bridge mutation has the property that it is the smallest change (4 edges) that can not be reverted in one step by 3-opt, LK and 2-opt.

## 2.8 Discussion and Conclusions

An intuitive explanation of how ACS works, which emerges from the experimental results presented in the preceding sections, is as follows. Once all the ants have generated a tour, the best ant deposits (at the end of iteration $t$) its pheromone, defining in this way a preferred tour for search in the following algorithm iteration $t + 1$. In fact, during iteration $t + 1$ ants will see edges belonging to the best tour as highly desirable and will choose them with high probability. Still, guided exploration (see Equations 2.4 and 2.1) together with the fact that local updating (Equation 2.10) eats pheromone away (i.e., it diminishes the amount of pheromone on visited edges, making them less desirable for future ants) allowing for the search of new, possibly better tours in the neighborhood of the previous best tour. So ACS can be seen as a sort of guided parallel stochastic search in the neighborhood of the best tour. In the last years there has been growing interest in the application of ant colony algorithms to difficult combinatorial problems. A first example is the work of O. Schoonderwoerd & Rothkrantz (1996) who apply an ant

Table 2.9: Comparison between ACS-3-opt and STSP-GA on TSP problems taken from the First International Contest on Evolutionary Optimization. We report the average length of the best tour found, the average CPU time used to find it, and the relative error with respect to the optimal solution for both approaches.

| Problem Name (cities) | ACS-3-opt average (lenght) | ACS-3-opt average (sec) | ACS-3-opt %error | ATSP-GA average (lenght) | ATSP-GA average (sec) | ATSP-GA %error |
|---|---|---|---|---|---|---|
| d198 (198-city problem) | 15,781.7 | 238 | 0.01 % | 15,780 | 253 | 0.00 % |
| lin318 (318-city problem) | 42,029 | 537 | 0.00 % | 42,029 | 2,054 | 0.00 % |
| att532 (532-city problem) | 27,718.2 | 810 | 0.11 % | 27,693.7 | 11,780 | 0.03 % |
| rat783 (783-city problem) | 8,837.9 | 1,280 | 0.36 % | 8,807.3 | 21,210 | 0.01 % |

colony algorithm to the load balancing problem in telecommunications networks. Their algorithm takes inspiration from the same biological metaphor as AS, although their implementation differs in many details due to the different characteristics of the problem. Another interesting research is that of Stützle & Hoos (1998) who have studied various extensions of AS to improve its performance: in Stützle & Hoos (1998) they impose an upper and lower bound on the value of pheromone on edges, in Stützle & Hoos (1997) they add local search, much in the same spirit as we did in the previous Section 2.7. Besides the two works above, among the *nature-inspired* heuristics, the closest to ACS seems to be Baluja and Caruanas Population Based Incremental Learning (PBIL) (Baluja & Caruana 1995). PBIL, which takes inspiration from genetic algorithms, maintains a vector of real numbers, the generating vector, which plays a role similar to that of the population in GAs. Starting from this vector, a population of binary strings is randomly generated: Each string in the population will have the $i$-th bit set to 1 with a probability which is a function of the $i$-th value in the generating vector (in practice, values in the generating vector are normalized to the interval [0, 1] so that they can directly represent the probabilities). Once a population of solutions is created, the generated solutions are evaluated and this evaluation is used to increase (or decrease) the probabilities in the generating vector so that good (bad) solutions in the future generations will be produced with higher (lower) probability. When applied to TSP, PBIL uses the

following encoding: a solution is a string of size $n \log 2$ bits, where $n$ is the number of cities; each city is assigned a string of length $n \log 2$ which is interpreted as an integer. Cities are then ordered by increasing integer values; in case of ties the leftmost city in the string comes first in the tour. In ACS, the pheromone matrix plays a role similar to Balujas generating vector, and pheromone updating has the same goal as updating the probabilities in the generating vector. Still, the two approaches are very different since in ACS the pheromone matrix changes while ants build their solutions, while in PBIL the probability vector is modified only after a population of solutions has been generated. Moreover, ACS uses heuristic to direct search, while PBIL does not. In conclusion, in this chapter we have shown that ACS is an interesting approach to parallel stochastic optimization of the TSP. ACS has been shown to compare favorably with previous attempts to apply other heuristic algorithms like genetic algorithms, evolutionary programming, and simulated annealing. Nevertheless, competition on the TSP is very tough, and a combination of a constructive method which generates good starting solution with local search which takes these solutions to a local optimum seems to be the best strategy (Johnson & McGeoch 1997). We have shown that ACS is also a very good constructive heuristic to provide such starting solutions for local optimizers.

# Chapter 3

# MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows

## 3.1 Introduction

This chapter presents MACS-VRPTW, a Multiple Ant Colony System for Vehicle Routing Problems with Time Windows (Gambardella et al. 1999). MACS-VRPTW, is based on ACS (Chapter 2, Gambardella & Dorigo 1996; Dorigo & Gambardella 1997), and, more generally, on ACO (Section 1.4.4, Dorigo et al. 1999).
The basic ACO idea is that a large number of simple artificial agents are able to build good solutions to hard combinatorial optimization problems via low-level based communications. Real ants cooperate in their search for food by depositing chemical traces (pheromones) on the floor. An artificial ant colony simulates this behavior. Artificial ants cooperate by using a common memory that corresponds to the pheromone deposited by real ants. This artificial pheromone is one of the most important components of ant colony optimization and is used for constructing new solutions. In the ACO metaheuristic, artificial pheromone is accumulated at run-time during the computation. Artificial ants are implemented as parallel processes whose role is to build problem solutions using a constructive procedure driven by a combination of artificial

pheromone, problem data and a heuristic function used to evaluate successive constructive steps.
ACO algorithms have been shown to be very efficient when combined with specialized local search procedures to solve the symmetric and asymmetric traveling salesman problems (TSP/ATSP, Chapter 2, Dorigo & Gambardella 1997; Stützle 1999; Stützle & Dorigo 1999b), vehicle routing problems (VRP Gambardella et al. 1999) and the quadratic assignment problem (QAP, Gambardella et al. 1999; Stützle & Dorigo 1999a). One of the most efficient ACO based implementations has been ACS, (Chapter 2, Gambardella & Dorigo 1996; Dorigo & Gambardella 1997) that introduced a particular pheromone trail updating procedure useful to intensify the search in the neighborhood of the best computed solution.
This chapter presents an ACS extension called MACS-VRPTW a Multiple Ant Colony System for Vehicle Routing Problems with Time Windows, which is able to solve the vehicle routing problem with time windows (VRPTW).
Vehicle routing problems with time windows VRPTW is defined as the problem of minimizing time and costs in case a fleet of vehicles has to distribute goods from a depot to a set of customers. The VRPTW considered in this chapter minimizes a multiple, hierarchical objective function: the first objective is to minimize the number of tours (or vehicles) and the second is to minimize the total travel time. A solution with a lower number of tours is always preferred to a solution with a higher number of tours even if the travel time is higher. This hierarchical objectives VRPTW is very common in the literature and in case problem constraints are very tight (for example when the total capacity of the minimum number of vehicles is very close to the total volume to deliver or when customers time windows are narrow), the two objectives can be antagonistic: the minimum travel time solution can include a number of vehicles higher than the solution with minimum number of vehicles (see e.g. Kohl et al. 1997). To adapt ACS for these multiple objectives the idea is to define two ACS colonies, each dedicated to the optimization of a different objective function. MACS-VRPTW is in fact organized with a hierarchy of artificial ant colonies designed to successively optimize a multiple objective function: the first colony minimizes the number of vehicles while the second colony minimizes the traveled distances. Cooperation between colonies is performed by exchanging information through pheromone updating. We show that MACS-VRPTW is competitive both in terms of

solution quality and computation time. Moreover, MACS-VRPTW has been able to improve some of the best known solutions for a number of problem instances in the literature.

This chapter is organized as follows: First in Section 3.2, vehicle routing problems are introduced by presenting a formal definition of the capacitated vehicle routing problem (CVRP) and the vehicle routing problem with time windows (VRPTW). Then Section 3.3, extends ACS to deal with VRPTW and the resulting MACS-VRPTW is investigated by presenting its main components. Last in Section 3.4 numerical results are reported and some conclusions are drawn.

## 3.2 Vehicle Routing Problems

The most elementary version of the vehicle routing problem is the capacitated vehicle routing problem (CVRP). The CVRP is described as follows: $n$ customers must be served from a unique depot. Each customer asks for a quantity $q_i$ of goods ($i = 1, \ldots, n$) and a vehicle of capacity $Q$ is available to deliver goods. Since the vehicle capacity is limited, the vehicle has to periodically return to the depot for reloading. In the CVRP, it is not possible to split customer delivery. Therefore, a CVRP solution is a collection of tours where each customer is visited only once and the total tour demand is at most $Q$. From a graph theoretical point of view the CVRP may be stated as follows: Let $G = (C, L)$ be a complete graph with node set $C = (c_0, c_1, c_2, ..., c_n)$ and arc set $L = (i, j) : i, j \in C, i \neq j$. In this graph model, $c_0$ is the depot and the other nodes are the customers to be served. Each node is associated with a fixed quantity $q_i$ of goods to be delivered (a quantity $q_0 = 0$ is associated to the depot $c_0$). To each arc $(c_i, c_j)$ is associated a value $t_{ij}$ representing the travel time between $c_i$ and $c_j$. The goal is to find a set of tours of minimum total travel time. Each tour starts from and terminates at the depot $c_0$, each node $c_i (i = 1, ..., n)$ must be visited exactly once, and the quantity of goods to be delivered on a route should never exceed the vehicle capacity $Q$. One of the most successful exact approaches for the CVRP is the k-tree method of (Fisher 1994) that succeeded in solving a problem with 71 customers. However, there are smaller instances that have not been exactly solved yet. To treat larger instances, or to compute solutions faster, heuristic methods must be used. The most used heuristic methods are tabu searches (Taillard 1993; Rochat & Taillard 1995; Rego & Roucairol 1996; Xu & Kelly

1996) and large neighborhood search (Shaw 1998). The CVRP can be extended in many ways. For example a service time $s_i$ for each customer (with $s_0 = 0$) and a time limit over the duration of each tour can be considered. The goal is again to search for a set of tours that minimizes the sum of the travel times. An important extension of the CVRP that is the subject of this chapter is the vehicle routing problem with time windows (VRPTW). In addition to the mentioned CVRP features, this problem includes, for the depot and for each customer $c_i (i = 0, ..., n)$ a time window $[b_i, e_i]$ during which the customer has to be served (with $b_0$ the earliest start time and $e_0$ the latest return time of each vehicle to the depot). The tours are performed by a fleet of $v$ identical vehicles. The additional constraints are that the service beginning time at each node $c_i (i = 1, ..., n)$ must be greater than or equal to $b_i$, the beginning of the time window, and the arrival time at each node $c_i$ must be lower than or equal to $e_i$, the end of the time window.

In case the arrival time is less than $b_i$, the vehicle has to wait till the beginning of the time window before starting servicing the customer. In the literature the fleet size $v$ is often a variable and a very common objective is to minimize $v$. This objective is related to the real situation in which driver salaries are variable costs for the company or when the company has to rent vehicles to perform deliveries. Usually, two different solutions with the same number of vehicles are ranked by alternative objectives such as the total traveling time or total delivery time (including waiting and service times). These objectives are also used for companies owning a fixed fleet of vehicles. A number of exact and heuristic methods exist for the VRPTW. Among exact methods, that of (Kohl et al. 1997) is one of the most efficient and succeeded in solving a number of 100 customer instances. Note that exact methods are more efficient in case the solution space is restricted by narrow time windows since less combinations of customers are possible to define feasible tours. The most successful heuristic methods for the VRPTW are adaptive memory programs (see Taillard et al. 1998 for an introduction to adaptive memory programming), embedding tabu searches (Rochat & Taillard 1995; Taillard et al. 1997; Badeau et al. 1997), guided local search (Kilby et al. 1999) and large neighborhood search (Shaw 1998).

## 3.3 MACS-VRPTW for Vehicle Routing Problems with Time Windows

The first ant inspired algorithm for vehicle routing problems has been designed by Bullnheimer et al. (1999b) who considered the most elementary version of the problem: the capacitated vehicle routing problem (CVRP). This chapter considers a more elaborated vehicle routing problem with two objective functions: (i) the minimization of the number of tours (or vehicles) and (ii) the minimization of the total travel time, where number of tours minimization takes precedence over travel time minimization. A solution with a lower number of tours is always preferred to a solution with a higher number of tours even if the travel time is higher. This hierarchical objectives VRPTW is very common in the literature and in case problem constraints are very tight (for example when the total capacity of the minimum number of vehicles is very close to the total volume to deliver or when customers time windows are narrow), the two objectives can be antagonistic: the minimum travel time solution can include a number of vehicles higher than the solution with minimum number of vehicles.

In order to adapt ACS (Chapter 2) to multiple objectives the Multiple Ant Colony System for the VRPTW (MACS-VRPTW, Gambardella et al. 1999) has been defined. Briefly, in ACS (Algorithm 12) two measures are associated to each arc of the problem graph: the closeness $\mu_{ij}$, and the pheromone trail $\tau_{ij}$. Closeness, defined as the inverse of the arc length or the arc traveling time, is a static heuristic value that never changes for a given problem instance, while the pheromone trail is dynamically changed by ants at runtime. Therefore, the most important component of ACS is the management of pheromone trails which are used, in conjunction with the objective function, for constructing new solutions (Equations 2.1, 2.4). Informally, pheromone levels give a measure of how desirable it is to insert a given arc in a solution. Pheromone trails are used for exploration and exploitation. Exploration concerns the probabilistic choice of the components used to construct a solution: a higher probability is given to elements with a strong pheromone trail. Exploitation chooses the component that maximizes a blend of pheromone trail values and heuristic evaluations.
The goal of ACS is to find a shortest tour. In ACS $m$ ants build tours

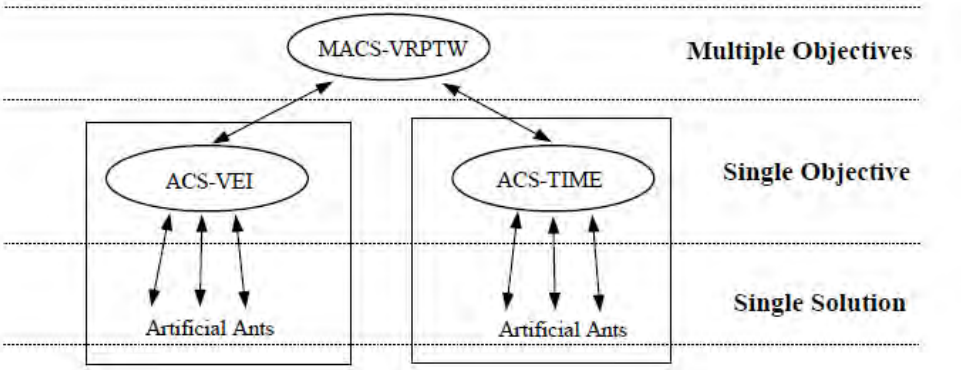


Figure 3.1: Architectur e of the Multiple Ant Colony System for the Vehicle Routing Problem with Time Windows

in parallel. Each ant is randomly assigned to a starting node and has to build a solution, that is, a complete tour. Once each ant has built a complete solution, this is tentatively improved using a local search procedure (Algorithm 13). Next, the best solution found from the beginning of the trial is used to globally update the pheromone trails (Equations 2.5). The rationale is that in this way a *preferred route* is memorized in the pheromone trail matrix and future ants will use this information to generate new solutions in a neighborhood of this preferred route. Then, the process is iterated by starting again $m$ ants until a termination condition is met (i.e. a fixed number of solutions has been generated or a fixed CPU time has elapsed).
In ACS, pheromone trail is updated not only globally but also locally. Local updating (Equation 2.10) is performed during solutions construction on the edges connecting two successive cities. Shortly, the effect of local updating is to change dynamically the desirability of these edges: every time an ant uses an edge the quantity of pheromone associated to this edge is decreased and the edge becomes less attractive.

In VRPTW the goal is to minimize two objective functions: the number of vehicles and the total traveling time. In the MACS-VRPTW algorithm (Figure 3.1, Algorithm 14) both objectives are optimized simultaneously by coordinating the activities of two ACS based colonies. The goal of the first colony, ACS-VEI, is to try to diminish the num-

**Algorithm 14** MACS-VRPTW: Multiple Ant Colony System for Vehicle Routing Problems with Time Windows

**Initialize**
/* $L_{gb}$ is the best feasible solution: lowest number of vehicles and shortest travel time. */
$L_{gb} \leftarrow$ compute a feasible initial solution with unlimited number of vehicles produced with a nearest neighbor heuristic
$vehicles = \#active\_vehicles(L_{gb})$ /* compute the number of active vehicles in the feasible solution $L_{gb}$ */
**repeat**
  $vehicles = \#active\_vehicles(L_{gb})$
  Activate ACS-VEI($vehicles - 1$)
  Activate ACS-TIME($vehicles$)
  **while** ACS-VEI and ACS-TIME are active **do**
    WAIT an improved solution $L$ from ACS-VEI or ACS-TIME
    $L_{gb} \leftarrow L$
    **if** $\#active\_vehicles(L_{gb} < vehicles)$ **then**
      kill ACS-TIME and ACS-VEI
    **end if**
  **end while**
**until** a stopping criterion is met

ber of vehicles used, while the second colony, ACS-TIME, optimizes the feasible solutions found by ACS-VEI. The two colonies use independent pheromone trails but collaborate by sharing the variable $L_{gb}$ managed by MACS-VRPTW. Initially, $L_{gb}$ is a feasible VRPTW solution found with a nearest neighbor heuristic. Then, $L_{gb}$ is improved by the two colonies. When ACS-VEI is activated, it tries to find a feasible solution with one vehicle less than the number of vehicles used in $L_{gb}$. The goal of ACS-TIME is to optimize the total travel time of solutions that use as many vehicles as vehicles used in $L_{gb}$. $L_{gb}$ is updated each time one of the colonies computes an improved feasible solution. In case the improved solution contains less vehicles than the vehicles used in $L_{gb}$, MACS-VRPTW stops ACS-TIME and ACS-VEI. Then, the process is iterated and two new colonies are activated, working with the new, reduced number of vehicles.

ACS-TIME colony (Algorithm 15) is a traditional ACS based colony whose goal is to compute a tour as short as possible. In ACS-TIME $m$

artificial ants are activated to construct problems solutions $L_1, ..., L_m$. Each solution is build by calling the *new_active_ant* Algorithm 17, a constructive procedure explained in details in Section 3.3.2 that is similar to the ACS constructive procedure designed for the TSP (Section 2.3). When $L_1, ..., L_m$ have been computed, they are compared to $L_{gb}$ and, in case one solution is better, it is sent to MACS-VRPTW. MACS-VRPTW uses this solution to update $L_{gb}$. After solutions generation, the global updates are performed using Equation 2.5 and $L_{gb}$.

**Algorithm 15** ACS-TIME: Travel Time Minimization.

**Procedure** ACS-TIME(*vehicles*)
/* Parameter *vehicles* is the smallest number of vehicles for which a feasible solution has been computed */
**Initialize**
/* initialize pheromone and data structure using *vehicles* */
**repeat**
  **for each** *ant* $k$ **do**
    /* construct a solution $L_k$ */
    $L_k \leftarrow new\_active\_ant(k, local\_search = TRUE, 0)$
  **end for each**
  /* update the best solution if it is improved */
  **if** $\exists k : L_k \; is \; feasible$ **and** $L_k < L_{gb}$ **then**
    **send** $L_k$ to MACS-VRPTW
  /* perform global updating according to Equations 2.5 */
  $\tau_{ij} \leftarrow (1-\alpha) \cdot \tau_{ij} + \alpha \cdot \frac{1}{L_{gb}} \qquad \forall (i,j) \in L_{gb}$
**until** a stopping criterion is met

ACS-VEI colony (Algorithm 16) searches for a feasible solution by maximizing the number of visited customers. ACS-VEI starts its computation using $(v-1)$ vehicles, that is, one vehicle less than the smallest number of vehicles for which a feasible solution has been computed (i.e., the number of vehicles in $L_{gb}$). During this search the colony produces unfeasible solutions in which some customers are not visited. In ACS-VEI, the solution computed since the beginning of the trial with the highest number of visited customers is stored in the variable $L_{ACS-VEI}$. A solution is better than $L_{ACS-VEI}$ only when the number of visited customers is increased. Therefore ACS-VEI is different from the traditional ACS applied to the TSP. In ACS-VEI the current best solution

$L_{ACS-VEI}$ is the solution (usually unfeasible) with the highest number of visited customers, while in ACS applied to TSP (Algorithm 12) the current best solution is the shortest one.

In order to maximize the number of customers serviced, ACS-VEI manages a vector $IN$ of integers. The entry $IN_j$ stores the number of time customer $j$ has not been inserted in a solution. $IN$ is used by the constructive procedure *new_active_ant* for favoring the customers that are less frequently included in the solutions. In ACS-VEI, at the end of each cycle, pheromone trails are globally updated with two different solutions: $L_{ACS-VEI}$, the unfeasible solution with the highest number of visited customers and $L_{gb}$, the feasible solution with the lowest number of vehicles and the shortest travel time. Numerical experiments have shown that a double update greatly improves the system performances. Indeed, the updates with $L_{ACS-VEI}$ are not increasing the trails toward the customers that are not included in the solution. Since $L_{bg}$ is feasible, the updates with $L_{bg}$ are increasing trails toward all customers.

### 3.3.1 Solution Model

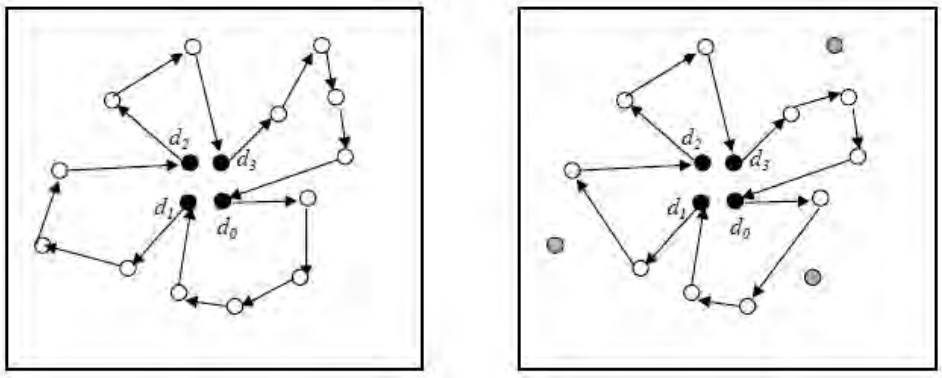


Figure 3.2: Feasible (left) and unfeasible (right) solutions for a vehicle routing problem with four duplicated depots and four active vehicles

MACS-VRPTW uses a solution model in which each ant builds a single tour (Figure 3.2). A solution is represented as follows: First, the depot with all its connections to/from the customers is duplicated a number of times equal to the number of available vehicles. Distances between copies of the depot are set to zero. This approach makes the vehicle routing

**Algorithm 16** ACS-VEI: Number of Vehicles Minimization.

**Procedure** ACS-VEI($s$)
/* Parameter $s$ is set to $vehicles - 1$, one vehicle less than the smallest number of vehicles for which a feasible solution has been computed
$\#visited\_customers(L)$ computes the number of customers that have been visited in solution $L$ */
**Initialize**
/* initialize pheromone and data structure using $s$ */
$L_{ACS-VEI} \leftarrow$ initial solution with $s$ vehicles produced with a nearest neighbor heuristic. /* $L_{ACS-VEI}$ is not necessary feasible */
**repeat**
  **for each** *ant* $k$ **do**
    /* construct a solution $L_k$ */
    $L_k \leftarrow new\_active\_ant(k, local\_search = FALSE, IN)$
    $\forall$ *customer* $j \notin L_k : IN_j \leftarrow IN_j + 1$
  **end for each**
  /* update the best solution if it is improved */
  **if** $\exists k : \#visited\_customers(L_k) > \#visited\_customers(L_{ACS-VEI})$ **then**
    $L_{ACS-VEI} \leftarrow L_k$
    $\forall\, j : IN_j \leftarrow 0$
    **if** $L_{ACS-VEI}$ is feasible **then**
      **send** $L_{ACS-VEI}$ to MACS-VRPTW
  **end if**
  /* perform global updating according to Equations 2.5 */
  $\tau_{ij} \leftarrow (1-\alpha) \cdot \tau_{ij} + \alpha \cdot \frac{1}{L_{ACS-VEI}} \qquad \forall (i,j) \in L_{ACS-VEI}$
  $\tau_{ij} \leftarrow (1-\alpha) \cdot \tau_{ij} + \alpha \cdot \frac{1}{L_{gb}} \qquad \forall (i,j) \in L_{gb}$
**until** a stopping criterion is met

problem closer to the traditional traveling salesman problem.

So, both in the TSP and in this model a feasible solution is a path that visits all the nodes exactly once. Figure 3.2 shows a vehicle routing problem solution represented as a single tour. Duplicated depots $(d_0, ..., d_3)$ are black points while clients are white points. All duplicated depots have the same coordinates but they have been split to clarify the picture. An advantage of such a solution representation is that the trails in direction of the duplicated depots are less attractive than in case of a single depot (due to the pheromone update rules). This positively affects the quality of the solutions produced by the constructive procedure.

### 3.3.2 Solution Constructive Procedure

ACS-VEI and ACS-TIME use the same *new_active_ant* constructive procedure that is presented in details in Algorithm 17. This constructive procedure is similar to the ACS constructive procedure designed for the TSP (Algorithm 12): Each artificial ant starts from a randomly chosen copy of the depot and, at each step, moves to a not yet visited node that does not violate time window constraints and vehicle capacities. The set of available nodes, in case the ant is not located in a duplicated depot, also includes not yet visited duplicated depots. An ant positioned in node $i$ chooses probabilistically the next node $j$ to be visited by using exploration and exploitation mechanisms (Equation 2.1). The attractiveness $\mu_{ij}$ is computed by taking into account the traveling time $t_{ij}$ between nodes $i$ and $j$, the time window $[b_j, e_j]$ associated to node $j$ and the number of times $IN_j$ node $j$ has not been inserted in a problem solution. When the *new_active_ant* is called by ACS-TIME, the variables $IN$ are not used and the corresponding parameter is set to zero.

Each time an ant moves from one node to another, a local update of the pheromone trail is executed according to Equation 2.10. Last, at the end of the constructive phase, the solution might be incomplete (some customers might have been omitted) and the solution is tentatively completed by performing further insertions. The insertion is executed by considering all the non visited customers sorted by decreasing delivery quantities. For each customer it is searched for the best feasible insertion (shortest travel time) until no further feasible insertion is possible.

In addition, ACS-TIME implements a local search procedure to improve the quality of the feasible solutions. The local search uses moves similar to CROSS exchanges of Taillard et al. (1997). This procedure is based on the exchange of two sub-chains of customers. One of this sub-chain may eventually be empty, implementing a more traditional customer insertion.

## 3.4 Computational Results

This section reports computational results showing the efficiency of MACS-VRPTW. MACS-VRPTW has been tested on a classical set of 56 benchmark problems (Solomon 1987) composed of six different problem types ($C1, C2, R1,\ R2, RC1, RC2$). Each data set contains between eight to

**Algorithm 17** $new\_active\_ant(k, local\_search, IN)$: Constructive Procedure for Ant $k$ Used by ACS-VEI and ACS-TIME

**Procedure** $new\_active\_ant(k, local\_search, IN)$
**Initialize**
/* put ant $k$ in a randomly selected duplicated depot $i$ */
$L_k \leftarrow < i >$
$current\_time_k \leftarrow 0, \quad load_k \leftarrow 0$
**repeat**
/* Starting from node $i$ compute the set $J_i^k$ of feasible nodes (i.e., all the nodes $j$ still to be visited and such that $current\_time_k$ and $load_k$ are compatible with time windows $[b_j, e_j]$ and delivery quantity $q_j$ of customer $j$) */
**for each** $j \in J_i^k$ compute the attractiveness $\mu_{ij}$ as follows: **do**
$delivery\_time_j \leftarrow \max(current\_time_k + t_{ij}, b_j)$
$delta\_time_{ij} \leftarrow delivery\_time_j - current\_time_k$
$distance_{ij} \leftarrow delta\_time_{ij} \cdot (e_j - current\_time_k)$
$distance_{ij} \leftarrow \max(1.0, distance_{ij} - IN_j)$
$\mu_{ij} \leftarrow \frac{1}{distance_{ij}}$
**end for each**
/* Choose probabilistically the next node $j$ using $\mu_{ij}$ in exploitation and exploration (Equation 2.1) */
$L_k \leftarrow L_k + < j >$
$current\_time_k \leftarrow delivery\_time_j$
$load_k \leftarrow load_k + q_j$
**if** $j$ is the depot **then** $current\_time_k \leftarrow 0, load_k \leftarrow 0$
**end if**
/* perform local updating according to Equation 2.10 */
$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0$
$i \leftarrow j$ /* next node for ant $k$ */
**until** $J_i^k = \{\}$
/* Path $L_k$ is extended by tentatively inserting non visited customers */
$L_k \leftarrow insertion\_procedure(L_k)$
/* Feasible paths are optimized by a local search procedure. The parameter $local\_search$ is $TRUE$ in ACS-TIME and it is $FALSE$ in ACS-VEI */
**if** $local\_search = TRUE$ **and** $L_k$ is feasible **then**
$L_k \leftarrow local\_search\_procedure(L_k)$
**end if**

twelve 100-node problems. The names of the six problem types have the following meaning. Sets C have clustered customers whose time windows were generated based on a known solution. Problem sets R have customers location generated uniformly randomly over a square. Sets *RC* have a combination of randomly placed and clustered customers. Sets of type 1 have narrow time windows and small vehicle capacity. Sets of type 2 have large time windows and large vehicle capacity. Therefore, the solutions of type 2 problems have very few routes and significantly more customers per route.

Experiments are made by executing, for each problem data, 3 runs that are stopped after a fixed computation time. Solutions are then averaged for each problem type and the result is reported in the tables. The code was written in C++.

Table 3.1 compares MACS-VRPTW with a number of the methods available for the VRPTW. The methods considered are: the adaptive memory programming (RT, Rochat & Taillard 1995), the large neighbourhood search (SW, Shaw 1998), the guided local search (KPS, Kilby et al. 1999), the alternate K-exchange Reduction (CW, Cordone & Calvo 2001) and the adaptive memory programming (TB, Taillard et al. 1997). Table 3.1 provides 3 columns for each data set: the average number of vehicles (main goal), the average tour length and the computation time (in seconds). The computational times cannot be directly compared for different reasons. First, the authors have used different computers; second, some methods (RT and TB) were designed to solve harder problems than the VRPTW and implementations specifically designed for the VRPTW might be faster.
MACS-VRPTW was executed on a Sun UltraSparc 1 167MHz, 70 Mflop/s, RT used a 15 Mflop/s Silicon Graphics computer, SW used a 63 Mflop/s Sun UltraSparc, KPS used a 25Mflops/s DEC Alpha, CW used a 18 Mflop/s Pentium and TB used a 10 Mflop/s Sun Sparc 10.

In Table 3.1 is shown that MACS-VRPTW is very competitive: for $C1$ and $RC2$ types it is clearly the best method and it is always among the best methods for the other problem sets. A characteristic of MACS-VRPTW is that it is able to produce relatively good solutions in a short amount of time.

Table 3.2 reports the average of the best solutions obtained in all our experiments. Similar results were also provided by other authors. In addition to the methods (RT, Rochat & Taillard 1995) and (TB, Taillard

Table 3.1: Performance comparison among VRPTW algorithms for different computational time (in seconds). RT=Rochat & Taillard (1995), SW=Shaw (1998), KPS=Kilby et al. (1999), CW=Cordone & Calvo (2001), TB=Taillard et al. (1997)

| | R1 | | | C1 | | | RC1 | | | R2 | | | C2 | | | RC2 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | VEI | DIST | TIME | VEI | DIST | TIME | VEI | DIST | TIME | VEI | DIST | TIME | VEI | DIST | TIME | VEI | DIST | TIME |
| | 12.55 | 1214.80 | 100 | 10.00 | 828.40 | 100 | 12.46 | 1395.47 | 100 | 3.05 | 971.97 | 100 | 3.00 | 593.19 | 100 | 3.38 | 1191.87 | 100 |
| MACS- | 12.45 | 1212.95 | 300 | 10.00 | 828.38 | 300 | 12.13 | 1389.15 | 300 | 3.00 | 969.09 | 300 | 3.00 | 592.97 | 300 | 3.33 | 1168.34 | 300 |
| VRPTW | 12.38 | 1213.35 | 600 | 10.00 | 828.38 | 600 | 12.08 | 1380.38 | 600 | 3.00 | 965.37 | 600 | 3.00 | 592.89 | 600 | 3.33 | 1163.08 | 600 |
| | 12.38 | 1211.64 | 1200 | 10.00 | 828.38 | 1200 | 11.96 | 1385.65 | 1200 | 3.00 | 962.07 | 1200 | 3.00 | 592.04 | 1200 | 3.33 | 1153.63 | 1200 |
| | 12.38 | 1210.83 | 1800 | 10.00 | 828.38 | 1800 | 11.92 | 1388.13 | 1800 | 3.00 | 960.31 | 1800 | 3.00 | 591.85 | 1800 | 3.33 | 1149.28 | 1800 |
| | 12.83 | 1208.43 | 450 | 10.00 | 832.59 | 540 | 12.75 | 1381.33 | 430 | 3.18 | 999.63 | 1600 | 3.00 | 595.38 | 1200 | 3.62 | 1207.37 | 1300 |
| RT | 12.58 | 1202.00 | 1300 | 10.00 | 829.01 | 1600 | 12.50 | 1368.03 | 1300 | 3.09 | 969.29 | 4900 | 3.00 | 590.32 | 3600 | 3.62 | 1155.47 | 3900 |
| | 12.58 | 1197.42 | 2700 | 10.00 | 828.45 | 3200 | 12.33 | 1269.48 | 2600 | 3.09 | 954.36 | 9800 | 3.00 | 590.32 | 7200 | 3.62 | 1139.79 | 7800 |
| | 12.45 | 1198.37 | 900 | | | | 12.05 | 1363.67 | 900 | | | | | | | | | |
| SW | 12.35 | 1201.47 | 1800 | | | | 12.00 | 1363.68 | 1800 | | | | | | | | | |
| | 12.33 | 1201.79 | 3600 | | | | 11.95 | 1364.17 | 3600 | | | | | | | | | |
| KPS | 12.67 | 1200.33 | 2900 | 10.00 | 830.75 | 2900 | 12.12 | 1388.15 | 2900 | 3 | 966.56 | 2900 | 3.00 | 592.29 | 2900 | 3.38 | 1133.42 | 2900 |
| CW | 12.50 | 1241.89 | 1382 | 10.00 | 834.05 | 649 | 12.38 | 1408.87 | 723 | 2.91 | 995.39 | 1332 | 3.00 | 591.78 | 292 | 3.38 | 1139.70 | 946 |
| | 12.64 | 1233.88 | 2296 | 10.00 | 830.41 | 2926 | 12.08 | 1404.59 | 1877 | 3.00 | 1046.56 | 3372 | 3.00 | 592.75 | 3275 | 3.38 | 1248.34 | 1933 |
| TB | 12.39 | 1230.48 | 6887 | 10.00 | 828.59 | 7315 | 12.00 | 1387.01 | 5632 | 3.00 | 1029.65 | 10116 | 3.00 | 591.14 | 8187 | 3.38 | 1220.28 | 5798 |
| | 12.33 | 1220.35 | 13774 | 10.00 | 828.45 | 14630 | 11.90 | 1381.31 | 11264 | 3.00 | 1013.35 | 20232 | 3.00 | 590.91 | 16375 | 3.38 | 1198.63 | 11596 |

Table 3.2: Average of the best solutions computed by different VRPTW algorithms. Best results are in boldface. MA-TW=MACS-VRPTW, RT=Rochat & Taillard (1995), TB=Taillard et al. (1997), CR=Chiang & Russel (1993), PB=Potvin & Bengio (1996), TH=Thangiah et al. (1994)

| | R1 | | C1 | | RC1 | | R2 | | C2 | | RC2 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Vei | Dist | Vei | Dist | Vei | Dist | Vei | Dist | Vei | Dist | Vei | Dist |
| MA-TW | **12.00** | 1217.73 | **10.00** | **828.38** | 11.63 | 1382.42 | **2.73** | 967.75 | 3.00 | **589.86** | **3.25** | 1129.19 |
| RT | 12.25 | 1208.50 | **10.00** | **828.38** | 11.88 | 1377.39 | 2.91 | 961.72 | 3.00 | **589.86** | 3.38 | 1119.59 |
| TB | 12.17 | 1209.35 | **10.00** | **828.38** | **11.50** | 1389.22 | 2.82 | 980.27 | 3.00 | **589.86** | 3.38 | 1117.44 |
| CR | 12.42 | 1289.95 | 10.00 | 885.86 | 12.38 | 1455.82 | 2.91 | 1135.14 | 3.00 | 658.88 | 3.38 | 1361.14 |
| PB | 12.58 | 1296.80 | 10.00 | 838.01 | 12.13 | 1446.20 | 3.00 | 1117.70 | 3.00 | 589.3 | 3.38 | 1360.57 |
| TH | 12.33 | 1238.00 | 10.00 | 832.00 | 12.00 | 1284.00 | 3.00 | 1005.00 | 3.00 | 650.00 | 3.38 | 1229.00 |

et al. 1997) already compared in Table 3.1. Table 3.2 includes the results of the hybrid method (CR, Chiang & Russel 1993), the genetic algorithm (PB, Potvin & Bengio 1996) and the hybrid method (TH, Thangiah et al. 1994). With the exception of $RC1$ type problem, MACS-VRPTW has been able to produce the best results for all other problem types.

During this experimental campaign, the best solution known of a number of problem instances have been improved. The value of these new best solutions are reported in Table 3.3. In addition to the VRPTW instances, the ACS-TIME colony has been tested on CVRP instances. In Table 3.3 are also reported new best solution value for CVRP problem instances *tainnn* used in Rochat & Taillard (1995), where *nnn* stands for the number of customers.

Table 3.3: New best solution values computed by MACS-VRPTW. RT=Rochat & Taillard (1995), S=Shaw (1998), TB=Taillard et al. (1997)

| | | Old | | New | |
|---|---|---|---|---|---|
| Problem | source | vehicles | length | vehicles | length |
| r112.dat | RT | 10 | 953.63 | **9** | 982.140 |
| r201.dat | S | 4 | 1254.09 | 4 | **1253.234** |
| r202.dat | TB | 3 | 1214.28 | 3 | **1202.529** |
| r204.dat | S | 2 | 867.33 | 2 | **856.364** |
| r207.dat | RT | 3 | 814.78 | **2** | 894.889 |
| r208.dat | RT | 2 | 738.6 | 2 | **726.823** |
| r209.dat | S | 3 | 923.96 | 3 | **921.659** |
| r210.dat | S | 3 | 963.37 | 3 | **958.241** |
| rc202.dat | S | 4 | 1162.8 | **3** | 1377.089 |
| rc203.dat | S | 3 | 1068.07 | 3 | **1062.301** |
| rc204.dat | S | 3 | 803.9 | 3 | **798.464** |
| rc207.dat | S | 3 | 1075.25 | 3 | **1068.855** |
| rc208.dat | RT | 3 | 833.97 | 3 | **833.401** |
| | | | | | |
| tai100a.dat | RT | 11 | 2047.90 | 11 | **2041.336** |
| tai100c.dat | RT | 11 | 1406.86 | 11 | **1406.202** |
| tai100d.dat | RT | 11 | 1581.25 | 11 | **1581.244** |
| tai150b.dat | RT | 14 | 2727.77 | 14 | **2656.474** |

## 3.5 Conclusions

This chapter described MACS-VRPTW, an Ant Colony Optimization based approach to solve vehicle routing problems with time windows. In particular, MACS-VRPTW has been designed to solve vehicle routing problems with two objectives: (i) the minimization of the number of tours (or vehicles) and (ii) the minimization of the total travel time, where number of tours minimization takes precedence over travel time minimization. MACS-VRPTW introduces a new methodology for optimizing multiple objective functions. The basic idea is to coordinate the activity of different ant colonies, each of them optimizing a different objective. These colonies work by using independent pheromone trails but they collaborate by exchanging information. MACS-VRPTW is the first ant colony optimization algorithm that adopts multiple colonies to solve a multi-objective optimization problem.

MACS-VRPTW couples ACS with a dedicated local search is able to solve vehicle routing problems with time windows. MACS-VRPTW has been shown to be competitive with the other effective methods both in terms of solution quality and computation time.

# Chapter 4

# HAS-SOP: An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem

In previous chapters we have shown that by coupling ACS with an extended version of local search procedure it is possible to obtain high-quality solutions for symmetric and asymmetric TSPs (Chapter 2, Dorigo & Gambardella 1997), as well as for vehicle routing problems (Chapter 3, Gambardella et al. 1999). In Gambardella et al. (1999) we have defined HAS-QAP, an ant colony algorithm coupled with a simple form of local search, to solve the quadratic assignment problem (QAP). HAS-QAP has been able to produce better solutions on structured, real-world problem instances than reactive tabu search (Battiti & Tecchiolli 1994b), robust tabu search (Taillard 1991), simulated annealing (Connolly 1990), and genetic hybrid search (Fleurent & Ferland 1994).

In this chapter we attack the sequential ordering problem (SOP) by an ACS algorithm coupled with SOP-3-exchange, a local search procedure specifically designed for the SOP that extends a TSP heuristic to directly handle SOP multiple constraints without increasing computational complexity. The resulting hybrid ant system for the SOP (HAS-SOP, Gambardella & Dorigo 2000) has been able to outperform all known heuristic SOP approaches. Also, we have been able to improve many of the best published results using the SOP-3-exchange with our ant colony opti-

mization system or in combination with the algorithm MPO/AI (Chen & Smith 1996).

This chapter is organized as follows: First in Section 4.1, the sequential ordering problem is introduced by presenting its formal definition and the main resolution approaches. Second, Section 4.2 extends ACS to deal with SOP and the resulting HAS-SOP is investigated by presenting its main components including the SOP-3-Exchange local search (Section 4.3). Last in Section 4.4, numerical results are reported and some conclusions are drawn.

## 4.1 The Sequential Ordering Problem

The sequential ordering problem with precedence constraints (SOP) was first formulated by Escudero (1988) to design heuristics for a production planning system. It consists of finding a minimum weight Hamiltonian path on a directed graph with weights on the arcs and the nodes, subject to precedence constraints among nodes.

### 4.1.1 Problem Definition

Consider a complete graph $G = (V, A)$ with node set $V$ and arc set $A$, where nodes correspond to jobs $0, ..., i, ..., n$ ($n+1 = |V|$). A cost $t_{ij} \in \Re$ with $t_{ij} \geq 0$, is associated to each arc $(i, j)$. This cost represents the waiting time between the end of job $i$ and the beginning of job $j$. A cost $p_i \in \Re$ with $p_i \geq 0$, representing the processing time of job $i$, is associated with each node $i$. The set of nodes $V$ includes a starting node (node 0) and a final node (node $n$) connected with all the other nodes. The costs between node 0 and the other nodes are equal to the setup time of node $i$, $t_{ij} = p_i \ \forall i$, and $t_{ij} = 0 \ \forall i$. Precedence constraints are given by an additional acyclic digraph $P = (V, R)$ defined on the same node set $V$. An arc $(i, j) \in R$ if job $i$ has to precede job $j$ in any feasible solution. $i$ has the transitive property (that is, if $(i, j) \in R$ and $(j, k) \in R$ then $(i, k) \in R$). Since a sequence always starts at node 0 and ends at node $n$, $(0, i) \in R \ \forall i \in V \setminus \{0\}$, and $(i, n) \in R \ \forall i \in V \setminus \{n\}$. In the following we will indicate with predecessor$[i]$ and successor$[i]$ the sets of nodes that have to precede/succeed node $i$ in any feasible solution.

Given the above definitions, the SOP can be stated as the problem of finding a job sequence that minimizes the total makespan subject to

the precedence constraints. This is therefore equivalent to the problem of finding a feasible Hamiltonian path with minimal cost in $G$ under precedence constraints given by $P$.

The SOP can also be formulated as a general case of the asymmetric traveling salesman problem (ATSP) by giving only the weights on the edges (in the SOP a solution connects the first and the last node by a path that visits all nodes once, as opposed to the ATSP in which a solution is a closed tour that visits all nodes once). This formulation is equivalent to the previous: it suffices to remove weights from nodes and to redefine the weight $c_{ij}$ of arc $(i, j)$ by adding the weight $p_j$ of node $j$ to each $t_{ij}$. In this representation $c_{ij}$ is an arc weight (where $c_{ij}$ may be different from $c_{ji}$), which can either represent the cost of arc $(i, j)$ when $c_{ij} \geq 0$, or an ordering constraint when $c_{ij} = -1$ ($c_{ij} = -1$ means that element $j$ must precede, not necessarily immediately, element $i$). In this chapter we will use this last formulation.

### 4.1.2 Heuristic Methods for the SOP

The SOP models real-world problems such as production planning (Escudero 1988), single vehicle routing problems with pick-up and delivery constraints (Pulleyblank & Timlin 1991; Savelsbergh 1990), transportation problems in flexible manufacturing systems (Ascheuer 1995).

The SOP can be seen as a general case of both the asymmetric TSP and the pick-up and delivery problem. It differs from ATSP because the first and the last nodes are fixed, and in the additional set of precedence constraints on the order in which nodes must be visited. It differs from the pick-up and delivery problem because this is usually based on symmetric TSPs, and because the pick-up and delivery problem includes a set of constraints between nodes with a unique predecessor defined for each node, in contrast to the SOP where multiple precedences can be defined.

### 4.1.3 Approaches Based on the ATSP

Sequential ordering problems were initially solved as constrained versions of the ATSP. The main effort has been put into extending the mathematical definition of the ATSP by introducing new equations to model the additional constraints. The first mathematical model for the SOP was introduced in Ascheuer et al. (1993) where a cutting-plane approach was

proposed to compute lower bounds on the optimal solution. In Escudero et al. (1994) a Lagrangian relax-and-cut method was described and new valid cuts to obtain strong lower bounds were defined. In addition, Ascheuer (1995) has proposed a new class of valid inequalities and has described a branch-and-cut algorithm for a broad class of SOP instances based on the polyhedral investigation carried out on ATSP problems with precedence constraints by Balas et al. (1995). His approach also investigates the possibility to compute and improve sub-optimal feasible problem solutions starting from the upper bound computed by the polyhedral investigation. The upper bound is the initial solution of a heuristic phase based on well-known ATSP heuristics that are iteratively applied in order to improve feasible solutions. These heuristics do not handle constraints directly: infeasible solutions are simply rejected. With this approach Ascheuer was able to compute new upper bounds for the SOP instances in TSPLIB, although a genetic algorithm called Maximum Partial Order/Arbitrary Insertion (MPO/AI), proposed by Chen & Smith (1996), seems to work better on the same class of problems. MPO/AI always works in the space of feasible solutions by introducing a sophisticated crossover operator that preserves the common schema of two parents by identifying their maximum partial order through matrix operations. The new solution is completed using a constructive heuristic.

### 4.1.4 Approaches Based on the Pick-up and Delivery Problem

Heuristic approaches to pick-up and delivery problems are based on particular extensions of TSP heuristics able to handle precedence constraints while improving feasible solutions without any increase in computation times. Psaraftis (1983) has introduced a preprocessing technique to ensure feasibility checking in constant time by starting the algorithm with a screening procedure that, at an initial cost of $O(n^2)$, produces a feasibility matrix that contains information about feasible edge exchanges. Subsequently, Solomon (1987) proposed a search procedure based on a tailored updating mechanism, while Savelsbergh (1990), Van der Bruggen et al. (1993), and Kindervater & Savelsbergh (1997) presented a lexicographic search strategy, a variation of traditional edge-exchange TSP heuristics, that reduces the number of visited nodes without losing any feasible exchange. In order to ensure constraint checking in constant time, the lexicographic search strategy has been combined with a *labeling procedure* where nodes in the sequence are labeled with information related to their

unique predecessor/successor, and a set of global variables are updated to keep this information valid. Savelsbergh (1990) presented a lexicographic search based on 2-opt and 3-opt strategies that exchanges a fixed number of edges, while Van der Bruggen et al. (1993) proposed a variable-depth search based on the Lin & Kernighan (1973) approach. Unfortunately, this *labeling procedure* is not applicable in the case of multiple precedence constraints because it requires that nodes in the sequence have a unique predecessor/successor. On the other hand, the lexicographic search strategy itself is independent of the number of precedence constraints and can therefore be used to solve sequential ordering problems where multiple precedence constraints are allowed.

The approach to the SOP presented in this chapter is the first in the literature that uses an extension of a TSP heuristic to handle directly multiple constraints without any increase in computational time. Our approach combines a constructive phase based on the ACS algorithm (Chapter 2.3) with a new local search procedure called SOP-3-exchange. SOP-3-exchange is based on a lexicographic search heuristic due to Savelsbergh (1990) and a new *labeling procedure* able to handle multiple precedence constraints. In addition, we test and compare different methods to select nodes during the search and different stopping criteria. In particular we test two different selection heuristics: one based on the *don't look bit* data structure introduced by Bentley (1992), and the other based on a new data structure called *don't push stack* introduced by the author of this thesis.

## 4.2 ACS for the Sequential Ordering Problem

The application of an ACO algorithm to a combinatorial optimization problem requires definition of a constructive algorithm and possibly a local search (Section 1.4.4). Accordingly, the ACO metaheuristic can be adapted to the SOP by letting ants build a path from source to destination while respecting the ordering constraints (this can be achieved by having ants choose not-yet-visited nodes that do not violate any ordering precedence). Therefore we have designed a constructive algorithm called ACS-SOP in which a set of artificial ants builds feasible solutions to the SOP.

ACS-SOP is strongly based on Ant Colony System (Chapter 2, Algorithm 12, Gambardella & Dorigo 1996; Dorigo & Gambardella 1997) and

it differs from the original ACS in the way the set of feasible nodes is computed and in the setting of one of the algorithm's parameters that is made dependent on the problem dimensions.
ACS-SOP implements the constructive phase of HAS-SOP, and its goal is to build feasible solutions for the SOP. Informally, ACS-SOP works as follows. Each ant iteratively starts from node 0 and adds new nodes until all nodes have been visited and node $n$ is reached. When in node $i$, an ant applies a so-called transition rule, that is, it probabilistically chooses the next node $j$ from the set $F(i)$ of feasible nodes. $F(i)$ contains all the nodes $j$ still to be visited and such that all nodes that have to precede $j$, according to precedence constraints, have already been inserted in the sequence. As in ACS the ant chooses (Equations 2.4, 2.1), with probability $q_0$, the node $j$ with a deterministic rule (exploitation) while with probability $(1-q_0)$ the node $j$ is chosen in a probabilistic way (exploration). The value $q_0$ in ACS-SOP is given by $q_0 = 1 - s/n$; $q_0$ is based on a parameter $s$ that represents the number of nodes we would like to choose using the probabilistic transition rule. The parameter $s$ allows the system to define $q_0$ independently of the problem size, so that the expected number of nodes selected with the probabilistic rule is $s$.

As in ACS also in ACS-SOP only the best ant, that is the ant that built the shortest tour, is allowed to deposit pheromone trail. The rationale is that in this way a preferred route is memorized in the pheromone trail matrix $\tau$ and future ants will use this information to generate new solutions in a neighborhood of this preferred route. The formula used to update the pheromone trail is Equation 2.5 where $L_{gb}$ is the length of the path built by the best ant, that is, the length of the shortest path generated since the beginning of the computation.

Pheromone is also updated during solution building. In this case, however, it is removed from visited edges. In other words, each ant, when moving from node $i$ to node $j$, applies a pheromone updating (Equation 2.10) rule that causes the amount of pheromone trail on edge $(i, j)$ to decrease. Also in case of ACS-SOP we found that good values for the algorithm's parameters are $\tau_0 = (n \cdot L_{nn}^{-1})$, $\alpha = \rho = 0.1$, $s = 10$, where $L_{nn}$ is the length of the shortest solution generated by the ant colony following the ACS-SOP algorithm without using the pheromone trails. These values are rather robust: values in the following ranges didn't cause any appreciable change in performance: $0.05 \leq \alpha, \rho \leq 0.3, 5 \leq s \leq 15$. The number of ants in the population was set to 10. The rationale for

using Equation 2.10 is that it causes ants to eat away pheromone trail while they build solutions so that a certain variety in generated solutions is assured (if pheromone trail was not consumed by ants they would tend to generate very similar tours).

The algorithm stops when one of the following conditions becomes true: a fixed number of solutions has been generated; a fixed CPU time has elapsed; no improvement has been observed during a fixed last number of iterations.

### 4.2.1 HAS-SOP. ACS-SOP Coupled with SOP-3-exchange Local Search

HAS-SOP is ACS-SOP plus local search following the Algorithm schema of Algorithm 13. Local search is an optional component of ACO algorithms, although it has been shown since early implementations that it can greatly improve the overall performance of the ACO metaheuristic when static combinatorial optimization problems are considered In HAS-SOP local search is applied once ants have built their solutions: each solution is carried to its local optimum by an application of the local search routine called SOP-3-exchange. Locally optimal solutions are then used to update pheromone trails on arcs, according to the pheromone trail update rule of ACS (Equation 2.5). SOP-3-exchange is a very efficient local search procedure. SOP-3-exchange has been explicitly designed to solve SOP problems starting from the work of Savelsbergh (1990) on the asymmetric travelling salesman problem. SOP-3-exchange is capable to handle multiple precedence constraints without increasing the computational complexity of original local search.

## 4.3 Locas Search: SOP-3-Exchange

Much research went into defining ad-hoc TSP heuristics, and in particular edge-exchange improvement heuristics (see Johnson & McGeoch (1997) and Sections 1.2, 1.3 for an overview).

Starting from an initial solution, an edge-exchange procedure generates a new solution by replacing $k$ edges with another set of $k$ edges. This operation is usually called a *k-exchange* and is iteratively executed until no additional improving *k-exchange* is possible. When this is the case the final solution is said to be *k-optimal*; the verification of k-optimality

requires $O(n^k)$ time. For a *k-exchange* procedure to be efficient it is necessary that the improving criterion for new solutions can be computed in constant time.

In this section we first make some observations about *edge-exchange* techniques for TSP/ATSP problems. Then, we concentrate our attention on *path-preserving-edge-exchanges* for ATSPs, that is, edge exchanges that do not invert the order in which paths are visited. Next, we discuss *lexicographic-path-preserving-edge-exchange*, a *path-preserving-edge-exchange* procedure that se-arches only in the space of feasible exchanges. We then add to the *lexicographic-path-preserving-edge-exchange* a *labeling procedure* whose function is to check feasibility in constant time. Finally, we present different possible strategies to select nodes during the search, as well as different search stopping criteria.

### 4.3.1 Path-Preserving Edge-Exchange Heuristics

We remind the reader that the SOP can be formulated as a general case of the asymmetric traveling salesman problem (ATSP) in which a solution connects the first and the last node by a path that visits all nodes once, as opposed to the ATSP in which a solution is a closed tour that visits all nodes once. *Edge-exchange* techniques for TSP/ATSP problems are therefore directly relevant for the SOP. A *k-exchange* deletes $k$ edges from the initial solution creating $k$ disjointed paths that are reconnected with $k$ new edges. In some situations this operation requires an inversion in the order in which nodes are visited within one of the paths (*path-inverting-edge-exchange*, while in other situations this inversion is not required (*path-preserving-edge-exchange*).

Consider a *2-exchange* (Figure 4.1) where two edges to be removed, $(h, h+1)$ and $(i, i+1)$, have been selected. In this situation there are only two ways to perform the exchange: in the first case (Figure 4.1b) edges $(h, i)$ and $(h+1, i+1)$ are inserted and the traveling direction for path $\langle i, ..., h+1\rangle$ is inverted; in the second case (Figure 4.1c) edges $(i, h)$ and $(i+1, h+1)$ are inserted inverting the traveling direction for path $\langle h, ..., i+1\rangle$.

In the case of a *3-exchange*, however, there are several possibilities to build a new solution when edges $(h, h+1)$, $(i, i+1)$, and $(j, j+1)$ are selected to be removed (Figure 4.2). In Figure 4.2 a path preserving *3-exchange* (Figure 4.2b) and a path inverting 3-exchange (Figure 4.2c) are shown. It is then clear that any *2-exchange* procedure determines

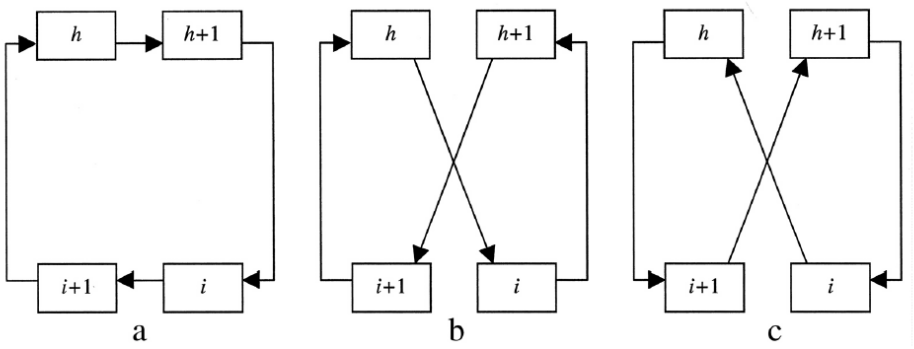


Figure 4.1: A 2-exchange always inverts a path

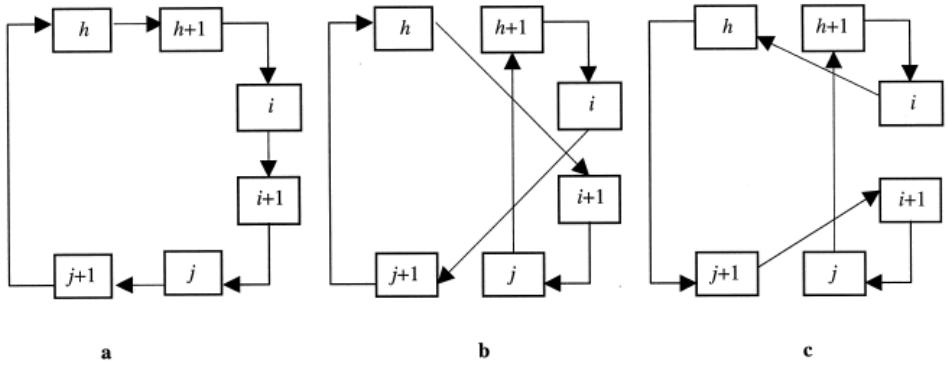


Figure 4.2: A 3-exchange without (b) and with (c) path inversion

the inversion of one of the involved paths, while for $k = 3$ this inversion is caused only by particular choices of the inserted edges.

In the case of TSP problems, where arc costs $d_{ij} = d_{ji}\ \forall(i, j)$, inverting a path does not modify its length. Therefore, the quality of the new solution depends only on the length of the inserted and deleted edges. On the other hand, for ATSP problems, where $d_{ij} \neq d_{ji}$ for at least one $(i, j)$, inverting a path can modify the length of the path itself, and therefore the length of the new solution does not depend only on the inserted and deleted edges. This situation contrasts with the requirement that the improving criterion be verifiable in constant time. Therefore, the only suitable edge-exchange procedures for sequential ordering problems, which are a constrained version of ATSP problems, are *path-*

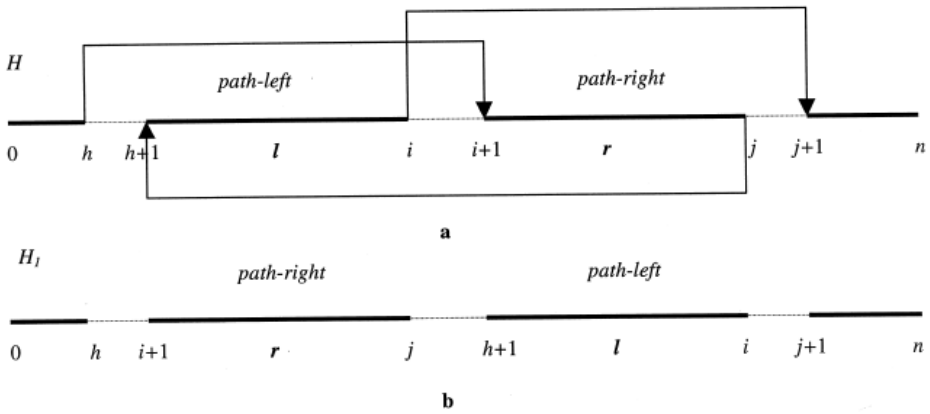


Figure 4.3: A path-preserving-3-exchange

*preserving-edge-exchange* heuristics. In the following, we concentrate on *path-preserving-k-exchange*, *pp-k-exchange* for short, with $k = 3$, that is, the smallest $k$ that allows a path preserving edge exchange.

Starting from a feasible SOP sequence $H$, a *pp-3-exchange* tries to reduce the length of $H$ by replacing edges $(h, h+1), (i, i+1)$ and $(j, j+1)$ with edges $(h, i+1), (i, j+1)$ and $(j, h+1)$ (Figure 4.3a). The result of a *pp-3-exchange* is a new sequence $H1$ (Figure 4.3b) where, while walking from node 0 to node $n$, the order we visit *path_left*$=\langle h+1, ..., i\rangle$ and *path_right*$=\langle i+1, ..., j\rangle$ is swapped. In this situation, the new sequence $H1$ is feasible only if in the initial solution $H$ there were no precedence constraints between a generic node $l \in$ *path_left* and a generic node $r \in$ *path_right*.

Given two generic paths *path_left* and *path_right*, to test the feasibility of the *pp-3-exchange* requires computational effort of order $O(n^2)$ (precedence constraints must be checked between each pair of nodes in the two paths).

Savelsbergh (1990) studied how to limit the computational effort needed for checking solution feasibility in the case of precedence constraints for dial-a-ride problems. He introduced a particular exploration strategy called lexicographic search strategy that allows for generating and exploring only feasible exchanges. Savelsbergh presents a combination of the lexicographic search strategy with a *labeling procedure* where a set of global variables is updated so that precedence-constraint checking can be performed in constant time.

The lexicographic search strategy was introduced to solve dial-a-ride problems where only one precedence constraint for each node is allowed. Nevertheless, it is independent of the number of constraints. We have applied a version of Savelsbergh's lexicographic search strategy restricted to the case $k = 3$, *lpp-3-exchange*, to sequential ordering problems with multiple constraints for each node.

However, Savelsbergh's *labeling procedure* was designed to handle unique precedence constraints under particular search conditions and cannot be extended to sequential ordering problems. Before explaining our new *labeling procedure* for the SOP, we present the *lpp-3-exchange*.

### 4.3.2 Lexicographic Search Strategy in the Case of Precedence Constraints

The *lpp-3-exchange* procedure identifies two paths, *path_left* and *path_right*, which once swapped give rise to a new feasible solution. These two paths are initially composed of one single node and are incrementally expanded, adding one node at each step. This feature makes it possible to test feasibility easily because precedence conditions must be checked only for the new added node.

To explain how an *lpp-3-exchange* works let us consider a feasible solution H in which nodes are ordered from 0 to $n$. Then we consider three indexes $h, i$, and $j$, that point to nodes in the sequence. As explained below, *lpp-3-exchange* is composed of two procedures that differ in the order nodes in the sequence $H$ are explored. We start by explaining the *forward-lpp-3-exchange*, *f-lpp-3-exchange* for short.

The *f-lpp-3-exchange* procedure starts by setting the value of $h$ to 0 (that is, $h$ points to node 0 in the sequence $H$). Then it sets the value of $i$, which identifies the rightmost node of *path_left*, to $h+1$, and performs a loop on the value of $j$, which identifies the rightmost node of *path_right* (Figures 4.4a, 4.4b). In other words, *path_right*=$\langle i+1, ..., j\rangle$ is iteratively expanded by adding new edges $(j, j+1)$. Once all available nodes have been added to *path_right* (that is, until a precedence constraint is violated or when $j+1$ points to node $n$, (see Figure 4.4b), *path_left* is expanded by adding the new edge $(i, i+1)$ (Figure 4.4c), and then *path_right* is searched again. *Path_left* $\langle h+1, ..., i\rangle$ is expanded until $i+1$ points to node $n-1$. Then $h$ is set to $h+1$ and the process is repeated. The *f-lpp-3-exchange* procedure stops when $h$ points to node $n-2$.

As we said, *f-lpp-3-exchange* considers only forward exchanges, that is,

exchanges obtained considering indexes $i$ and $j$ such that $j > i > h$. The *backward-lpp-3-exchange* procedure, *b-lpp-3-exchange* for short, considers backward exchanges, that is, exchanges obtained considering indexes $j$ and $i$ such that $j < i < h$ (with $2 \leq h < n$). In *b-lpp-3-exchange* (Figure 4.5) *path_left* is identified by $\langle j+1, ..., i\rangle$ and *path_right* by $\langle i+1, ..., h\rangle$. After fixing $h$, $i$ is set to $h-1$ and $j$ to $i-1$ (Figure 4.5a). Then *path_left* is expanded backward (Figure 4.5b) moving $j$ till the beginning of the sequence, that is, iteratively setting $j$ to the values $i$ -2, $i$ -3, ..., 0 (i.e., each backward expansion adds a new node to the left of the path: $\langle j+1, ..., i\rangle$ is expanded to $\langle j, j+1, ..., i\rangle$). Then, *path_right* is iteratively expanded in a backward direction with the new edge $(i, i+1)$, and the loop on *path_left* is repeated.

The complete SOP-3-exchange procedure performs a forward and a backward lexicographic search for each value $h$, visiting in this way all the possible nodes in the sequence (just like any other *3-exchange* procedure).

The important point is that the method for defining *path_left* and *path_right* permits an easy solution of the feasibility-checking problem: the search is restricted to feasible exchanges only, since it can be stopped as soon as an infeasible exchange is found. Consider for example an *f-lpp-3-exchange*: once *path_left*=$\langle h+1, ..., i\rangle$ has been fixed, we set *path_right* to $j = i+1$. In this situation it is possible to check exchange feasibility by testing whether there is a precedence relation between node $j$ and nodes in *path_left*. Before expanding *path_right* with the new edge $(j, j+1)$ we check whether the resulting paths are still feasible by testing again the precedence relations between the new node $j+1$ and nodes in *path_left*. If the test is not feasible we stop the search. In fact, any further expansion of $j+1$ in $\langle j+2, j+3, ..., n\rangle$ will always generate an infeasible exchange because it still violates at least the precedence constraint between $j+1$ and *path_left*.

Note that expanding *path_left* with edge $(i, i+1)$ does not induce any precedence constraint violations because the order of nodes inside *path_left* is not modified and the search for a profitable *f-lpp-3-exchange* always starts by setting *path_right* equal to element $j = i+1$.

Without considering any additional *labeling procedure*, the feasibility test in this situation has a computational cost of $O(n)$: each time a new $j$ is selected we test if there is a precedence relation between $j$ and the nodes in *path_left*. In the case of the SOP this test should check whether $c_{jl} \neq -1 \ \forall \ l$ in *path_left* (recall that for sequential ordering problems

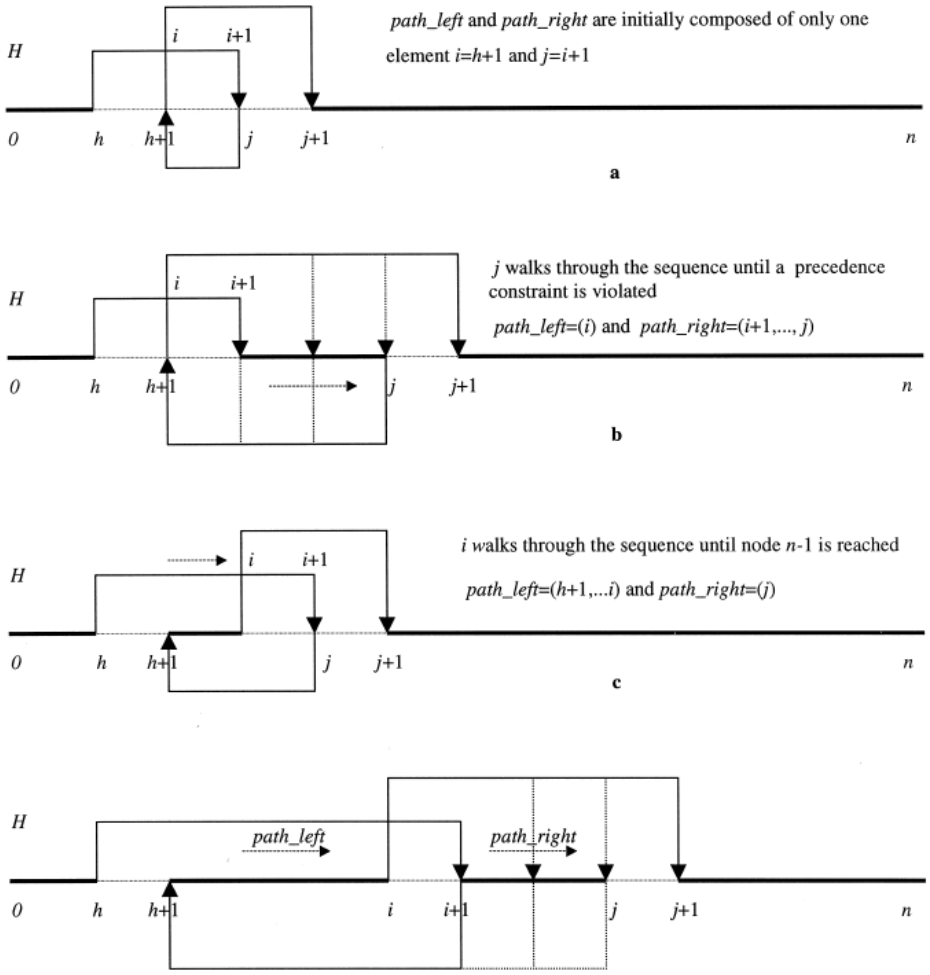


Figure 4.4: Lexicographic forward path-preserving-3-exchange

$c_{jl} = -1$ if $l$ has to precede $j$), and in the final solution $H1$ the order in which we visit *path_left* and *path_right* is swapped and therefore $l$ will follow $j$, (Figure 4.3b). Similar considerations should be made in the case of *b-lpp-3-exchange* where the feasibility test checks if $c_{r,j+1} \neq -1$ $\forall r \in$ *path_right*.

The previous complete lexicographic search procedure requires a check of all predecessors/successors of node $j$. This procedure increases the computational effort to check *3-optimality* from $O(n^3)$ to $O(n^4)$. In order

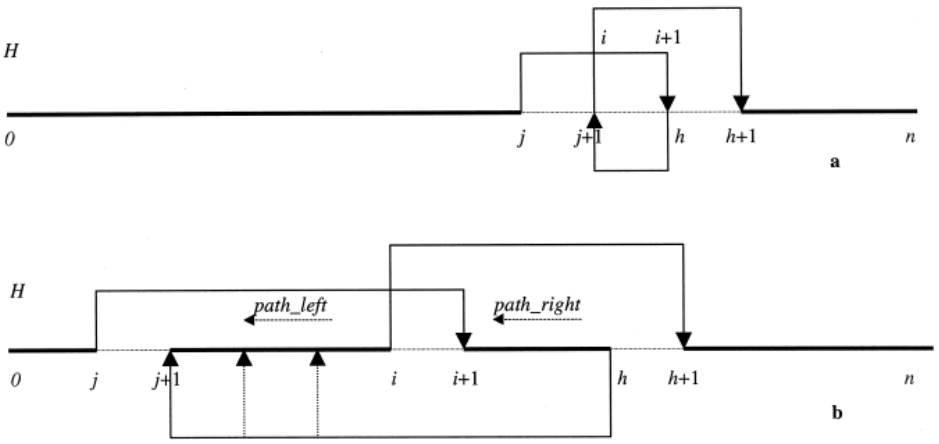


Figure 4.5: Lexicographic backward path-preserving-3-exchange

to keep the cost at $O(n^3)$ we introduce the SOP *labeling procedure* to handle multiple constraints.

### 4.3.3 The SOP *Labeling Procedure*

The SOP *labeling procedure* is used to mark nodes in the sequence with a label that allows for feasibility checking for each selected $j$ in constant time. The basic idea is to associate with each node a label that indicates, given *path_left* and *path_right*, whether or not it is feasible to expand *path_right* with the following node $j + 1$.

We have implemented and tested different SOP *labeling procedure*s that set and update nodes in different phases of the search. In the following, we will present a combination of the best-performing SOP *labeling procedure* with the *lexicographic search* strategy, with different selection criteria for node h and with different search-stopping criteria.

Our SOP *labeling procedure* is based on a set of global variables that are updated during the *lexicographic search* procedure. As in the previous subsection, we will distinguish between forward and backward search. First we introduce a global variable *count_h* that is set to 0 at the beginning of the search, and which is increased by 1 each time a new node $h$ is selected. Second, we associate a global variable *f_mark(v)* to each node $v \in H$ in the case of *f-lpp-3-exchange*, and a global variable *b_mark(v)* in the case of *b-lpp-3-exchange*. These global variables are initially set

to 0 for each $v$. An *f-lpp-3-exchange* starts by fixing $h$, $i = h + 1$, and *path_left*=$\langle i \rangle$. At this point, for all nodes $s \in$ successor$[i]$ we set *f_mark(s)=count_h*. We repeat this operation each time *path_left* is expanded with a new node $i$. Therefore the *labeling procedure* marks with the value *count_h* all the nodes in the sequence that must follow one of the nodes belonging to *path_left*. When *path_right* is expanded moving $j$ in $\langle i+2, ..., n \rangle$ if *f_mark(j)* =*count_h* we stop the search because the label indicates that $j$ must follow a node in *path_left*. At this point, if no other search-termination condition is met, the procedure restarts expanding again *path_left*. In this situation all the previous defined labels remain valid and the search continues by labeling all the successors of the new node $i$. On the other hand, when we move $h$ forward into the sequence we invalidate all previously set labels by setting *count_h=count_h*+1. The same type of reasoning holds for *b-lpp-3-exchange*. Each time node $i$ is selected we identify a new *path_right*=$\langle i + 1, ..., h \rangle$ and for all nodes $s \in$ predecessor$[i+1]$ we set *b_mark(s)=count_h*. When expanding *path_left* by iteratively adding a new edge $(j, j + 1)$, the expansion is not accepted if *b_mark(j)=count_h*.

### 4.3.4 Heuristics for the Selection of Node $h$ and Search Stopping Criteria

This sequential search procedure for sequential ordering problems is a general description of how the lexicographic search works in combination with the SOP *labeling procedure*. Although the SOP *labeling procedure* reduces the complexity of the lexicographic search to $O(n^3)$, this is still too expensive from a practical point of view; in fact, the exploration of all the feasible exchanges is still required. There are different ways to reduce this effort: for example, heuristic criteria can be introduced to reduce the number of visited nodes, or the search can be stopped and the exchange executed as soon as some improving condition is met.

**Heuristic Selection of Node** $h$. In order to reduce the number of explored nodes, Savelsbergh (1990) and Van der Bruggen et al. (1993) proposed to use a particular type of *k-exchange* called OR-*exchange* (Or 1976) that limits the choice of $i$ among the three closest nodes of $h$. In practice, $i$ is selected among $(h + 1, h + 2, h + 3)$ in the case of a forward exchange, and among $(h - 1, h - 2, h - 3)$ in the case of a backward exchange. Alternatives decrease the number of visited nodes, introducing two heuristics that influence how node $h$ is chosen: one is based on the

*don't look bit* data structure introduced by Bentley (1992), while the other is based on a new data structure called *don't push stack* introduced by the author. The *don't look bit* is a data structure in which a bit is associated with each node of the sequence. At the beginning of the search all bits are turned off. The bit associated with node $h$ is turned on when a search for an improving move starts from node $h$. If a profitable exchange is executed the bit of the six nodes involved in the exchange (that is, $j+1, i+1, h+1, j, i, h$) are turned off. The use of *don't look bits* favors the exploration of nodes that have been involved in a profitable exchange. The search procedure visits all the nodes in the sequence, moving from the first node 0 to the last node $n$ but only nodes with the *don't look bit* turned off are taken into consideration as candidates for node $h$. The search procedure is repeatedly applied until all nodes have their *don't look bit* turned on. The *don't push stack* is a data structure based on a stack, which contains the set of nodes $h$ to be selected, associated with a particular push operation. At the beginning of the search the stack is initialized with all the nodes (that is, it contains $n+1$ elements). During the search, node $h$ is popped off the stack and feasible *3-exchange* moves starting from $h$ are investigated. If a profitable exchange is executed the six nodes involved in this exchange (that is, $j+1, i+1, h+1, j, i, h$) are pushed onto the stack (if they do not already belong to it). Using this heuristic, once a profitable exchange is executed starting from node $h$, the top node in the *don't push stack* remains node $h$. In addition, the maximum size of the stack is limited to $n+1$ elements. The use of the *don't push stack* gives the following benefits. First, the search is focused on the neighborhood of the most recent exchange: this has been experimentally shown to result in better performance than that obtained using the *don't look bit*. Second, the selection of node $h$ is not constrained to be a sequential walk through the sequence $H$. This is an important feature given the fact that the SOP *labeling procedure* is designed to work with independent and random choices of $h$, where independent means that the choice of the new $h$ is not constrained to be the choice of the old $h$. In fact, it does not require, as is the case of Savelsbergh's *labeling procedure* (Savelsbergh 1990), retention of valid labeling information while walking through the sequence from one $h$ to the next: in our case a new labeling is started as soon as a new $h$ is chosen; this allows for selecting $h$ in any sequence position without introducing additional computational costs.

**Stopping Criteria**. The number of visited nodes can be decreased by stopping the search once an improving exchange is found. In our experiments we have tested three different stopping conditions: (ExchangeFirstCriterion= $h, i, j$) stops the search as soon as the first feasible exchange is found in the $h$, $i$ or $j$ loops respectively. We have also tested the standard exploration strategy where the most profitable exchange is selected among all the possible exchanges, but this method is not presented here because the results obtained are much worse given the same amount of computation time.

### 4.3.5 The SOP-3-Exchange Procedure: An Example

In this section we discuss briefly the local search procedure using a simple example. Algorithm 18 presents the pseudo-code of the SOP-3-exchange procedure with all the possible options. The example we consider is the following: Let a sequence $\langle 0, a, b, c, d, e, f, g, n\rangle$ represent a feasible solution of a SOP in which node e is constrained to follow a in any feasible solution. The forward SOP-3-exchange procedure works as follows. Initially, $h$ is set to point to node 0 (i.e., $h = 0$), variable *count_h* is set to zero, direction is set to forward, $i$ is set to $a$ and $j$ to $b$. In this state of the computation *path_left* (from node $h + 1$ to node $i$) and *path_right* (from node $i + 1$ to node $j$) consist of the sequences $\langle a\rangle$ and $\langle b\rangle$ respectively. In the following the notation $[\langle a\rangle\langle b\rangle]$ will be used to indicate the pair *path_left* and *path_right*.

Inside the $i$ loop the successor of node $a$, node $e$, is labeled by setting *f_mark(e)* = *count_h* = 0. In the $j$ loop, *path_right* is expanded by adding nodes of the sequence until either the end of the sequence is reached or a precedence constraint is violated. The first expansions are $[\langle a\rangle\langle b, c\rangle]$ and $[\langle a\rangle\langle b, c, d\rangle]$. At this point, *path_right* should not be extended to $\langle b, c, d, e\rangle$ because $e$ is labeled with a value equal to *count_h*. In fact, the new sequence generated by using $[\langle a\rangle\langle b, c, d, e\rangle]$ would be $\langle 0, b, c, d, e, a, f, g, n\rangle$ where node $a$ follows node $e$, in contrast with the precedence constraint. Therefore, the $j$ loop is terminated and the $i$ loop is resumed. Node $i$ is moved through the sequence by setting $i$ equal to node $b$ and the two paths are set to $[\langle a, b\rangle\langle c\rangle]$. Node $b$ does not have any successor node to label; therefore, the $j$ loop is executed again. Paths are expanded to $[\langle a, b\rangle\langle c, d\rangle]$ but, as before, they should not be extended to $[\langle a, b\rangle\langle c, d, e\rangle]$ due to the precedence constraint. The procedure continues generating the paths $[\langle a, b, c\rangle\langle d\rangle]$, while

**Algorithm 18** The SOP-3-exchange Procedure

```
Procedure SOP-3-exchange
/* input:
   a feasible solution given as a sequence (0.....n)
   a sequential ordering problem G
   a SelectionCriterion for h in (sequential, dont_look_bit, dont_push_stack)
   a WalkingCriterion for i in (3-exchange, OR-exchange)
   an ExchangeFirstCriterion in (h, i, j)
output:
   a new feasible solution that is 3-optimal */
repeat
   h ← 0 /* h is set to the first node in the sequence */
   while there is an available h /* h loop */ do
      /* Selects node h according to SelectionCriterion.
         In case SelectionCriterion=sequential, h is the next node in the sequence.
         In case SelectionCriterion=dont_look_bit, h is the next node in the sequence
         with dont_look_bit[h]=off.
         In case SelectionCriterion=dont_push_stack, h is popped from the stack */
      h ← SelectAccordingCriterion(SelectionCriterion,G);
      direction ← forward; /* the search starts in forward direction */
      gain ← 0; i ← h + 1; j ← i + 1; SearchTerminated ← false
      while SearchTerminated /* i loop */ do
         /* When i has reached the end of the sequence during a forward search we
         start a new search in backward direction starting from the same h. In case
         WalkingCriterion=OR-exchange the direction is inverted after three selections
         of i */
         feasible ← true
         if (direction=forward and EndOfSequence(i,WalkingCriterion,G)) then
            direction ← backward; i ← h − 1; j ← i − 1
         end-if
         /* in case direction=forward we update labeling information for successor[i]; in
         case direction=backward we update labeling information for predecessor[i+1] */

         UpdateGlobalVariables(h, i, direction, G)
         while feasible /* j loop */ do
            /* Using labeling information we test if the 3-exchange involving h,i,j is fea-
            sible */
            feasible ← FeasibleExchange(h, i, j, direction, G)
            /* Checks if the new 3-exchange is better then the previous one; if the case,
            saves it */
            gain ← ComputeBestExchange(h, i, j, direction, G, feasible, gain)
            if (gain >0 and ExchangeFirstCriterion=j) then goto EXCHANGE
            /* j is moved through the sequence according to direction: in case direc-
            tion=forward j ← j + 1, in case direction=backward j ← j − 1 */
            j ← jWalkThroughTheSequence(h, i, j, direction, G)
            SearchTerminated ← f(j, direction, WalkingCriterion)
         end while
         if (gain >0 and ExchangeFirstCriterion=i) then goto EXCHANGE
         /* i is moved through the sequence according to direction; in case
         direction=forward i ← i + 1, in case direction=backward i ← i − 1*/
         i ← iWalkThroughTheSequence(h, i, direction, G)
         SearchTerminated ← f(i, direction, WalkingCriterion)
      end while
      EXCHANGE
      if (gain > 0) then
         /* the best exchange is executed and the search starts again */
         PerformExchange(h, i, j, direction, G)
         goto REPEAT
      end if
   end while
until
```

the following paths $[\langle a, b, c\rangle\langle d, e\rangle]$ and $[\langle a, b, c, d\rangle\langle e\rangle]$ are not feasible because of the constraint between $a$ and $e$. The next feasible steps are $[\langle a, b, c, d, e\rangle\langle f\rangle]$,$[\langle a, b, c, d, e\rangle\langle f, g\rangle]$,$[\langle a, b, c, d, e, f\rangle\langle g\rangle]$ .

## 4.4 Computational Results

Our experiments were aimed at (i) finding the best parameters for the SOP-3-exchange procedure, (ii) comparing ACS-SOP and HAS-SOP with a set of competing methods over a significant set of test problems, and (iii) evaluating the relative contribution to overall performance of the SOP-3-exchange local search with respect to the constructive methods. The results obtained are presented and discussed in the following of the section. Experiments were run on a SUN Ultra1 SPARC Station (167Mhz). The code was written in C++. Before presenting and discussing the computational results we briefly describe the experimental setting.

### 4.4.1 Experimental Settings: Test Problems

We tested our algorithms on the set of problems available in the TSPLIB (http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/). Sequential ordering problems in TSPLIB can be classified as follows: a set of problems (rbgxxxa) are real-life problems derived from a stacker crane application by Ascheuer (1995). These problems were originally defined as ATSPs with time windows: to obtain SOP instances, time window precedences are relaxed to generate SOP precedences. Prob.100 (Ascheuer 1995) is a randomly generated problem, and problems (ftxx.x, and kroxxxp.x) have been generated by Ascheuer (1995) starting from ATSP instances in TSPLIB by adding a number $\leq k$ of random precedence constraints, where $k = (n/4, n/2, 2, 2n)$ correspond to the problem extension $(.1, .2, .3, .4)$. ESC78 is taken from Escudero (1988).

### 4.4.2 Experimental Settings: Competing Methods

The algorithms with which we compared HAS-SOP are the following:

- MPO/AI: This was previously the best known algorithm for the SOP (Chen & Smith 1996). MPO/AI (Maximum Partial Order/Arbitrary

Insertion) is a genetic algorithm explicitly designed to solve sequencing problems. Each individual is a feasible sequence represented by an $nxn$ Boolean matrix. An element $(i, j)$ of the matrix is set to 1 if node $j$ follows (not necessary immediately) node $i$ in the sequence, and is set to 0 otherwise. New individuals are generated by a specialized crossover operation. First, the two matrices are intersected; the intersection generates a new matrix where, in general, only partial subsequences (with fewer than $n$ elements) are present. Next, the longest subsequence (Maximum Partial Order) in the new matrix is selected and is completed by using an Arbitrary Insertion (AI) procedure. AI starts from a sub-tour, picks an arbitrary node not already included, and inserts it in the feasible position with minimum cost. This simple local search procedure is applied until no further elements are available. The code has been implemented by Chen & Smith (1996). Experiments were run setting the population to many different dimensions. Using 500 individuals, the same population dimension as proposed by Chen & Smith (1996), resulted in the best performance, and this value was used in all the experiments presented in the following.

- MPO/AI+LS: This is MPO/AI to which we added the SOP-3-exchange local search. The hybridization is similar to what was done with ACS-SOP: each time a new individual is created by the MPO/AI crossover operation, it is optimized by the SOP-3-exchange local search (with the main structure of the genetic algorithm remaining unchanged).
- RND: This algorithm generates random feasible solutions. The constructive procedure is the same as in ACS-SOP except that pheromone trail and distance information are not used.
- RND+LS: This is RND plus local search. As with the other hybrid algorithms considered, each time a new individual is created it is optimized by the SOP-3-exchange local search.

### 4.4.3 Computational Results: Selection Criteria for Node $i$ and Search Stopping Criteria

In this section we test different selection criteria for node $i$ and different search stopping criteria. We ran five experiments for each problem, setting the computational time to 100 seconds for the ft53.x, ft70.x and

Table 4.1: Ranking of median rank on 22 SOP test problems for different combinations of selection and stopping criteria. Results are obtained running five experiments for each problem (CPU time was set to 100 seconds for the ft53.x, ft70.x and ESCxx problems, to 300 seconds for the kro124p.x problems, and to 600 seconds for the other problems).

| SelectionCriterion | ExchangeFirstCriterion | WalkingCriterion | Median |
|---|---|---|---|
| don't push stack | i | 3-exchange | 4 |
| don't push stack | j | 3-exchange | 5 |
| don't push stack | h | 3-exchange | 6 |
| don't push stack | j | OR_exchange | 6 |
| don't look bit | h | 3-exchange | 7 |
| don't push stack | h | OR_exchange | 8 |
| don't look bit | j | 3-exchange | 8 |
| don't look bit | i | 3-exchange | 8 |
| sequential | h | OR_exchange | 9 |
| don't push stack | i | OR_exchange | 10 |
| don't look bit | h | OR_exchange | 10 |
| sequential | i | OR_exchange | 11 |
| don't look bit | j | OR_exchange | 12 |
| sequential | j | OR_exchange | 12 |
| sequential | h | 3-exchange | 13 |
| sequential | j | 3-exchange | 13 |
| sequential | i | 3-exchange | 13 |
| don't look bit | i | OR_exchange | 14 |

ESCxx problems, to 300 seconds for the kro124p.x problems, and to 600 seconds for the other problems.

The stopping criteria tested are: $ExchangeFirstCriterion = j, i, h$. The selection criteria tested are *sequential*, *don't look bit*, and *don't push stack*, coupled with either the *3-exchange* or the OR-*exchange* walking criterion. For each test problem (the problems are reported in Table 4.2 and Table 4.3), we ranked results computed by the different combinations of selection and stopping criteria according to the average results obtained. In Table 4.1 the methods are ranked by the median of each method over the set of test problems. Results indicate that the *don't push stack* is the best selection criterion, followed by the *don't look bit* and finally by the *sequential* selection criterion. Figure 4.6 compares

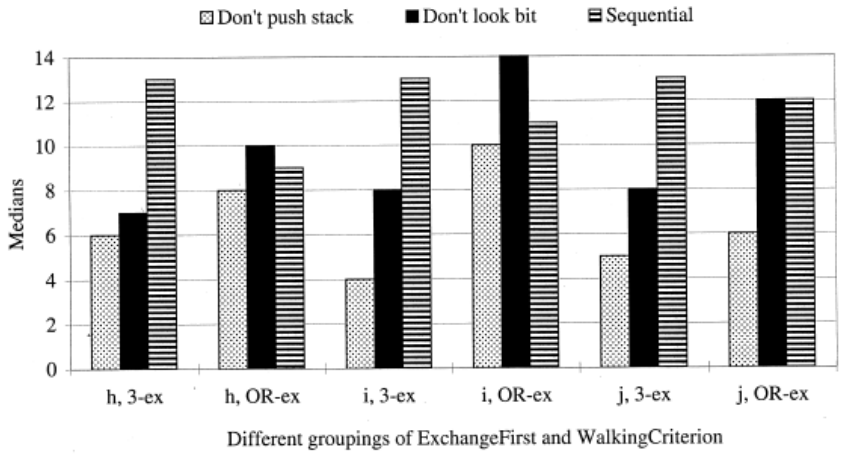


Figure 4.6: Comparisons of selection criteria on median ranks. Each comparison involves the same value of ExchangeFirstCriterion and WalkingCriterion. Results are obtained running five experiments for each problem (CPU time was set to 100 seconds for the ft53.x, ft70.x and ESCxx problems, to 300 seconds for the kro124p.x problems, and to 600 seconds for the other problems).

Table 4.2: Small Problems (≤100 Nodes). Shown are the average percentages of deviation from the Best-Known solution. Results are obtained over five runs of 120 seconds. Best results are in boldface.

| | RND | MPO/AI | ACS-SOP | RND+LS | MPO/AI+LS | HAS-SOP |
|---|---|---|---|---|---|---|
| ESC78 | 49.81% | 0.86% | 2.15% | **0.00%** | **0.00%** | **0.00%** |
| ft53.1 | 167.93% | 0.49% | 13.11% | 0.10% | **0.00%** | **0.00%** |
| ft53.2 | 154.94% | 0.72% | 12.27% | 0.36% | **0.00%** | **0.00%** |
| ft53.3 | 100.51% | 0.59% | 18.51% | **0.00%** | **0.00%** | **0.00%** |
| ft53.4 | 40.99% | **0.00%** | 5.03% | **0.00%** | **0.00%** | **0.00%** |
| ft70.1 | 64.94% | 0.76% | 11.65% | 0.37% | 0.10% | **0.00%** |
| ft70.2 | 59.18% | 0.03% | 11.63% | 0.85% | **0.00%** | 0.02% |
| ft70.3 | 52.22% | 0.03% | 13.22% | 0.49% | **0.00%** | **0.00%** |
| ft70.4 | 24.62% | 0.09% | 3.92% | 0.08% | **0.02%** | 0.05% |
| kro124p.1 | 301.69% | 4.17% | 28.81% | 2.65% | 0.68% | **0.00%** |
| kro124p.2 | 278.99% | 3.00% | 27.90% | 2.90% | **0.19%** | 0.26% |
| kro124p.3 | 215.49% | 3.20% | 24.49% | 3.75% | 1.40% | **0.31%** |
| kro124p.4 | 94.07% | **0.00%** | 8.66% | 1.23% | **0.00%** | **0.00%** |
| Average | 123.49% | 1.07% | 13.95% | 0.98% | 0.18% | **0.05%** |

Table 4.3: Big (>100 Nodes). Shown are the average percentages of deviation from the Best-Known solution. Results are obtained over five runs of 600 seconds. Best results are in boldface.

| | RND | MPO/AI | ACS-SOP | RND+LS | MPO/AI+LS | HAS-SOP |
|---|---|---|---|---|---|---|
| prob.100 | 1440.17% | 134.66% | 40.62% | 50.07% | 47.58% | **17.46%** |
| rbg109a | 64.57% | 0.33% | 1.93% | 0.08% | 0.06% | **0.00%** |
| rbg150a | 37.85% | 0.19% | 2.54% | 0.08% | 0.13% | **0.00%** |
| rbg174a | 40.86% | 0.01% | 2.16% | 0.15% | **0.00%** | 0.08% |
| rbg253a | 45.85% | 0.03% | 2.68% | 0.21% | **0.00%** | **0.00%** |
| rbg323a | 80.14% | 1.08% | 9.60% | 1.27% | **0.08%** | 0.21% |
| rbg341a | 125.46% | 3.02% | 12.64% | 4.41% | **0.96%** | 1.54% |
| rbg358a | 151.92% | 7.83% | 20.20% | 4.98% | 2.51% | **1.37%** |
| rbg378a | 131.58% | 5.95% | 22.02% | 4.17% | 1.40% | **0.88%** |
| Average | 235.38% | 17.01% | 12.71% | 7.27% | 5.86% | **2.39%** |

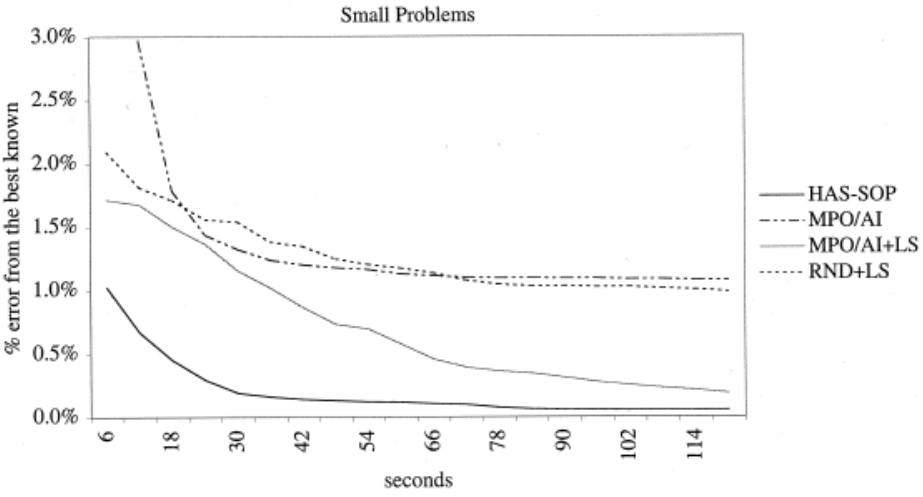


Figure 4.7: Comparison across algorithms over small problems. Results are obtained over five runs of 120 seconds.

the three selection criteria for the same values of ExchangeFirstCriterion and WalkingCriterion. Again, it is clear that *don't push stack* performs better than the other two criteria. These results were obtained using HAS-SOP. That is, the SOP-3-exchange local search was applied to feasible solutions generated by ACS-SOP. We ran the same experiment using

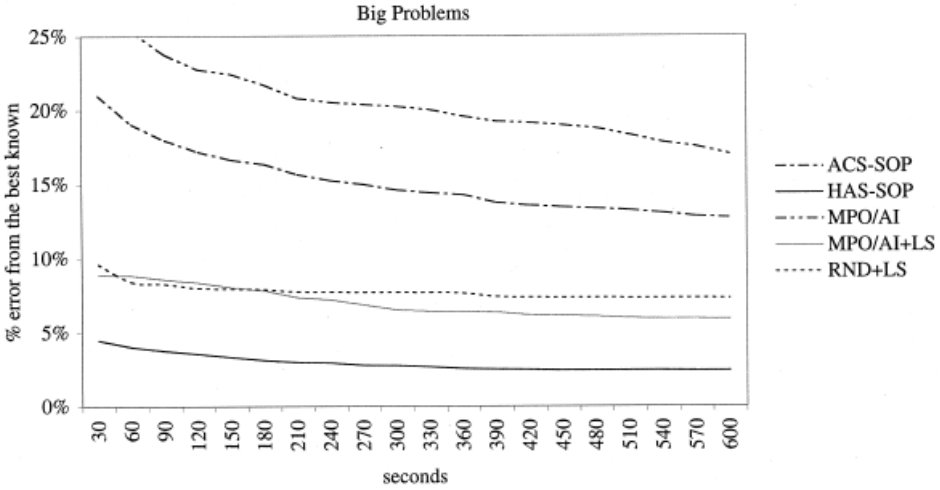


Figure 4.8: Comparison across algorithms over big problems. Results are obtained over five runs of 600 seconds.

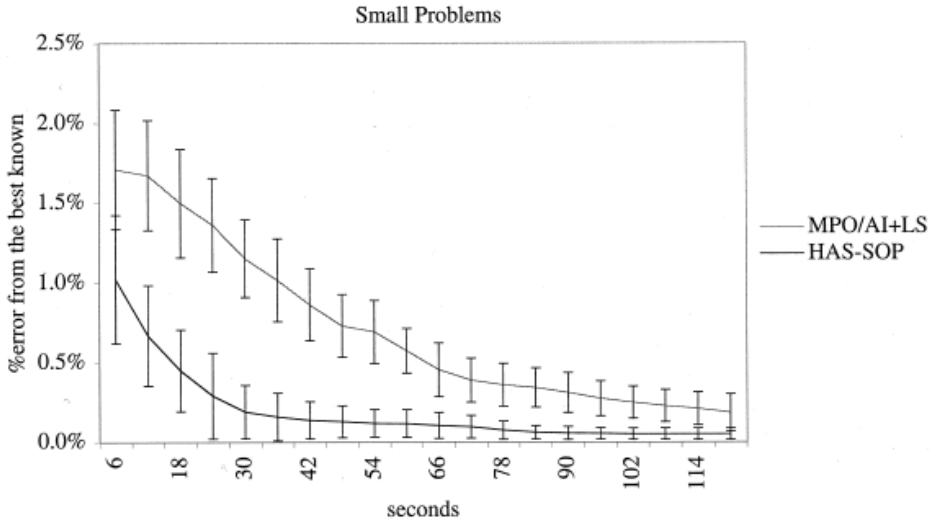


Figure 4.9: Comparison between MPO/AI+LS and HAS-SOP over small problems. Results are obtained over five runs of 120 seconds. Error bars (1 standard deviation) are shown.

the other solution generation methods (i.e., MPO/AI and RND), and we found that also in these cases the best performance was obtained by

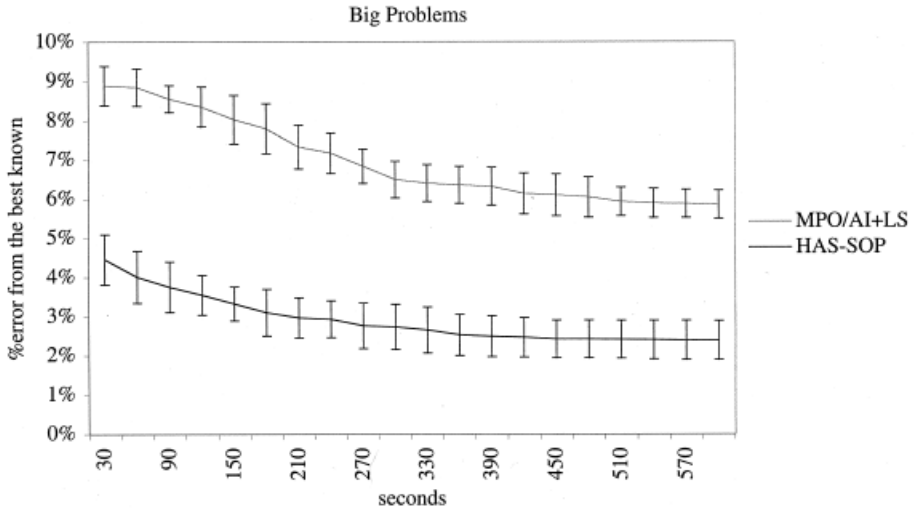


Figure 4.10: Comparison between MPO/AI+LS and HAS-SOP over big problems. Results are obtained over five runs of 600 seconds. Error bars (1 standard deviation) are shown.

setting *SelectionCriterion=don't push stack, ExchangeFirstCriterion=i, and WalkingCriterion=3-exchange*. These parameters are therefore used in all the experiments involving local search presented in the following sections. (It should otherwise be noted that, for MPO/AI and RND, although the best parameter settings remained the same, the ordering of the other possible combinations of parameter values was different).

### 4.4.4 Computational Results and Comparisons with Other Methods

In this section, we compare the ACS-SOP, RND, and MPO/AI algorithms and their hybrid versions (using the local search with the best parameters experimentally found as explained in Section 4.4.3. To run the comparisons we divided the set of test problems in two sets: smaller easier problems, and larger more difficult problems. The separation point was set to be 100 nodes: small problems have 100 or fewer nodes, big problems have more than 100 nodes (with the exception of prob.100 that, because of its difficulty, although having 100 nodes was assigned to the set of big problems). Experiments were run giving a fixed amount of

Table 4.4: Percentage of improvement due to local search. The last three rows report respectively the average over the Small, Big and All problems. Results are obtained over five runs of 120 seconds (Small problems, that is, ft53.x, ft70.x, ESCxx and kro124p.x problems) and 600 seconds for the others.

| | △ % RND | △% MPO/AI | △% ACS-SOP |
|---|---|---|---|
| ESC78 | 49.81% | 0.86% | 2.15% |
| ft53.1 | 167.83% | 0.49% | 13.11% |
| ft53.2 | 154.58% | 0.72% | 12.27% |
| ft53.3 | 100.51% | 0.59% | 18.51% |
| ft53.4 | 40.99% | 0.00% | 5.03% |
| ft70.1 | 64.57% | 0.66% | 11.65% |
| ft70.2 | 58.33% | 0.03% | 11.61% |
| ft70.3 | 51.73% | 0.03% | 13.22% |
| ft70.4 | 24.55% | 0.06% | 3.88% |
| kro124p.1 | 299.04% | 3.50% | 28.81% |
| kro124p.2 | 276.09% | 2.81% | 27.65% |
| kro124p.3 | 211.74% | 1.81% | 24.18% |
| kro124p.4 | 92.84% | 0.00% | 8.66% |
| prob.100 | 1390.10% | 87.08% | 23.16% |
| rbg109a | 64.49% | 0.27% | 1.93% |
| rbg150a | 37.77% | 0.07% | 2.54% |
| rbg174a | 40.71% | 0.01% | 2.09% |
| rbg253a | 45.64% | 0.03% | 2.68% |
| rbg323a | 78.87% | 1.00% | 9.39% |
| rbg341a | 121.05% | 2.06% | 11.10% |
| rbg358a | 146.94% | 5.31% | 18.83% |
| rbg378a | 127.41% | 4.55% | 21.14% |
| | | | |
| small avg | 122.51% | 0.89% | 13.90% |
| big avg | 228.11% | 11.15% | 10.32% |
| all avg | 165.71% | 5.09% | 12.44% |

CPU time to the algorithms. The CPU time was fixed to be the same for all algorithms running on the same set of problems: 120 seconds for each small problem, 600 seconds for each big problem. Results at the end of the experiment are reported in Table 4.2 and Table 4.3 for small and big problems respectively, while the runtime behavior of the various

Table 4.5: Results obtained by HAS-SOP and MPO/AI+LS on a set of 22 test problems. See text for explanation of boldface and total wins row. Results are obtained over five runs of 120 seconds (Small problems, that is, ft53.x, ft70.x, ESCxx and kro124p.x problems) and 600 seconds for the others.

| | MPO/AI+LS | | | | HAS-SOP | | | |
|---|---|---|---|---|---|---|---|---|
| PROB | Best Result | Avg. Result | Std. Dev. | Avg. Time (sec) | Best Result | Avg. Result | Std. Dev. | Avg. Time (sec) |
| ESC78 | 18230 | 18230.0 | 0.0 | 12.4 | 18230 | 18230.0 | 0.0 | **3.5** |
| ft53.1 | 7531 | 7531.0 | 0.0 | 16.6 | 7531 | 7531.0 | 0.0 | **16.3** |
| ft53.2 | 8026 | 8026.0 | 0.0 | **14.0** | 8026 | 8026.0 | 0.0 | 17.0 |
| ft53.3 | 10262 | 10262.0 | 0.0 | 7.8 | 10262 | 10262.0 | 0.0 | **3.8** |
| ft53.4 | 14425 | 14425.0 | 0.0 | 11.4 | 14425 | 14425.0 | 0.0 | **0.5** |
| ft70.1 | 39313 | 39352.4 | 33.2 | 81.4 | 39313 | **39313.0** | 0.0 | 20.9 |
| ft70.2 | 40419 | **40419.6** | 1.2 | 81.6 | 40419 | 40428.6 | 12.0 | 41.0 |
| ft70.3 | 42535 | 42535.0 | 0.0 | **27.0** | 42535 | 42535.0 | 0.0 | 36.8 |
| ft70.4 | 53530 | **53542.8** | 15.7 | 42.2 | 53530 | 53554.6 | 20.5 | 58.3 |
| kro124p.1 | 39502 | 39686.2 | 214.0 | 97.4 | **39420** | **39420.0** | 0.0 | 60.8 |
| kro124p.2 | 41336 | **41415.4** | 114.2 | 95.2 | 41336 | 41442.8 | 127.8 | 53.2 |
| kro124p.3 | 49835 | 50189.6 | 298.0 | 97.4 | **49499** | **49653.2** | 66.3 | 24.2 |
| kro124p.4 | 76103 | 76103.0 | 0.0 | 47.8 | 76103 | 76103.0 | 0.0 | **34.2** |
| prob.100 | 1722 | 1756.2 | 30.6 | 333.0 | **1344** | **1397.8** | 38.5 | 404.4 |
| rbg109a | 1038 | 1038.6 | 0.5 | 75.8 | 1038 | **1038.0** | 0.0 | 27.5 |
| rbg150a | 1751 | 1752.2 | 0.7 | 17.2 | **1750** | **1750.0** | 0.0 | 128.1 |
| rbg174a | 2033 | **2033.0** | 0.0 | 82.8 | 2033 | 2034.6 | 1.1 | 189.4 |
| rbg253a | 2950 | 2950.0 | 0.0 | **68.6** | 2950 | 2950.0 | 0.0 | 145.0 |
| rbg323a | **3143** | **3143.6** | 0.4 | 458.8 | 3146 | 3147.6 | 2.2 | 271.1 |
| rbg341a | **2588** | **2598.8** | 2.0 | 553.6 | 2609 | 2613.6 | 18.1 | 421.3 |
| rbg358a | 2602 | 2609.0 | 6.2 | 482.8 | **2574** | **2579.8** | 4.8 | 454.1 |
| rbg378a | 2841 | 2856.4 | 8.1 | 516.0 | **2831** | **2841.8** | 6.8 | 500.6 |
| Total wins | 2 | 6 | — | 3 | 6 | 8 | — | 5 |

algorithms is shown in Figure 4.7 and Figure 4.8.

If we analyze the average performance of the algorithms on the set of small problems (Table 4.2) we can make the following observations: (i) RND is, as it was expected, the worst performing algorithm; (ii) ACS-SOP performs better than RND, which means that the additional use of pheromone trails and local heuristic information (i.e., distance between nodes) is useful, (iii) MPO/AI was the best of the algorithms not using our local search (in fact, MPO/AI uses a simple form of local search, which can explain its better performance), (iv) when the SOP-3-exchange

Table 4.6: New bounds for Sequential Ordering Problems. New upper bounds were obtained by HAS-SOP and MPO/AI+LS, while new Lower Bounds were obtained by a Branch-and-Cut program starting from HAS-SOP solutions. All Best columns report the best solutions computed by the two algorithms. In parentheses are the results obtained by applying the post optimization.

| PROB | $n$ | $\mid R \mid$ | TSPLIB Bounds | NEW Lower Bounds | NEW Upper Bounds | All Best HAS-SOP | All Best MPO/AI+LS |
|---|---|---|---|---|---|---|---|
| ESC63 | 65 | 95 | 62 | | | 62 | 62 |
| ESC78 | 80 | 77 | 18230 | | | 18230 | 18230 |
| ft53.1 | 54 | 12 | [7438,7570] | | 7531 | 7531 | 7531 |
| ft53.2 | 54 | 25 | [7630,8335] | | 8026 | 8026 | 8026 |
| ft53.3 | 54 | 48 | [9473,10935] | | 10262 | 10262 | 10262 |
| ft53.4 | 54 | 63 | 14425 | | | 14425 | 14425 |
| ft70.1 | 71 | 17 | 39313 | | | 39313 | 39313 |
| ft70.2 | 71 | 35 | [39739,40422] | 39803 | 40419 | 40419 | 40419 |
| ft70.3 | 71 | 68 | [41305,42535] | | | 42535 | 42535 |
| ft70.4 | 71 | 86 | [52269,53562] | 53072 | 53530 | 53530 | 53530 |
| kro124p.1 | 101 | 25 | [37722,40186] | 37761 | 39420 | 39420 | 39420 |
| kro124p.2 | 101 | 49 | [38534,41677] | 38719 | 41336 | 41336 | 41336 |
| kro124p.3 | 101 | 97 | [40967,50876] | 41578 | 49499 | 49499 | 49519 |
| kro124p.4 | 101 | 131 | [64858,76103] | | | 76103 | 76103 |
| prob.100 | 100 | 41 | [1024,1385] | 1027 | 1190 | 1219 (1190) | 1573 |
| rbg109a | 111 | 622 | 1038 | | | 1038 | 1038 |
| rbg150a | 152 | 952 | [1748,1750] | | | 1750 | 1750 |
| rbg174a | 176 | 1113 | 2033 | | | 2033 | 2033 |
| rbg253a | 255 | 1721 | [2928,2987] | 2940 | 2950 | 2950 | 2950 |
| rbg323a | 325 | 2412 | [3136,3157] | 3137 | 3141 | 3141 | 3141 |
| rbg341a | 343 | 2542 | [2543,2597] | | 2570 | 2576 (2574) | 2572 (2570) |
| rbg358a | 360 | 3239 | [2518,2599] | 2529 | 2545 | 2549 (2545) | 2555 |
| rbg378a | 380 | 3069 | [2761,2833] | | 2816 | 2817 | 2816 |

local search is added all the algorithms, as expected, increase their performance, and HAS-SOP with an average 0.05% deviation from the best-known solutions is the best performing algorithm. Similar observations can be done for the set of big problems (Table 4.3). The only difference is that in the average ACS-SOP performs better than MPO/AI. This is mainly due to problem prob.100, a difficult problem that ACS-SOP solves much better than the competing methods. Also in the case of big problems HAS-SOP is the best performing algorithm, with an average error of 2.39% from the best-known solutions. Figure 4.7 shows the run-time behavior of HAS-SOP, MPO/AI, MPO/AI+LS, and RND+LS on small problems (RND and ACS-SOP are not plotted because they are

out of scale). It is clear that, besides reaching slightly better results than MPO/AI+LS, HAS-SOP has also a better convergence speed: it reaches after 12 seconds the same performance level reached by MPO/AI+LS after approximately 60 seconds. Similar considerations can be done for big problems (Figure 4.8) where all algorithms are plotted (with the exception of RND, which is out of scale). Note the small difference in behavior between RND+LS and MPO/AI+LS. A more detailed version of Figure 4.7 and Figure 4.8 showing the performance of the two best algorithms, HAS-SOP and MPO/AI+LS, with error bars is given in Figure 4.9 and Figure 4.10.

Table 4.4 shows the percentage improvement due to local search (this is computed as the difference between the performance of the basic algorithm and the performance of the corresponding hybrid algorithm reported in Table 4.2 and Table 4.3). Data show that MPO/AI profits from local search less than ACS-SOP and RND. This is probably due to the fact that MPO/AI generates solutions that are already close to local optima and therefore the SOP-3-exchange procedure quickly gets stuck. On the contrary, RND is the algorithm that best exploits local search. Unfortunately, this is due to the very poor quality of the solution given as a starting point to the local search: notwithstanding the great improvement caused by the local search, the final result is not competitive with that produced by HAS-SOP. In some sense it seems that solutions generated by ACS-SOP are good enough to let local search work fruitfully, yet they are not so good as to impede local search to work, as it is the case for MPO/AI.

In Table 4.5 we compare HAS-SOP with MPO/AI+LS. As in the previous experiment, runs lasted different amounts of CPU time, 120 seconds for small problems and 600 seconds for big problems. Each experiment was run 5 times. For both algorithms in Table 4.5 we report:

- Best Result: the best result obtained over 5 experiments.
- Avg. Result: average of the best results obtained in each experiment.
- Std. Dev.: standard deviation of the best results obtained in each experiment.
- Avg. Time: average time (in seconds) needed to reach the best result in each experiment.

In the table we have marked in boldface the results according to these criteria: First we consider the columns Best Result and for each problem we mark in boldface the best of the best results obtained by the two algorithms. Similarly we compare and mark in boldface the best average results. Then, only for those problems on which the two algorithms obtained the same average result, we mark with boldface the lowest average time. In the last row of Table 4.5 we report the number of wins, that is, the number of times one algorithm was better than the other one for each of the considered criteria (this corresponds to the number of boldface entries in each column). The Total wins row synthetically shows that HAS-SOP has a better performance than MPO/AI+LS on all the measured criteria.

In conclusion, in Table 4.6 we report the new upper bounds obtained by HAS-SOP and by MPO/AI+LS, as well as new lower bounds obtained by a branch-and-cut program run by Ascheuer (1995) starting from HAS-SOP solutions. The first column gives the problem names, the second column gives the size of the problem in term of the number $n$ of nodes, the third columns gives the number $\mid R \mid$ of constraints, and the fourth column the bounds reported in TSPLIB. The other columns report the new upper and lower bounds we computed, and finally the All Best columns report the best solutions computed by the HAS-SOP and MPO/AI+LS algorithms. In parentheses the results obtained applying a post optimization consisting of re-running the algorithm (HAS-SOP or MPO/AI), starting from the best found solution but using as local search one of the variants presented in Table 4.1 (the post-optimization was run for all the problems, but only in four cases it was able to improve the best solution found in the first optimization phase).

## 4.5 Conclusions

The contribution of this chapter is twofold. First, we have introduced a new local search procedure for the sequential ordering problem (SOP) called SOP-3-exchange. This procedure has been shown to produce solutions of quality higher than that of solutions produced by MPO/AI. This has been shown to be the case even when the local search is applied to poor-quality, randomly generated initial solutions. Second, we have shown that the performance of the algorithm obtained by coupling MPO/AI with SOP-3-exchange can still be improved by coupling the

local search with ACS-SOP, a straightforward extension of Ant Colony System (Chapter 2.3).

# Chapter 5

# EACS: Coupling Ant Colony Systems With Strong Local Searches

## 5.1 Introduction

Ant Colony System (Gambardella & Dorigo 1996; Dorigo & Gambardella 1997) has been presented in Chapter 2. In this thesis the coupling between ACS and local search algorithms have been successfully applied to many combinatorial optimization problems. In particular ACS has been able to successfully solve symmetric and asymmetric TSP problems (Chapter 2, Dorigo & Gambardella 1997), the vehicle routing problems with time windows (MACS-VRPTW, Chapter 3, Gambardella et al. 1999) and the sequential ordering problem (HAS-SOP, Chapter 4, Gambardella & Dorigo 2000).
After these experiences in this chapter we deeply analyze the situation where a strong local search routine is available for an optimization problem. It is shown how the original ACS framework can be enhanced to achieve a better integration between ACS and the local search. Experimental results on SOP optimization problem arising in transportation is discussed. The results show the effectiveness of the enhancements introduced.

The chapter is organized as follows: Section 5.2 describes in detail some drawbacks of the classic ACS algorithm when coupled with a strong local search, and introduces two new operations to overcome these draw-

backs, leading to the EACS algorithm (Gambardella et al. 2012). Section 5.3 is devoted to the experimental validation of the new EACS paradigm. In particular, the method is applied to the sequential ordering problem, and the results are reported in Section 5.4. Section 5.5 contains our conclusions.

## 5.2 An Enhanced Ant Colony System

We begin this section with a definition which will be central in the following discussion, and in the remainder of the chapter.

**Definition 1.** Given an optimization problem, a local search procedure is referred to as a **strong local search** if it is able to autonomously and efficiently retrieve high quality solutions for the given problem, without the help of an external procedure guiding it.

The definition above formalizes a well-known concept to practitioners in the metaheuristics field. An example of strong local search procedure can be found when considering the classic traveling salesman problem. With reference to Section 7.6 of Reinelt (1994), the *Lin-Kernighan* local search procedure, that guarantees optimality gaps in the order of 1%, is a strong local search, while the simpler 2-opt local search procedure (optimality gaps above 5%) is not.

With reference to the ACS paradigm described in Chapter 2 and Algorithm 19), we can observe that the *constructive phase* of the original ACS framework carries out both diversification (exploring new regions of the search space) and intensification (searching very deeply a given region of the search space), since the original ACS algorithm did not consider any local search. The constructive phase of ACS is therefore able to generate improving solutions that are in a neighborhood of the best solution computed so far. On the other hand, the local search procedure is considered as a strong intensification process, able to bring each solution computed by the artificial ants to its local minimum. So, in presence of a strong local search procedure, the role of the construction phase is less prominent and has to be revised, since the local search itself is able to efficiently cover vast regions of the search space in its intensification process. In particular the constructive phase could drop any kind of intensification,

Algorithm 19: ACS Components for EACS, the Enhanced Ant Colony System

$L_{gb} \leftarrow \infty$
**for each** move $(i, j)$ **do**
  $\tau_{ij} \leftarrow \tau_0$
**end for each**
**while** (termination criteria not met)
  **for** $k := 1$ to $m$ **do**
    **while** (Ant $k$ has not completed its solution) **do**
      **for each** move $(i, j)$ from the current state $i$ **do**
        Compute $\mu_{ij}$
      **end for each**
      **if** (uniform random number in $[0, 1] > q_0$) **then**
        Choose the move $(i, j)$ at random, with probability $\frac{[\tau_{ij}]\cdot[\mu_{ij}]^\beta}{\sum_{u\in J_r^k}[\tau_{ij}]\cdot[\mu_{ij}]^\beta}$ (2.1)
      **else**
        Choose the move $(i, j)$ maximizing $p_{ij}^k$ with $arg\ \max_{u\in J_r^k}\{[\tau_{ij}] \cdot [\mu_{ij}]^\beta\}$ (2.4)
      **end if**
      Append new infeasible moves to the $k$-th ant's set $J_i^k$
      Update the trail level $\tau_{ij}$ by means of $\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0$ (2.10)
    **end while**
    $L_k \leftarrow$ Apply a local search to improve the current solution;
    **if** $(L_k < L_{gb})$ **then**
      $L_{gb} \leftarrow L_k$;
    **end if**
  **end for**
  **for each** move $(i, j) \in L_{gb}$ **do**
    Update the trail level $\tau_{ij}$ by means of $\tau_{ij} \leftarrow (1 - \alpha) \cdot \tau_{ij} + \alpha \cdot \frac{1}{L_{gb}}$ (2.5)
  **end for each**
**end while**

concentrating only on a (milder) form of diversification. This is the main consideration at the basis of the EACS framework we propose, together with the observation that the local search itself can be better integrated within the framework. Similar considerations had led in the past to efficient combinations of simulated annealing with local search procedures Martin & Otto (1996), or path-relinking with scatter search and GRASP methods (Resende et al. 2010; Resende & Ribeiro 2005), leading however to algorithmic frameworks that are intrinsically very different from the one we propose. Finally, it is important to observe that since 1999, when the first ACO algorithm was introduced, many variants and improvements to the original paradigm have been proposed, with the aim of improving the performance of the method (we refer the interested reader

to Dorigo & Blum (2005) and Monmarché et al. (2010)). However, none of the ideas developed has been in the direction of a better integration between the constructive phase and the (strong) local search components, like the method we propose.

In this chapter two operations to enhance the performance of ACS are suggested: the first one regards the constructive phase, and the second the integration between the constructive phase itself, and the local search procedure. These enhancements, introduced with the aim of speeding up the original ACS algorithm, will be described in detail in the remainder of this section. Notice that the enhancements proposed are in line with the theoretical results discussed in Kötzing et al. (2010). Thanks to the two enhancements we propose to heuristically limit the search space considered by the algorithm, the quality of the solution produced by the Enhanced Ant Colony System will be shown to be higher than those of the classic ACS (and often than those of state-of-the-art methods), due to the remarkable speed-up guaranteed by the enhancements introduced.

One known drawback of the ACS approach is the large total running time required to build new solutions by each artificial ant. Let $n$ be the number of steps necessary to build a solution. Usually the constructive process takes time $O(n)$ for each of the $n$ steps required. This is acceptable in case of small problems, but it is too expensive in case of larger problems. In fact, ACS algorithms are not usually able to replicate on large instances the good performance they are able to provide for smaller instances. We propose to modify the ACS algorithm in two directions to overcome the computational speed issue, as described in the remainder of this section.

### 5.2.1 An Improved Constructive Phase

Our proposal to speed up the constructive phase is based on a new approach which - in contrast with the classic ACS algorithm - directly considers the best solution computed so far $L_{gb}$ already during the constructive phase. In the classic ACS algorithm an ant in state $i$ selects the next state $j$ according to a probabilistic criterion. With probability $q_0$ the state selected is that with the best weighted compromise between pheromone trail and heuristic desirability, while with probability $(1 - q_0)$ the edge is selected according to a Monte Carlo sampling mechanism. In our new proposal, the state selected with probability $q_0$ is the state reached after the current state $i$ in the best solution $L_{gb}$ computed so far

(in case this state is not feasible, the classic mechanism described above is applied). Since probability $q_0$ is usually greater than or equal to 0.9, the new approach drastically reduces the running time required to select the next edge to visit (typically from $O(n)$ to something approximable by a constant).

### 5.2.2 A Better Integration Between the Constructive Phase and the Local Search Procedure

In the conventional ACS framework, after the constructive phase is completed, each solution is brought to its local minimum using a local search procedure. The second enhancement we propose is again in the direction of speeding up the whole algorithm, and concerns a better integration between the constructive and the local search phases of the algorithm, leading to a faster overall method. Three different ways to obtain such an integration can be identified:

1. A first idea is to apply the local search procedure only on a (promising) subset of the solutions generated, contrary to the canonical ACS paradigm, where the local search is applied to all the solutions generated.

2. The local search procedure is applied (probabilistically) only on those solutions on which the local search has not been recently applied, in order to avoid searching the neighborhood of the same solution over and over again.

3. The local search procedure is run only from those solution components (parts of a solution, e.g. subtours) which are not present in the solutions obtained during the previous iterations.

All the three strategies listed above are very general, and their implementation is extremely problem-dependent.

### 5.2.3 Pseudo-Code

A pseudo-code for the new EACS framework is provided in Algorithm 20. Lines marked with an asterisk are those containing differences with respect to the original ACS algorithm (see Algorithm 19).

Algorithm 20: EACS, the Enhanced Ant Colony System

$L_{gb} \leftarrow \infty$
**for each** move $(i,j)$ **do**
  $\tau_{ij} \leftarrow \tau_0$
**end for each**
**while** (termination criteria not met)
  **for** $k := 1$ to $m$ **do**
    **while** (Ant $k$ has not completed its solution)
(*)  **if** (uniform random number in $[0,1] > q_0$) **then**
(*)  // The next loop is executed with low probability
(*)  **for each** move $(i,j)$ from the current state $i$ **do**
(*)  Compute $\mu_{ij}$
(*)  **end for each**
(*)  Choose the move $(i,j)$ at random, with probability $\frac{[\tau_{ij}]\cdot[\mu_{ij}]^\beta}{\sum_{u\in J_r^k}[\tau_{ij}]\cdot[\mu_{ij}]^\beta}$ (2.1)
(*)  **else**
(*)  $b^* \leftarrow$ state such that move $(i,b^*) \in L_{gb}$
(*)  **if** (move $(i,b^*) \notin J_i^k$) **then**
(*)  Choose the move $(i,b^*)$
(*)  **else**
(*)  **for each** move $(i,j)$ from the current state $i$ **do**
(*)  Compute $\mu_{ij}$
(*)  **end for each**
(*)  Choose the move $(i,j)$ maximizing $p_{ij}^k$ with $arg\ \max_{u\in J_r^k}\{[\tau_{ij}]\cdot[\mu_{ij}]^\beta\}$ (2.4)
(*)  **end if**
(*)  **end if**
  Append new infeasible moves to the $k$-th ant's set $J_i^k$
  Update the trail level $\tau_{ij}$ by means of $\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \rho\cdot\tau_0$ (2.10)
    **end while**
    // be $L_k$ the current solution computed by ant $k$
(*)  // Depending on the implementation, some of the following
(*)  // conditions might be not implemented
(*)  **if** ($L_k$ is close enough to $L_{gb}$)
(*)  **and** (local search has not been recently run on the current solution $L_k$)
(*)  **and** (the current solution $L_k$ is different from $L_{gb}$) **then**
(*)  Apply a local search to those components of the current
(*)  solution $L_k$ that are not in common with $L_{gb}$
(*)  **end if**
    **if** ($L_k < L_{gb}$) **then**
      $L_{gb} \leftarrow L_k$
    **end if**
  **end for**
  **for each** move $(i,j) \in L_{gb}$ **do**
    Update the trail level $\tau_{ij}$ by means of $\tau_{ij} \leftarrow (1-\alpha)\cdot\tau_{ij} + \alpha\cdot\frac{1}{L_{gb}}$ (2.5)
  **end for each**
**end while**

## 5.3 Applications

The EACS paradigm was tested on different combinatorial optimization problems. EACS was compared both with the standard ACS method, and with state-of-the-art methods. For some well-known combinatorial optimization problems for which a strong, dominant local search is available, the EACS was however able to improve both ACS and the best known methods, leading to new reference results. Experiments in this direction have been presented in Gambardella et al. (2012) for the Team Orienteering Problem (TOP, Butt & Cavalier 1994; Chao et al. 1996) for the Probabilistic Traveling Salesman Problems (PTSP, Jaillet 1985) and for the Sequential Ordering Problem (SOP, Chapter 4). In the remainder of this section we concentrate on the Sequential Ordering Problem (SOP).

### 5.3.1 SOP: The Sequential Ordering Problem

#### Problem description and literature review

The Sequential Ordering Problem (SOP), also referred to as the Asymmetric Travelling Salesman Problem with Precedence Constraints has been presented in Section 4.1.

The SOP models real-world problems such as production planning (Escudero 1988), single vehicle routing problems with pick-up and delivery constraints (Pulleyblank & Timlin 1991; Savelsbergh 1990), transportation problems in flexible manufacturing systems (Ascheuer 1995).

The first mathematical model and exact algorithms for the SOP were introduced in Ascheuer et al. (1993), Escudero et al. (1994), Ascheuer (1995) and Balas et al. (1995). Next, a branch and bound algorithm was presented by Hernàdvölgyi (2003) and Hernàdvölgyi (2004). A genetic algorithm has been proposed Chen & Smith (1996). A hybrid genetic algorithm was discussed by Seo & Moon (2003). A parallelized roll-out algorithm was described by Guerriero & Mancini (2003). Chapter 4, presented an approach based on Ant Colony System (ACS) coupled with a sophisticated Local Search (LS) procedure (Gambardella & Dorigo 2000). Montemanni et al. (2007), Montemanni et al. (2008) and Montemanni et al. (2009) built on top of this method, adding a Heuristic Manipulation Technique (HMT). A Discrete Particle Swarm Optimization (DPSO) method has been finally discussed in Anghinolfi et al. (2009) and Angh-

inolfi et al. (2011).

**ACS for the SOP**

The adaptation of the ACS paradigm (see Chapter 2) to the SOP is straightforward (see also Chapter 4). The constructive phase of each ant starts at the origin node and chooses the next nodes probabilistically. The only complication is represented by precedence constraints: each time the next node has to be chosen, nodes that if selected would violate some precedence constraint have to be inserted in the list of forbidden nodes. This can be easily done without any increase in the computation time of the classic constructive phase.

The local search adopted by the ACS for the SOP is extremely efficient (Section 4.3). This local search routine is a specialization to the sequential ordering problem of a known local search method for the asymmetric travelling salesman problem (Savelsbergh 1990). It is able to directly handle multiple constraints without increasing the computational complexity of the original local search by using a special labeling procedure (Section 4.3.3). Other important SOP-3-exchange features are related to the way nodes are selected during the local search (Section 4.3.4). The goal is to decrease the number of visited nodes by introducing two heuristics that influence how nodes are chosen: one is based on a *don't look bit* data structure proposed by Bentley (1992), while the other is based on a data structure called *don't push stack* introduced in Gambardella & Dorigo (2000). The *don't look bit* (Section 4.3.3) is a data structure in which a bit is associated with each node of the sequence. At the beginning of the search all bits are turned off. The bit associated with the selected node is turned on when a search for an improving move starting from this node fails. The bit associated with the node is turned off again when an improving exchange involving the node is executed. The use of *don't look bits* favors the exploration of nodes that have been involved in a profitable exchange. The *don't push stack* (Section 4.3.3) is a data structure based on a stack, which contains the set of nodes to be selected. At the beginning of the search the stack is initialized with all the nodes in the sequence (that is, it contains $n + 1$ elements). During the search, nodes are popped off the stack and feasible 3-exchanges starting from these nodes are investigated. In case a profitable exchange is executed the six nodes involved in this exchange are pushed onto the stack (if they do not already belong to it).

**EACS for the SOP**

With respect to the classic ACS implementation (see Chapter 2), the modification to the constructive phase, according to Section 5.2, is straightforward. If we are in node $i$, with probability $q_0$ we move to node $j$ if it follows $i$ in the best solution available, unless $j$ is not feasible (already visited or violating some precedence constraint). Otherwise the classic Monte Carlo sampling technique is applied.

The implementation of the integration between the constructive phase and the local search (see Section 5.2.2) works as follows. The local search is run only if the cost of the solution produced in the constructive phase is within 20 % of the best solution retrieved so far (this implements point 1 of Section 5.2.2. Moreover, the *don't push stack* (see Section 4.3.3) is initialized in such a way that only the elements that in the current solution are out of sequence with respect to the best solution, are in the stack. This pushes the exploration of the search space towards areas that were potentially unexplored in the previous iterations, and is an implementation of point 3 of Section 5.2.2. Notice that in case the current solution coincides with the best solution, the local search is not applied.

**Benchmark problems**

Benchmark problems are those adopted by Anghinolfi et al. (2011). The instances are publicly available[1]. Each instance is identified as *n-r-p*, where the following naming convention is adopted: $n$ is the number of nodes of the problem, i.e. $V = \{1, 2, \ldots, n\}$; $r$ is the cost range, i.e. $0 \leq c_{ij} \leq r$, $\forall i, j \in V$; $p$ is the approximate percentage of precedence constraints, i.e. the number of precedence constraints of the problem will be about $\frac{p}{100} \cdot \frac{n(n-1)}{2}$. Instances were created from the following values for the parameters: $n \in \{200, 300, 400, 500, 600, 700\}$; $r \in \{100, 1000\}$; $p \in \{1, 15, 30, 60\}$.

## 5.4 Results

The EACS algorithm has been coded in ANSI C, and the experiments presented have been run on a Dual AMD Opteron 250 2.4 GHz/4GB

[1] The SOPLIB2006 library is available at http://www.idsia.ch/~roberto/SOPLIB06.zip.

computer. The parameter settings adopted are as follows both for ACS and EACS (settings are those used in Gambardella & Dorigo (2000) and reported in Chapter 4). Comparison are executed among HAS-SOP in Table 5.1 and Table 5.2 referred as ACS (Gambardella & Dorigo 2000, Chapter 2), Discrete Particle Swarm Optimization (DPSO, Anghinolfi et al. 2009, 2011), and a heuristic manipulation technique for the sequential ordering problem (HMT, Montemanni et al. 2008).

**Comparison of the constructive phases (without local search).** In this set of experiments only the constructive phases of ACS and EACS are considered, and no local search is run, similarly to what was already presented in Gambardella et al. (2012) for the TOPTW. In Table 5.1 the instances are grouped by number of nodes, and the results obtained by the two methods are presented. For each instance-group some statistics over 10 runs with a maximum computation time of 600 seconds for each instance are reported. In particular, the average percentage deviation from the best known solutions, the percentage deviation of the best solutions produced over 10 runs from the best known solutions and the average number of solutions generated in the given time are reported both for $\text{ACS}_{NoLS}$ and $\text{EACS}_{NoLS}$. It emerges that the role of the local search for the SOP is more prominent than for the TOPTW (Team Orienteering Problems with Time Windows) as reported in Gambardella et al. (2012) since the constructive phases alone obtain poor results in general, both in terms of average and best results over the 10 runs considered. $\text{ACS}_{NoLS}$ is better than $\text{EACS}_{NoLS}$ both in terms of average and best results, indicating once more that the idea at the basis of EACS makes sense only when coupled with a good local search. Notice also that the best solutions provided by $\text{EACS}_{NoLS}$ are consistently worse than those of $\text{ACS}_{NoLS}$. In general, the experiments suggest once more that the pheromone has an important role in the original ACS paradigm when no (strong) local search is available. It is also interesting to observe that the difference in the average number of solutions generated in the given time is extremely favorable to EACS (more than in the TOPTW case), giving a measure of the speed-up guaranteed by the new constructive phase. Finally, it is interesting to observe how the quality of the solutions reported in the table deteriorates as the number of nodes increases. This is a known drawback of the ACS paradigm in general.

Table 5.1: The Sequential Ordering Problem. Comparison of the constructive phases (without local search).

| Instance | Avg results | | Best results | | Solutions generated | |
|---|---|---|---|---|---|---|
| group | $ACS_{NoLS}$ | $EACS_{NoLS}$ | $ACS_{NoLS}$ | $EACS_{NoLS}$ | $ACS_{NoLS}$ | $EACS_{NoLS}$ |
| R.200.*.* | 43.36 | 69.38 | 37.08 | 54.22 | 182721 | 1003445 |
| R.300.*.* | 101.28 | 135.03 | 86.93 | 115.20 | 66887 | 567957 |
| R.400.*.* | 180.92 | 239.35 | 161.42 | 203.79 | 34419 | 381810 |
| R.500.*.* | 354.32 | 480.12 | 317.36 | 423.35 | 22989 | 269501 |
| R.600.*.* | 972.04 | 1310.45 | 884.54 | 1178.18 | 16221 | 211691 |
| R.700.*.* | 1430.92 | 1950.37 | 1254.06 | 1663.41 | 11953 | 151956 |
| Average | 513.80 | 697.45 | 456.90 | 606.36 | 55864.94 | 431060.10 |

**Comparison with the state-of-the-art, and new best known solutions.** The comparison is presented in Table 5.2. For the best known results we indicate the method used to obtain the result, and the cost of the solution (cost). For ACS and EACS we present the average and the best result obtained over 10 runs, where a maximum computation time of 600 seconds for each run is imposed. We also report in the last column the overall best results obtained during the parameter tuning campaign by the EACS algorithm (again with a maximum computation time of 600 seconds). Bold entries indicate the lowest average and best values for each line of the table.

The results of Table 5.2 give clear indications: over the 48 instances considered, EACS was able to improve 45 average results and 43 best results of ACS, never being worse than ACS. When compared with the best-known results, EACS was able to improve 32 of them, while matching the best known results in the remaining 16 cases.

## 5.5 Conclusions

In this chapter two main directions for improving Ant Colony System when a strong local search routine is available have been presented. With respect to the original framework - which performs better and is still the reference one in case a strong local search is not available - the Enhanced Ant Colony System better exploits the presence of the given local search, that itself already provides a strong intensification: the intensification

Table 5.2: The Sequential Ordering Problem. Computational results.

| Instance | Best known | | ACS | | EACS | | EACS |
|---|---|---|---|---|---|---|---|
| | method | cost | avg | best | avg | best | overall best |
| R.200.100.1 | DPSO | 64 | 90.3 | 88 | **67.2** | **63** | **63** |
| R.200.100.15 | DPSO | 1799 | 2066.0 | 2002 | **1818.3** | **1792** | **1792** |
| R.200.100.30 | HMT | **4216** | 4254.6 | 4247 | **4216.0** | **4216** | **4216** |
| R.200.100.60 | ACS | **71749** | **71749.0** | **71749** | **71749.0** | **71749** | **71749** |
| R.200.1000.1 | DPSO | 1414 | 1549.5 | 1532 | **1432.5** | **1411** | **1411** |
| R.200.1000.15 | DPSO | **20481** | 22602.9 | 21775 | **20717.0** | **20481** | **20481** |
| R.200.1000.30 | HMT | **41196** | 41371.6 | 41278 | **41196.0** | **41196** | **41196** |
| R.200.1000.60 | ACS | **71556** | **71556.0** | **71556** | **71556.0** | **71556** | **71556** |
| R.300.100.1 | DPSO | 31 | 76.4 | 74 | **34.6** | 31 | **30** |
| R.300.100.15 | DPSO | 3167 | 3738.6 | 3520 | **3207.1** | 3162 | **3161** |
| R.300.100.30 | DPSO | **6120** | 6228.2 | 6151 | **6120.0** | **6120** | **6120** |
| R.300.100.60 | ACS | **9726** | **9726.0** | **9726** | **9726.0** | **9726** | **9726** |
| R.300.1000.1 | DPSO | 1338 | 1586.7 | 1536 | **1369.6** | **1331** | **1331** |
| R.300.1000.15 | DPSO | 29475 | 34447.9 | 33533 | **29784.4** | 29248 | **29183** |
| R.300.1000.30 | DPSO | **54147** | 55013.4 | 54367 | **54172.6** | **54147** | **54147** |
| R.300.1000.60 | ACS | **109471** | 109530.5 | **109471** | **109471.0** | **109471** | **109471** |
| R.400.100.1 | DPSO | 21 | 64.1 | 59 | **22.6** | 21 | **17** |
| R.400.100.15 | DPSO | 3946 | 5087.1 | 4838 | **3986.2** | 3925 | **3906** |
| R.400.100.30 | DPSO | **8165** | 8476.5 | 8289 | **8165.9** | **8165** | **8165** |
| R.400.100.60 | ACS | **15228** | 15232.4 | **15228** | **15228.0** | **15228** | **15228** |
| R.400.1000.1 | DPSO | 1484 | 1811.1 | 1783 | **1475.5** | 1456 | **1419** |
| R.400.1000.15 | DPSO | 40054 | 46638.6 | 45055 | **40122.9** | 39612 | **29685** |
| R.400.1000.30 | DPSO | 85221 | 85979.6 | 85579 | **85203.3** | 85192 | **85132** |
| R.400.1000.60 | HMT | **140816** | 140994.9 | 140862 | **140816.0** | **140816** | **140816** |
| R.500.100.1 | DPSO | 14 | 55.0 | 51 | **16.1** | 11 | **8** |
| R.500.100.15 | DPSO | 5525 | 6931.5 | 6584 | **5507.8** | 5431 | **5361** |
| R.500.100.30 | DPSO | 9683 | 10333.2 | 10047 | **9668.1** | **9665** | **9665** |
| R.500.100.60 | HMT | **18240** | 18260.4 | 18246 | **18247.4** | **18240** | **18240** |
| R.500.1000.1 | DPSO | 1514 | 1877.4 | 1840 | **1522.5** | 1501 | **1436** |
| R.500.1000.15 | DPSO | 51624 | 62693.7 | 60175 | **51763.0** | 51091 | **50880** |
| R.500.1000.30 | DPSO | 99181 | 101751.8 | 100453 | **99112.0** | 99018 | **98987** |
| R.500.1000.60 | HMT | **178212** | 178478.1 | 178323 | **178212.0** | **178212** | **178212** |
| R.600.100.1 | DPSO | 11 | 49.8 | 44 | **9.4** | 6 | **3** |
| R.600.100.15 | DPSO | 5923 | 7806.6 | 7610 | **5881.7** | 5798 | **5684** |
| R.600.100.30 | DPSO | 12542 | 13001.8 | 12810 | **12475.5** | **12465** | **12465** |
| R.600.100.60 | DPSO | **23293** | 23357.4 | 23342 | **23293.0** | **23293** | **23293** |
| R.600.1000.1 | DPSO | 1628 | 1986.7 | 1936 | **1598.9** | 1534 | **1521** |
| R.600.1000.15 | DPSO | 59177 | 72701.1 | 70454 | **58281.6** | 57812 | **57387** |
| R.600.1000.30 | DPSO | 127631 | 132314.3 | 130244 | **126961.7** | **126789** | **126789** |
| R.600.1000.60 | HMT | **214608** | 214970.2 | 214724 | **214608.0** | **214608** | **214608** |
| R.700.100.1 | DPSO | 9 | 42.6 | 41 | **7.9** | 5 | **2** |
| R.700.100.15 | DPSO | 7719 | 9573 | 9383 | **7444.0** | 7380 | **7331** |
| R.700.100.30 | DPSO | 14706 | 15905.8 | 15733 | **14520.0** | 14513 | **14510** |
| R.700.100.60 | DPSO | 24106 | 24192.3 | 24151 | **24172.0** | **24102** | **24102** |
| R.700.1000.1 | DPSO | 1606 | 1969.2 | 1912 | **1614.8** | 1579 | **1461** |
| R.700.1000.15 | DPSO | 72618 | 85177.7 | 81439 | **68630.0** | 67510 | **66837** |
| R.700.1000.30 | DPSO | 136031 | 141557.9 | 139769 | **134651.2** | **134474** | **134474** |
| R.700.1000.60 | DPSO | **245589** | 246489.6 | 246128 | **245684.0** | 245632 | **245589** |

capabilities of the original constructive phase of the Ant Colony System are traded with a computational speed-up. Moreover, some mechanisms to prevent the local search to be run on non promising solutions, are introduced. As a result, the enhanced method is shown to improve the original one, leading to better solutions.

The new enhanced framework has been implemented for some known combinatorial optimization problems: the Team Orienteering Problem with Time Windows, the Sequential Ordering Problem and the Probabilistic Traveling Salesman Problem. Experimental results reported in this Chapter for the Sequential Ordering Problem suggest that the enhanced algorithms are able to outperform the original one in many occasions, leading to many new best-known results. This provides an experimental validation of the new proposed ideas.

Possible further research directions consist in implementing the enhanced framework for other combinatorial optimization problems: there are many problems for which a strong dominant local search is known. The application of the Enhanced Ant Colony System framework to these cases might potentially lead to extremely effective algorithms.

# Chapter 6

# Ant Colony Optimization for Real-World Vehicle Routing Problems: From Theory To Applications

## 6.1 Introduction

In this chapter (Rizzoli et al. 2007) we report on the successful application of ant colony optimization to the real-world vehicle routing problem (VRP). First, we introduce the VRP and some of its variants, such as the VRP with time windows, the time dependent VRP, the VRP with pickup and delivery, and the dynamic VRP. These variants have been formulated in order to bring the VRP closer to the kind of situations encountered in the real-world.
Then we briefly present the application of ant colony optimization to the solution of the VRP and of its variants.
Last, we discuss the applications of ACO to a number of real-world problems: a VRP with time windows for a major supermarket chain in Switzerland; a VRP with pickup and delivery for a leading distribution company in Italy; a time dependent VRP for freight distribution in the city of Padua, Italy, where the travel times depend on the time of the day; and an on-line VRP in the city of Lugano, Switzerland, where customers orders arrive during the delivery process. In all these applications ACO has been successfully coupled with dedicated local searches

in Section 1.5.
The vehicle routing problem (VRP) concerns the transport of items between depots and customers by means of a fleet of vehicles. Examples of VRPs are: milk delivery, mail delivery, school bus routing, solid waste collection, heating oil distribution, parcel pick-up and delivery, dial-a-ride systems, and many others. Although finding the most cost efficient way to distribute goods across the logistic network is the main objective of supply-chain systems, only in the early '90s enterprise resource planning software vendors started to integrate tools to solve the VRP in supply chain management software (a review of software for supply chain management can be found in Aksoy & Derbez (2003)).

The practical interest of the VRP has spawn a number of studies, which tackled the problem from many sides. Yet, the VRP is combinatorially complex, and therefore, as the size of the problem increases, it becomes harder and harder to obtain an exact solution for it in a reasonable amount of time. Thus, even the most advanced exact solution methods impose particular constraints on the problem instance, which are often violated when dealing with real-world vehicle routing problems, leaving practitioners unsatisfied with the performance and applicability of the algorithms.

Given the shortcomings of exact solution methods, researchers in the field of operations research (OR) started to develop *metaheuristics* (Blum & Roli. 2003), heuristic methods that can be applied to a wide class of problems. One of the advantages metaheuristics have over traditional optimization algorithms is their ability to produce a good suboptimal solution in short time. The integration of optimization algorithms based on metaheuristics, such as tabu search (Glover & Laguna 1997), simulated annealing (Kirkpatrick et al. 1983), ant colony optimization (Chapter 2, Dorigo & Gambardella 1997), and iterated local search (Lourenço et al. 2003), with advanced logistic systems for supply chain management opened new perspectives for operations research applications in industry. In particular, for the solution of VRP and its variants, a number of metaheuristics have been successfully applied, such as: simulated annealing (Osman 1993), tabu search (Gendreau et al. 1994; Taillard et al. 1997), granular tabu search (Toth & Vigo 2003), genetic algorithms (Van Breedam 1996), guided local search (Kilby et al. 1999), variable neighbourhood search (Bräysy 2003), greedy randomized adaptive search procedure (Resende & Ribeiro 2003), and Ant Colony Optimization (Chap-

ter 3, Gambardella et al. 1999; Reimann et al. 2003).

Ant Colony Optimization has been also used for the approximate solution of a number of traditional OR problems, among which the job shop scheduling problem, the quadratic assignment problem, the sequential ordering problem (Chapter 4), the graph coloring problem, and the shortest common supersequence problem by Dorigo & Stützle (2004). ACO has been also employed in a number of open shop scheduling problems (Blum 2005), in optimal product design (Albritton & McMullen 2007), and has also been used in some environmental problems, such as the design of a water distribution network (Zecchin et al. 2007) or the planning of wells for groundwater quality monitoring (Li & Chan Hilton 2007), thus proving its adaptability to very different domains of application.

The flexibility of the ACO metaheuristic allowed its application to many vehicle routing problems where heterogeneous vehicle fleets, limitations on customer accessibility, time windows and the order imposed by pick-ups and deliveries considerably complicate the problem formulation. These kinds of problems have been labelled as *rich* vehicle routing problems Hartl et al. (2006). Yet, real-world problems are even more complex; for instance, travel times may be uncertain and depend on traffic conditions, and not all customers' orders may be perfectly known in time and dimension. These problem variants have been called *dynamic* VRP and they are currently attracting a lot of research efforts, because of their closeness to real-world traffic and distribution models (Zeimpekis et al. 2007). The objective of this chapter is to describe how ant colony optimization can be successfully used to solve a number of VRP variants, both for some of the basic problem instances (the capacitated VRP, the VRP with time windows, the VRP with pickup and delivery) and for some of the dynamic extensions (the time dependent vehicle routing problem and the on-line vehicle routing problem) where ACO has been applied and its ability to find efficient solutions in a short time has been proven useful also in this setting.

The chapter is structured as follows: Section 6.2 outlines the VRP with its static and dynamic variants. Then Section 6.3 is devoted to real world applications: here we first introduce two large-scale industrial applications, showing how ACO can be successfully applied in the day-to-day operations of large real-world distribution processes, then we present the application of ACO to dynamic VRPs, showing its applicability in the context of urban freight distribution.

## 6.2 Vehicle Routing Problems

Finding optimal routes for a fleet of vehicles performing assigned tasks on a number of spatially distributed customers can be formulated as a combinatorial optimization problem: the vehicle routing problem. A solution of this problem is the best route serving all customers using a fleet of vehicles, respecting all operational constraints, such as vehicle capacity and the driver's maximum working time, and minimising the total transportation cost.

The algorithm solving the problem requires defining an objective function that may include multiple objectives that are often conflicting. The most common objective is the minimisation of transportation costs as a function of the travelled distance or of the travel time; the number of vehicles can be minimised expressing the costs associated with vehicles and drivers (Section 6.2.1). Vehicle efficiency, expressed as the percentage of load capacity weighted by distance, can be taken into account. "Soft" constraints, which can be violated paying a penalty, can be included. For instance, if a customer is not served according to the agreed time schedule, a penalty might have to be paid. Road pricing schemes can also be considered, for example attributing a higher cost to routes through city centres.

The elements that define and constrain each model of the VRP are: the *road network*, describing the connectivity among customers and depots; the *vehicles*, transporting goods between customers and depots on the road network; the *customers*, which place orders and receive goods.

The road network graph can be obtained from a detailed map of the distribution area on which the depots and the customers are georeferenced. Standard algorithms can then be used to find the shortest routes between all couples of nodes in order to build the travel cost matrix. The matrix coefficients can represent the time required to travel from node $i$ to node $j$, or the distance between the nodes, or any other metric that measures the travel cost. According to the adopted metric, different instances of the VRP may arise. For instance, if the travel time on edges depends on the time of the day, then we encounter the time dependent VRP (see paragraph 6.3.3 below).

The fleet of vehicles and their characteristics also impose constraints on the vehicle routing model. The fleet can be homogeneous, if all vehicles are equal in their characteristics, heterogeneous if this is not the case. Most real-world fleets are heterogeneous. Mechanical features (length,

weight, width) and configuration (trailer, semi-trailer, van, etc.) constrain the ability of a vehicle to access road segments. For instance, a vehicle cannot travel on some arcs of a road network because of excessive weight or dimensions. On-board equipment, such as loading/unloading devices, may also impose constraints that depend on the type of customer to be served. Capacity constraints, stating the maximum load to be transported by a vehicle, are also important.

Each customer requests a given amount of goods, an order, which must be delivered or collected at the customer location. Time windows within which the customer must be served can be specified. These time windows can be single (only one continuous interval) or multiple (disjoint intervals). If time windows cannot be violated at any cost they are said to be "hard"; on the other hand, when a penalty is paid in case of violation, time windows are said to be "soft". Finally, the vehicle routing model can also include an estimation of the loading and unloading times at the customer – the so-called service time.

### 6.2.1 Basic Problems of the Vehicle Routing Class

Combining the various elements of the problem, we can define a whole family of different VRPs. Toth & Vigo (2001b) present a detailed overview of the various VRPs.

The capacitated vehicle routing problem (CVRP, Section 3.2) is the basic version of the VRP. The name derives from the constraint of having vehicles with limited capacity. Customer demands are deterministic and known in advance. Deliveries cannot be split, that is, an order cannot be served using two or more vehicles. The vehicle fleet is homogeneous and there is only one depot. The objective is to minimise the total travel cost, usually expressed as the travelled distance required to serve all customers. The CVRP is NP-hard (Labbé et al. 1991) and the size of the problems which can be solved exactly in a reasonable time is up to 50 customers, using the branch-and-bound, branch-and-cut, and set-covering approaches (see Toth & Vigo 2001a).

When constraints on the delivery times are present, we have a vehicle routing problem with time windows (VRPTW): the capacity constraint still holds and each customer $i$ is associated with a time window $[b_i, e_i]$ and with a service time $s_i$. VRPTW is also NP-hard, and even finding a feasible solution to the VRPTW is an NP-hard problem (Savelsbergh 1985). Good overviews on the VRPTW formulation and on exact,

heuristic, and metaheuristic approaches to its solution can be found in Mester & Bräysy (2005), Li et al. (2005) and in Kytöjoki et al. (2007). Kallehauge et al. (2006) have solved VRPTW problems with 400 and 1000 customers by Lagrangian relaxation, but the problem formulation requires hard time windows.

In the VRP with pick-up and delivery (VRPPD) the transport items are not originally concentrated in the depots, but they are distributed over the nodes of the road network. A transportation request consists in transferring the demand from the pick-up point to the delivery point. These problems always include time windows for pick-up and/or delivery. A review of various approaches to the solution of the VRPPD is presented in Desaulniers et al. (2000).

In all of these approaches the problem data is supposed not to change, neither during the planning phase (computation of the solution), nor during the management phase (implementation of the solution). More realistic situations might require the relaxation of this assumption.

### 6.2.2 Dynamic Extensions of the VRP

The static formulations of the VRP, where customer demand is deterministic and travel times do not depend on the time of the day, have proven to be successful in modeling many practical problems. This is especially true for the VRPTW and VRPPD extensions. Yet, the availability of online information on the traffic conditions and the possibility of monitoring the vehicles' positions via the global positioning system, together with the online update of customer orders, can considerably change the problem settings. The availability of this information has a price: the assumption of time-invariancy must be relaxed and data become time-dependent. Moreover, using data on current traffic conditions to estimate travel times requires the relaxation of the assumption of determinism, introducing uncertainty and adding another level of complexity to the problem.

In the literature these problems have been labelled with different terms: probabilistic, dynamic, and stochastic vehicle routing. These terms are often interchanged, but in practice we can assume that in general we are confronted with a dynamic vehicle routing problem, where problem data are often generated by a stochastic process. For instance, a stochastic process can be assumed to be responsible for the presence or absence of the customers, for the quantity of their orders, and for the

travel and service times (Laporte & Louveaux 1998).

Stochastic customers and demands are typically formed when the planning horizon is longer than the horizon of the currently available data. These kinds of problems have been extensively studied (see for instance Gendreau et al. (1996), Bianchi et al. (2004)) also studied how various metaheuristics can used in the solution of the VRP with stochastic demands.

One approach to solve the problem of unknown orders is to remove uncertainty by processing new orders as they come, in batches of variable size, according to what is called a *reactive* strategy. Potvin et al. (2006) list a number of reactive dynamic strategies for vehicle routing and scheduling problems; in the case of VRP this has been called the online variant (OLVRP). Among possible applications of the OLVRP we find *feeder systems*, which typically are local dial-a-ride systems aimed at feeding another, wider area, transportation system at a particular transfer location (Gendreau & Potvin 1998; Psaraftis 1988, 1995).

Another side of the problem is the presence of stochastic travel and service times, which are very frequent in urban environments. Especially with respect to travel times, the variability can be very high and considerably affect the solution. In the time dependent VRP variant (TDVRP) the variability can be reduced if one assumes that travel times are nearly constant within time periods in a day. This is quite true for peak and off-peak traffic conditions, which are observed in most cities. Ichoua et al. (2003) present a structured introduction to the problem and a model formulation.

## 6.3 Solving the VRP with ACO

Sales and distribution processes require the ability to forecast customer demand and to optimally plan the distribution of the products to the consumers. These two strategic activities, forecast and optimization, must be tightly interconnected in order to improve the performance of the system as a whole (Gambardella et al. 1998).

In Figure 6.1 the workflow of a distribution-centred company is sketched. The sales department generates new orders by contacting the customers (old and new ones) to check whether they need a new delivery. The effectiveness of this operation can be increased thanks to inventory management modules, which estimate the demand of every customer, indi-

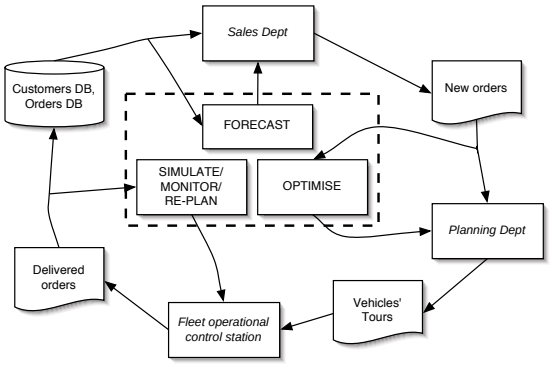


Figure 6.1: The Forecast-Optimise-Simulate loop: the role of optimisation in the efficient management of a distribution process.

cating the best re-order time for each of them. New orders are then processed by the planning department, which, according to the quantities requested, the location of the customers, and the time windows for the delivery, decides how many vehicles to employ and computes the best routes for the delivery, in order to minimise the total travel time and space. This task is assisted by a vehicle routing algorithm, represented by the OPTIMIZE block. The vehicle tours are then assigned to the fleet, which is monitored by the fleet operational control station, which monitors the evolution of deliveries in real time. This process is assisted by the SIMULATE/MONITOR/RE-PLAN module, which allows re-planning online in face of new urgent orders, which were not available during the previous off-line planning phase. Finally, after vehicles have returned to the depot, delivery data are off-loaded and transferred back to the company database.

In the next sub-sections, we describe a number of real-world applications, where ACO has been used for the implementation of the vehicle routing algorithms that are executed in the OPTIMIZE block of Figure 6.1.

### 6.3.1 A VRPTW Application for the Secondary Level in the Supply Chain

In this application the client is one of the major supermarket chains in Switzerland. The problem is to distribute palletized goods to more than 600 stores, all over Switzerland. The stores order daily quantities of goods to replenish their local stocks. They want the goods to be delivered within time windows, in order to plan in advance the daily availability of their personnel, allocating a fraction of their time to inventory management tasks.

There are three types of vehicles: trucks (capacity: 17 pallets), trucks with trailers (35 pallets), and tractor units with semi-trailers (33 pallets). Whether a vehicle can access a store or not depends on the store location. In some cases the truck with trailer can leave the trailer at a previous store and then continue to other less accessible locations. The number of vehicles is assumed to be infinite, since transport services can be purchased on the market according to the needs.

The road network graph has been computed using digital road maps. The distances in kilometres between couples of stores have been rescaled using a speed model, which depends on the distance: longer distances allow a higher average speed. For instance, if the distance is less than 5 km, the average speed is 20 km/h; if the distance is more than 90 km, the speed is 60 km/h; in between there is a range of other speed values. The data have been collected over many years and they have been validated by the drivers' experience. The time to set-up a vehicle for unloading and the time required to hook/unhook a trailer are constant. The service time is variable and depends on the number of pallets to unload.

All the routes must be performed in one day, and the client imposes an extra constraint stating that a vehicle must perform its latest delivery as far as possible from the inventory, since it could be used to perform extra services on its way back. These extra services were not included in the planning by explicit request of the client.

#### The algorithm.

This problem was modeled as a VRPTW, and solved by an implementation of the MACS-VRPTW algorithm (Chapter 3, Gambardella et al. 1999), named ANTROUTE. MACS-VRPTW is the most efficient ACO algorithm for the VRPTW and one of the most efficient metaheuristics

overall for this problem. ANTROUTE adds to MACS-VRPTW the ability to handle the choice of the vehicle type: at the start of each tour the ant chooses a vehicle. A waiting cost was also introduced in order to prevent vehicles arriving too early at the stores.

The central idea of the MACS-VRPTW algorithm (Section 3.3) is to use two ant colonies (MACS stands for multi ant colony system). One colony, named ACS-VEI (Algorithm 16), minimizes the vehicles while the other one, named ACS-TIME (Algorithm 15), minimizes time. The two colonies are completely independent, since each one has its own pheromone trail, but they collaborate by sharing the variable $L_{gb}$, which describes the best solution found so far (Algorithm 14). Each colony is composed of a number of ants. Every ant in the colony tries to build a feasible solution to the problem.

Note that in the ACS-VEI colony the ants usually construct infeasible solutions, that is, not all customers can be visited under the constraints given.

Given that the algorithm uses two colonies, at the end of each cycle, pheromone trails are globally updated for two different solutions: $L_{ACS-VEI}$, the infeasible solution with the highest number of visited customers, and $L_{gb}$, the feasible solution with the lowest number of vehicles and the shortest travel time. Thus, pheromone is updated also on arcs that are not included in a feasible solution, which could still become feasible in the next iteration.

ACS-TIME is coupled with a local search procedure to improve the quality of the feasible solutions, which is similar to the CROSS procedure of Taillard et al. (1997). Given that the CROSS procedure requires a feasible solution to operate on, such a local search can not be applied during ACS-VEI, where unfeasible solutions can still be returned at the end of a cycle. Both ACS-TIME and ACS-VEI try to repair infeasible solutions by inserting unvisited customers.

**Results.**

The first tours computed by ANTROUTE were not accepted as feasible by the human tour planners, even if the performance was considerably higher than theirs and no explicit constraints were violated. Thus, a further modelling iteration was required, to let "hidden" constraints emerge. The human planners were actually using a regional planning strategy, that led to petal shaped tours. This preference was included in the re-

Table 6.1: Comparison of the computer-generated vs. man-made tours in the VRPTW application

| | Human Planner | AR-RegTW | AR-Free |
|---|---|---|---|
| Total number of tours | 2056 | 1807 | 1614 |
| Total km | 147271 | 143983 | 126258 |
| Average truck loading | 76.91% | 87.35% | 97.81% |

formulation of the problem, but at the same time we tried to loosen the constraint. We attributed stores to distribution regions, but at the same time we allowed stores near the border of the distribution region to also belong to the neighbouring region. This allowed the generation of tours which were slightly worse than the unconstrained solution, but nevertheless better than the solutions found by the human planners. In Table 6.1 we present the results obtained by ANTROUTE compared with those of the human planners. ANTROUTE was run under two configurations: AR-RegTW, with regional planning and 1-hour time windows; AR-Free, where the regional and the time windows constraints were relaxed. The problem was to distribute 52000 pallets to 6800 customers over a period of 20 days. Every day ANTROUTE was run on the available set of orders and it took about 5 minutes to find a solution. At the same time, the planners were at work and it took them at least 3 hours to find a solution. At the end of the testing period, the performances of the algorithm and of the planners have been compared using the same objective function.

The advantage of an algorithm able to find the solution to an otherwise very hard problem in such a short time is the possibility of using it as a *strategic planning tool*. In Figure 6.2 it is shown how running the algorithm with wider time-windows at the stores returns a smaller number of tours, which can be translated in a substantial reduction of transportation costs. The logistic manager can therefore use the optimization algorithm as a tool to investigate how to re-design the time-windows in the stores.

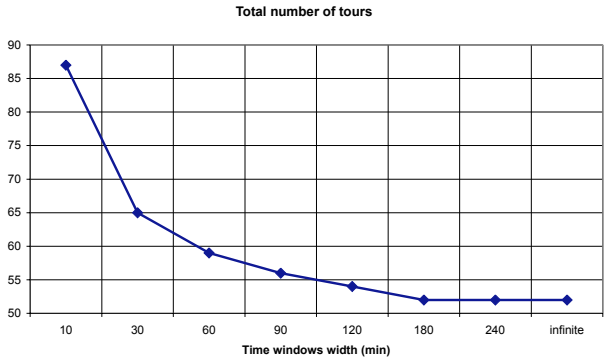


Figure 6.2: The relationship between the time window width and the number of vehicle routes.

### 6.3.2 A VRPPD Application for the Primary Level in the Supply Chain

In this application the client is a major logistics operator in Italy. The distribution process involves moving palletized goods from factories to inventory stores, before they are finally distributed to shops. A customer in this vehicle routing problem is either a pick-up or a delivery point. There is no central depot, and approximately 1000 – 1500 trucks per day are used. Routes can be performed within the same day, over two days, or over three days, since the Italian peninsula is quite long and there's a strict constraint on the maximum number of hours per day that a driver can travel. All pick-ups of a tour must take place before deliveries. Orders cannot be split among tours. Time windows are associated with each store.

There is only one type of truck: tractor with semi trailer. The load is measured in pallets, in kilograms, and in cubic metres. There are capacity constraints on each one of these measurement units, and the first one that is exceeded triggers the violation of the constraint. The availability of vehicles is assumed to be infinite, since they are provided by flexible sub-contractors. Sub-contractors are distributed all over Italy, and therefore vehicles can start their routes from the first assigned customer, and no cost is incurred in travelling to the first customer in the route.

The road network graph has been elicited from digital road maps, computing the shortest path between each couple of stores. The travel

times are computed according to the travelled distance, given the average speed that can be sustained on each road segment according to its type (highway, extraurban road, urban road). Loading and unloading times are assumed to be constant. This is a rough approximation imposed by the client, since they have been unable so far to provide better estimates. The client also imposed another constraint, related to the same problem, setting a maximum number of cities to visit per tour (usually less than six). Note that more than one customer can reside in a city. Moreover, the client requested that the distance between successive deliveries should be limited by a parameter.

**The algorithm.**

The problem can be modeled as a VRP with pickup and delivery and time windows (VRPPDTW). The objective function measures the average tour efficiency, $f = \frac{\sum_{i=1}^{N} e_i}{N}$, where $e_i$ is the efficiency of tour $i$: the occupancy ratio of a vehicle over the travelled distance within the tour. It is computed according to the formula $e_i = \frac{\sum_{j=1}^{M_i} q_j l_j}{Q_i L_i}$, where $M_i$ is the number of orders in the $i$-th tour; $q_j$ is the number of pallets in the $j$-th order; $l_j$ is the distance between source and destination points of the $j$-th order; $Q_i$ is the capacity of the vehicle serving the $i$-th tour, and $L_i$ is the total length of the $i$-th tour.

The ANTROUTE algorithm has also been used in this context, but since there is a single objective – to maximise average efficiency – it has been adapted removing the ant colony minimising the number of vehicles. The first step of an artificial ant is to select the starting city. Since this is a pickup and delivery problem, each source node must be paired with the corresponding destination node, and the search space is therefore harder to explore than in a delivery problem. The algorithm tries to simplify exploration using an approximation of the delivery phase, assuming that all deliveries will be performed in the reverse order with respect to pickups. Thus, a first stage local search exchanges nodes between tours, while preserving the order of deliveries; later, another local search procedure is applied, in which nodes are exchanged within the same tour.

Table 6.2: Comparison of the computer-generated vs. man-made tours in the VRPPDTW application

| | Human Planner | ANTROUTE | Absolute difference | Relative difference |
|---|---|---|---|---|
| Total nr of tours | 471.5 | 460.8 | -10.7 | -2.63% |
| Total km | 175441 | 173623 | -1818.2 | -1.32% |
| Efficiency | 84.08% | 88.27% | +4.19% | - |

**Results.**

Table 6.2 summarises the comparison between man-made and computer-generated tours over a testing period of two weeks. A noticeable improvement in the efficiency of computer-generated tours can be observed.

It is also interesting to remark that the algorithm performance is correlated with the difficulty of the problem, which is related to the number of orders to satisfy. In Figure 6.3 we plot on the x-axis the efficiency of the man-made tours, and on the y-axis the efficiency improvement obtained using the computer-generated tours. When the problem is easy, because it involves a limited number of orders, and the human planner performs well, the computer is not able to provide a significative improvement, but when the planner starts to fail coping with the problem complexity, and the performance decreases, the gain in using the algorithm sensibly increases.

### 6.3.3 Time Dependent VRPTW in the City of Padua

The city of Padua, Italy, set up a logistic platform to collect all incoming goods to be distributed to a number of shops in the city centre. Such a platform aims at a better organisation of the flow of goods into the city centre, which is affected by traffic congestion problems and where loading and unloading space is scarcely available. Centralised planning of vehicle routes can sensibly reduce pollution and traffic problems due to commercial transport. For this purpose, Donati et al. (2008) have developed an algorithm solving the time dependent VRPTW for a logistic platform serving the city of Padua. In the study, the central depot was open from 8 am to 6 pm and traffic data on the Padua road network during that period was collected. Four time intervals, with similar traffic

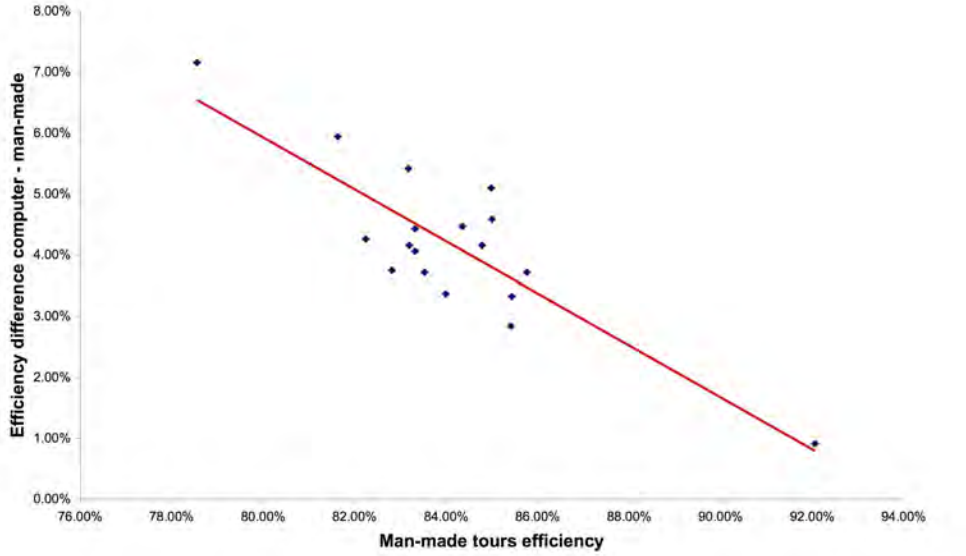


Figure 6.3: Comparing man-made and computer-generated tours. Higher efficiency improvements are observed when the human planner performance is lower. The dots are experimental values, and the solid line is a regression on those values.

patterns and the relative travel speeds on network arcs, were identified. A set of 30 customers was considered.

**The algorithm.**

The basic idea is the ACO algorithm of Donati et al. (2008). is to define a pheromone trail that is time dependent assuming that the travel times over the arcs of the graph depend on the time of the day. While the variation of travel times over time is continuous, it can be assumed that there are some distinctive time slices during one day when they are roughly constant. It is assumed that the duration of one working day can be partitioned in $l$ time slices, and therefore the pheromone trails can be described by $\tau_{ij}(l)$, with $l \in T_l$, where $T_l$ is the set of time slices into which the working time horizon is split. The objective is to minimise the total travel time.

The algorithm, based on MACS-VRPTW (see Chapter 3), then builds a solution making a probabilistic choice to select the next node $j$ starting from $i$ using the standard Equation 2.4. The attractiveness $\eta_{ij}$ of the next

node is given by:

$$\eta_{ij}(t) = \frac{1}{f_{ij}(t) + w_j} \tag{6.1}$$

where $f_{ij}(t)$ is the travel time from $i$ to $j$ evaluated at time $t \in T_l$ and $w_j$ is the waiting time at node $j$.

Pheromone updating is carried out as described in Chapter 3, independently for the pheromone corresponding to each time slice.

**Results.**

Donati et al. (2008) compared the solution of the VRPTW using the time dependent variant with a solution of the same problem where the travel times on the road arcs were constant, depending only on the distance. In a series of nine tests, where customers were chosen randomly out of a set of real customers, it turned out that the time dependent variant performed 7% better than the standard VRPTW algorithm.

### 6.3.4 On-line VRP for Fuel Distribution

A leading Swiss fuel oil distribution company, which serves its customers from its main depot located near Lugano with a fleet of 10 vehicles, observed that during every Winter season there was always a subset of their customers that ran out of fuel and had to place urgent orders. These unexpected orders affects the planned delivery routes of the vehicles, and the vehicle routing problem becomes very "dynamic", since a noticeable percentage of orders must be fulfilled after the vehicles have already left the depot.

The objective of this study was to evaluate the impact of a reactive strategy for vehicle routing, starting from an analysis of the data collected in periods when urgent deliveries were in high demand. From the company data base, a sample of 50 customers was randomly selected, and travel times among them were calculated. In the company records, customers randomly appeared during the working day with random requests for a quantity of fuel to be delivered. A working day of 8 hours was considered, assuming a service time of 10 minutes for each customer. The cut-off time, after which the new orders received were postponed to the following working day, was set to 4 hours.

**The algorithm.**

The problem description above matches the on-line VRP variant, where new orders can be assigned to vehicles which have already left the depot (e.g., parcel collection, feeder systems, fuel distribution, etc.).

To solve the on-line fuel oil distribution problem, (Montemanni et al. 2005) have developed an ACO-inspired algorithm, ACS-DVRP, based on the decomposition of the on-line VRP into a sequence of static VRPs. There are three main elements in the algorithm architecture: the event manager, the ant colony algorithm, and the pheromone conservation strategy.

The event manager receives new orders and keeps track of the already served orders and of the position and the residual capacity of each vehicle. This information is used to construct the sequence of static VRP-like instances. The working day is divided into time slices and for each of them a static VRP, which considers all the already received (but not yet executed) orders, is created. New orders received during a time slice are postponed until its end. At the end of each time slice, customers whose service time starts in the next time slice (according to the solution of the last static VRP) are assigned to the vehicles. They will not be taken into account in the following static VRPs.

The ACS algorithm employed is based on the one described in Chapter 3 for the VRPTW. The single ant colony is in charge of minimizing the total travel time.

Finally, the pheromone conservation strategy is based on the fact that, once a time slice is over and the relative static problem has been solved, the pheromone matrix contains information about good solutions. As each static problem is potentially very similar to the next one, this information is passed on to the next problem (Guntsch & Middendorf 2001): if a couple of customers appears in both the previous and the current time slice, the pheromone on the arcs connecting two nodes is brought forward as a fraction of its value in the previous problem.

**Results.**

Algorithm ACS-DVRP was executed on a number of test problems, obtained varying the number $n_{ts}$ of time slices into which the working day was divided. As the size of each problem in a time slice increases as the length of the time slice decreases, the time $t_{acs}$ allocated to executing the

Table 6.3: Experimental results on the case study of Lugano.

| $n_{ts}$ | 200 | 100 | 50 | 25 | 10 | 5 |
|---|---|---|---|---|---|---|
| $t_{acs}$ | 144 | 288 | 576 | 1152 | 2880 | 5760 |
| $t_{ls}$ | 15 | 30 | 60 | 120 | 240 | 480 |
| Travel time | 12702 | 12422 | 10399 | 9744 | 10733 | 11201 |

ant colony system and the time $t_{ls}$ dedicated to local search improving the solution were adjusted accordingly. In particular the ratio between $t_{acs}$ and $t_{ls}$ was kept approximately equal to 10.

The first three rows of Table 6.3 define the settings of the experiments, that is, the values of parameters $n_{ts}$, $t_{acs}$ and $t_{ls}$. The fourth row shows the total travel time of the solutions found by the ACS-DVRP algorithm.

The results suggest that, for the case study analyzed, good values for $n_{ts}$ are between 10 and 50. In particular, 25 seems to be the best choice. Large values of $n_{ts}$ did not lead to satisfactory results because optimization was restarted too often, before a good local minimum could be reached. On the other hand, when $n_{ts}$ was too small, the system was not able to take advantage of information on new incoming orders.

### 6.3.5 Conclusions

In this chapter we have described how the ant colony optimization metaheuristic couple with local searches can be successfully used to solve a number of variants of the basic vehicle routing problem. We presented two industrial-scale applications of ACO for the solution of static VRP problems: a VRP with time windows and a VRP with pickup and delivery. We then focused our attention on two important dynamic variants of the VRP: the time dependent VRP, and the on-line VRP. Both these problems are receiving increasing attention due to their relevance to real world problems, in particular for distribution in urban environments.

In conclusion, after more than ten years of research, ACO has been shown to be one of the most successful metaheuristics for the VRP and its application to real-world problems demonstrates that it has now become a fundamental tool in applied operations research.

# Chapter 7

# Conclusions

Ant Colony Optimization (Section 1.4.4, Dorigo et al. 1999) is an optimization framework inspired by the observation, made by ethologists, that ants use *pheromone trails* to communicate information regarding the shortest paths to food. A moving ant lays some pheromone (in varying quantities) on the ground, thus marking a path with a trail of this substance. An isolated ant moves mostly randomly and when it detects a previously laid pheromone trail it can decide, with high probability, to follow it, thus reinforcing the trail with its own pheromone. The collective behaviour that results is a form of autocatalytic behaviour, where the more ants follow a trail, the more attractive it becomes to other ants. The process is thus characterized by a positive feedback loop, where the probability with which an ant chooses a path increases with the number of ants that have previously chosen the same path.

The above process inspired the ACO metaheuristic. The main elements are *artificial ants* (from now on simply ants), simple computational agents that individually and iteratively construct solutions to the problem, which has been modelled as a graph. Ants explore the graph by visiting nodes connected by edges. A problem solution is an ordered sequence of nodes. The search process is executed in parallel over several constructive computational threads. A dynamic memory structure, inspired by the pheromone laying process, which incorporates information on the effectiveness of previously obtained results, guides the construction process of each thread.

The AS (Section 2.2, Dorigo et al. 1996) was the first ACO algorithm to be proposed to solve TSPs. It is organized into two main stages: construction of a solution and updating of the pheromone trail. In AS

each ant builds a solution. An ant is in a given state and it computes a set of feasible expansions from it. The ant selects the move to expand the state, taking into account the following two values: the attractiveness $\eta_{ij}$ of the move, as computed by some heuristic indicating the *a priori* desirability of that move, and the pheromone trail level $\tau_{ij}$ of the move, indicating how useful it has been in the past to make that particular move; it therefore represents an *a posteriori* indication of the desirability of that move.

Once a solution has been obtained, pheromone trails are updated. First, the pheromone is evaporated on all arcs, in order to progressively forget bad solutions; then all ants deposit pheromone on the arcs which are part of the solutions they have just computed.

The basic AS principle is very interesting but the performance of the final system has not yet been very convincing for medium-sized TSPs. Therefore, the first research work presented in this thesis has been to propose two new ACO algorithms, called Ant-Q and Ant Colony System (ACS, Chapter 2, Gambardella & Dorigo 1995, 1996), which are much more high-performing and efficient.

Ant-Q and ACS introduce a new way to build solutions with choices which exploit the accumulated pheromone and choices which allows the exploration of new paths. The management of the pheromone in Ant-Q and ACS is also different from that in AS. Part of the pheromone evaporates once it has been used by ants and only the best solution is used to reinforce the pheromone. Together with a special initialization of the pheromone, these innovations also lead to high quality solutions for problems of medium size. The next step has been to tackle larger problems coupling ACS with specialized local search techniques (Section 2.7). This is an interesting combination since local search often suffers from the initialization problem and metaheuristics are usually too slow (Section 1.5). The resulting system has been able to compete with the best available TSP algorithms. Once the framework has been established, we then decided to apply it to different combinatorial optimization problems.

This gives rise to the Multiple Ant Colony System for the Vehicle Routing Problems with Time Windows (MACS-VRPTW, Chapter 3, Gambardella et al. 1999) and Hybrid Ant System for the Sequential Ordering Problem (HAS-SOP, Chapter 4, Gambardella & Dorigo 2000). MACS-VRPTW realizes a system composed of two artificial colonies able to minimize in parallel the number of vehicles used and the to-

tal distance travelled. MACS-VRPTW is coupled with a sophisticated local search for the VRPTW and has been able to compete with state-of-the-art algorithms and to discover the new best known solutions. SOP is an asymmetric TSP with precedence constraints between nodes. HAS-SOP addresses this problem by coupling ACS with a new local search (SOP-3-Exchange, Section 4.3) able to handle precedence constraints in constant time and it has been able to enhance several benchmarks in the literature.

When we decided to tackle problems with thousands of nodes we struggled with the fact that the computation time was mostly too high for the time required by ants to construct new solutions. We then defined the Enhanced Ant Colony System (Chapter 5, Gambardella et al. 2012), which combines ACS with local searches in a more efficient way. In particular, EACS exploits the presence of the best solution computed so far to guide the construction of new solutions and uses the local search in a more efficient way. EACS has been able to solve problems of large size effectively, and for the SOP in particular, new results of excellent quality have been reported in this thesis.

Finally, we have shown how these ACO-based methods can be used to solve routing problems in the real world by presenting case studies and real-life applications (Chapter 6, Rizzoli et al. 2007).

# Bibliography


Aksoy, Y. & Derbez, A. (2003). Software survey: Supply chain management. *OR/MS Today*, *30*(3), 1–13.

Albritton, M. & McMullen, P. (2007). Optimal product design using a colony of virtual ants. *European Journal of Operational Research*, *176*(1), 498–520.

Anghinolfi, D., Montemanni, R., Paolucci, M., & Gambardella, L. M. (2009). A particle swarm optimization approach for the sequential ordering problem. In Caserta, M. & Voss, S. (Eds.), *Proceedings of the VIII Metaheuristic International Conference (MIC 2009), Hamburg, Germany, July 13-16, 2009*.

Anghinolfi, D., Montemanni, R., Paolucci, M., & Gambardella, L. M. (2011). A hybrid particle swarm optimization approach for the sequential ordering problem. *Computers and Operations Research*, *38*(7), 1076–1085.

Ascheuer, N. (1995). *Hamilton Path Problems in the On-line Optimization of Flexible Manufacturing Systems*. PhD thesis, Technische Universität Berlin, Berlin, Germany.

Ascheuer, N., Escudero, L., Grötschel, M., & Stoer, M. (1993). A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). *SIAM Journal on Optimization*, *3*, 25–42.

Badeau, P., Gendreau, M., Guertin, F., Potvin, J.-Y., & Taillard, É. D. (1997). A parallel tabu search heuristic for the vehicle routing problem with time windows. *Transportation Research-C*, *5*, 109–122.

Balas, E., Fischetti, M., & Pulleyblank, W. (1995). The precedence constrained asymmetric traveling salesman polytope. *Mathematical Programming*, *65*, 24–265.

Baluja, S. & Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. In Prieditis, A. & Russell, S. (Eds.), *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, (pp. 38–46). Morgan Kaufmann Publishers, Palo Alto, CA.

Battiti, R. & Tecchiolli, G. (1994a). The reactive tabu search. *ORSA Journal on Computi ng*, *6*(2), 126–140.

Battiti, R. & Tecchiolli, G. (1994b). Simulated annealing and tabu search in the long run: A comparison on QAP tasks. *Computer and Mathematics with Applications*, *28*(6), 1–8.

Beckers, R., Deneubourg, J. L., & Goss, S. (1992). Trails and U-turns in the selection of the shortest path by the ant Lasius Niger. *Journal of Theoretical Biology*, *159*, 397–415.

Beckers, R., Deneubourg, J.-L., & Goss, S. (1993). Modulation of trail laying in the ant *Lasius niger* (hymenoptera: Formicidae) and its role in the collective selection of a food source. *Journal of Insect Behavior*, *6*(6), 751–759.

Bentley, J. L. (1992). Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, *4*(4), 387–411.

Bersini, H., Dorigo, M., Langerman, S., Seront, G., & Gambardella, L. M. (1996). Results of the first international contest on evolutionary optimisation. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, (pp. 611–615). IEEE Press, Piscataway, NJ.

Bersini, H. & F.Varela (1993). Hints for adaptive problem solving gleaned from immune networks. In TH.-P.Scwefel & R.Mnner (Eds.), *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, (pp. 343–354). Springer Verlag, Berlin, Germany.

Bersini, H., Oury, C., & Dorigo, M. (1995). Hybridization of genetic algorithms. Technical Report IRIDIA/95-22, IRIDIA, Université Libre de Bruxelles, Belgium.

Bianchi, L., Birattari, M., Chiarandini, M., Manfrin, M., Mastrolilli, M., Paquete, L., Rossi-Doria, O., & Schiavinotto, T. (2004). Metaheuristics for the vehicle routing problem with stochastic demands. In X. Y. et al. (Ed.), *Parallel Problem Solving from Nature – PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science* (pp. 450–460). Berlin, Germany: Springer.

Blum, C. (2005). Beam-ACO – Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers and Operations Research*, *32*(6), 1565–1591.

Blum, C., Aguilera, M. J. B., Roli, A., & Sampels, M. (Eds.). (2008). *Hybrid Metaheuristics, An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*. Springer.

Blum, C. & Dorigo, M. (2004). The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, *34*(2), 1161–1172.

Blum, C. & Roli., A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, *35*(3), 268–308.

Brady, R. (1985). Optimization strategies gleaned from biological evolution. *Nature*, *317*, 804–806.

Bräysy, O. (2003). A reactive variable neighborhood search for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, *15*(4), 347–368.

Bullnheimer, B., Hartl, R., & Strauss, C. (1999a). A new rank-based version of the ant system: a computational study. *Central European Journal of Operations Research*, *7*(1), 25–38.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999b). *Applying the Ant System to the Vehicle Routing Problem, in Meta-heuristics: Advances and Trends in Local Search for Optimization, S.Voss, S. Martello, I.H. Osman and C.Roucairol (eds.)*, (pp. 285–296). Kluwer Academic Publishers.

Butt, S. & Cavalier, T. (1994). A heuristic for the multiple path maximum collection problem. *Computers and Operations Research*, *21*, 101–111.

Chao, I.-M., Golden, B., & Wasil, E. (1996). The team orienteering problem. *European Journal of Operational Research*, *88*, 464–474.

Chen, S. & Smith, S. (1996). Commonality and genetic algorithms. Technical Report CMU-RI-TR-96-27, The Robotic Institute, Carnegie Mellon University, Pittsburgh, PA.

Chiang, W. C. & Russel, R. (University of Tulsa, OK, USA, 1993). Hybrid heuristics for the vehicle routing problem with time windows.

Clarke, G. & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, *12*, 568–581.

Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Gli algoritmi genetici e il problema dell'orario. *Rivista di Ricerca Operativa*, *60*, 5–31. In Italian.

Colorni, A., Dorigo, M., & Maniezzo, V. (1992). Distributed optimization by ant colonies. In Varela, F. J. & Bourgine, P. (Eds.), *Proceedings of the First European Conference on Artificial Life*, (pp. 134–142). MIT Press, Cambridge, MA.

Colorni, A., Dorigo, M., & Maniezzo, V. (1995). New results of an Ant System approach applied to the asymmetric TSP. In Osman, I. H. & Kelly, J. P. (Eds.), *Proceedings of the MIC-95 Metaheuristics International Conference, Breckenridge, CO, 22–26 July, 1995*, (pp. 356–360).

Connolly, D. (1990). An improved annealing scheme for the QAP. *European Journal of Operational Research*, *46*, 93–100.

Cordone, R. & Calvo, R. W. (2001). A heuristic for the vehicle routing problem with time windows. *Journal of Heuristics*, *7*(2), 107–129.

Deneubourg, J.-L. (1977). Application de l'ordre par fluctuations à la description de certaines étapes de la construction du nid chez les termites. *Insectes Sociaux*, *24*, 117–130.

Desaulniers, G., Desrosiers, J., Erdmann, A., Solomon, M., & Soumis, F. (2000). VRP with pickup and delivery. In P. Toth & D. Vigo (Eds.), *The Vehicle Routing Problem* (pp. 225–242). SIAM, Society for Industrial and Applied Mathematics.

Di Caro, G. & Dorigo, M. (1998). AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, *9*, 317–365.

Donati, A., Montemanni, R., Casagrande, N., Rizzoli, A., & Gambardella, L. M. (2008). Time dependent vehicle routing problem with a multi ant colony system. *European Journal of Operational Research*, *18*(3), 1174–1191.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy.

Dorigo, M. & Blum, C. (2005). Ant colony optimization theory: a survey. *Theoretical Computer Science*, *344*, 243–278.

Dorigo, M., Di Caro, G., & Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial Life*, *5*(2), 137–172.

Dorigo, M. & Gambardella, L. M. (1996). A study of some properties of Ant-Q. In Voigt, H., Ebeling, W., Rechenberg, I., & Schwefel, H. (Eds.), *Proceedings of PPSN-IV, Fourth International Conference on Parallel Problem Solving from Nature*, volume 1141 of *Lecture Notes in Computer Science*, (pp. 656–665). Springer Verlag, Berlin, Germany.

Dorigo, M. & Gambardella, L. M. (1997). Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, *1*(1), 53–66.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, *26*(1), 29–41.

Dorigo, M. & Stützle, T. (2004). *Ant Colony Optimization*. Cambridge, Massachusetts: MIT Press.

Durbin, R. & Willshaw, D. (1987). An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, *326*, 689–691.

Eilon, S., Watson-Gandy, C., & Christofides, N. (1969). Distribution management: mathematical modeling and practical analysis. *Operational Research Quarterly*, *20*, 37–53.

Escudero, L., Guignard, M., & Malik, K. (1994). A lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships. *Annals of Operations Research*, *50*, 219–237.

Escudero, L. F. (1988). An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research*, *37*, 232–253.

Fisher, M. (1994). Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research*, *42*, 626–642.

Fleurent, C. & Ferland, J. A. (1994). Genetic hybrids for the quadratic assignment problem. In P. M. Pardalos & H. Wolkowicz (Eds.), *Quadratic Assignment and Related Problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science* (pp. 173–187). Providence, Rhode Island.

Flood, M. M. (1956). The traveling-salesman problem. *Operations Research*, *4*, 61–75.

Fogel, D. B. (1993). Applying evolutionary programming to selected traveling salesman problems. *Cybernetics and Systems: An International Journal*, *24*, 27–36.

Fredman, M., Johnson, D., McGeoch, L., & Ostheimer, G. (1995). Data structures for traveling salesmen. *Journal of Algorithms*, *18*, 432–479.

Freisleben, B. & Merz, P. (1996a). Genetic local search algorithm for solving symmetric and asymmetric traveling salesman problemss. In of IEEE International Conference on Evolutionary Computation IEEE-EC 96, P. (Ed.), *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, (pp. 616–621). IEEE Press, Piscataway, NJ.

Freisleben, B. & Merz, P. (1996b). New genetic local search operators for the traveling salesman problem. In of PPSN IV Fourth International Conference on Parallel Problem Solving From Nature 1996, P. (Ed.), *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, (pp. 890–899). Springer-Verlag, Berlin, 1996.

Gambardella, L. M. & Dorigo, M. (1995). Ant-Q: A reinforcement learning approach to the traveling salesman problem. In Prieditis, A. &

Russell, S. (Eds.), *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, (pp. 252–260). Morgan Kaufmann Publishers, Palo Alto, CA.

Gambardella, L. M. & Dorigo, M. (1996). Solving symmetric and asymmetric TSPs by ant colonies. In Baeck, T., Fukuda, T., & Michalewicz, Z. (Eds.), *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, (pp. 622–627). IEEE Press, Piscataway, NJ.

Gambardella, L. M. & Dorigo, M. (2000). An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, *12*(3), 237–255.

Gambardella, L. M., Montemanni, R., & Weyland, D. (2012). Coupling ant colony systems with strong local searches. *European Journal of Operational Research*, *220*(1), 831–843.

Gambardella, L. M., Rizzoli, A., & Zaffalon, M. (1998). Simulation and planning of an intermodal container terminal. *Simulation*, *71*(2), 107–116.

Gambardella, L. M., Taillard, É. D., & Agazzi, G. (1999). MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, & F. Glover (Eds.), *New Ideas in Optimization* (pp. 63–76). McGraw Hill, London, UK.

Gambardella, L. M., Taillard, É. D., & Dorigo, M. (1999). Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, *50*(2), 167–176.

Gendreau, M., Hertz, A., & Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, *40*(10), 1276–1290.

Gendreau, M., Laporte, G., & Séguin, R. (1996). Stochastic vehicle routing. *European Journal of Operational Research*, *88*(1), 3–12.

Gendreau, M. & Potvin, J.-Y. (1998). Dynamic vehicle routing and dispatching. In T. Crainic & G. Laporte (Eds.), *Fleet management and logistic* (pp. 115–226). Berlin, Germany: Springer.

Glover, F. (1989). Tabu search – Part I. *ORSA Journal on Computing*, *1*(3), 190–206.

Glover, F. & Laguna, M. (1997). *Tabu Search*. Boston, Massachusetts: Kluwer Academic Publishers.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA.

Golden, B. L. & Stewart, W. R. (1985). Enpirical analysis of heuristics. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, & D. B. Shmoys (Eds.), *The Traveling Salesman Problem* (pp. 307–360). John Wiley & Sons, Chichester, UK.

Goss, S., Aron, S., Deneubourg, J. L., & Pasteels, J. M. (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, *76*, 579–581.

Grassé, P. P. (1959). La reconstruction du nid et les coordinations interindividuelles chez *Bellicositermes natalensis et Cubitermes sp.* La théorie de la stigmergie : essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, *6*, 41–81.

Guerriero, F. & Mancini, M. (2003). A cooperative parallel rollout algorithm for the sequential ordering problem. *Parallel Computing*, *29*(5), 663–677.

Guntsch, M. & Middendorf, M. (2001). Pheromone modification strategies for ant algorithms applied to dynamic TSP. In E. B. et al. (Ed.), *Applications of Evolutionary Computing : EvoWorkshops 2001: EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM, Como, Italy, April 18-20, 2001, Proceedings*, volume 2037 of *Lecture Notes in Computer Science* (pp. 213–222). Berlin, Germany: Springer.

Hartl, R., Hasle, G., & Janssens, G. (2006). Special issue on rich vehicle routing problems: Editorial. *Central European Journal of Operations Research*, *14*(2), 103–104.

Hernàdvölgyi, I. (2003). Solving the sequential ordering problem with automatically generated lower bounds. In D. Ahr, R. Fahrion, M. O. & Reinelt, G. (Eds.), *Proceedings of Operations Research 2003*, (pp. 355–362). Springer-Verlang.

Hernàdvölgyi, I. (2004). *Automatically Generated Lower Bounds for Search*. PhD thesis, University of Ottawa.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI.

Holland, J. H. & Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In Hayes-Roth, F. & Waterman, D. (Eds.), *Pattern Directed Inference Systems*, (pp. 313–329). Academic Press, New York.

Hölldobler, B. & Wilson, E. O. (1990). *The Ants*. Berlin: Springer Verlag.

Ichoua, S., Gendreau, M., & Potvin, J.-Y. (2003). Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, *144*(2), 379–396.

Jaillet, P. (1985). *Probabilistic traveling salesman problems*. PhD thesis, M. I. T., Dept. of Civil Engineering.

Johnson, D. S. & McGeoch, L. A. (1997). The travelling salesman problem: A case study in local optimization. In E. H. L. Aarts & J. K. Lenstra (Eds.), *Local Search in Combinatorial Optimization* (pp. 215–310). John Wiley & Sons, Chichester, UK.

Johnson, D. S. & McGeoch, L. A. (2002). Experimental analysis of heuristics for the STSP. In G. Gutin & A. Punnen (Eds.), *The Traveling Salesman Problem and its Variations* (pp. 369–443). Kluwer Academic Publishers.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kallehauge, B., Larsen, J., & Madsen, O. B. G. (2006). Lagrangian duality applied to the vehicle routing problem with time windows. *Computers and Operations Research*, *33*(5), 1464–1487.

Kanellakis, P.-C. & Papadimitriou, C. (1980). Local search for the asymmetric traveling salesman problem. *Operations Research*, *28*(5), 1087–1099.

Kilby, P., Prosser, P., & Shaw, P. (1999). *Guided Local Search for the Vehicle Routing Problems With Time Windows, in Meta-heuristics: Advances and Trends in Local Search for Optimization, S.Voss, S. Martello, I.H. Osman and C.Roucairol (eds.)*, (pp. 473–486). Kluwer Academic Publishers.

Kindervater, G. & Savelsbergh, M. (1997). Vehicle routing: Handling edge exchanges. In E. H. L. Aarts & J. K. Lenstra (Eds.), *Local Search in Combinatorial Optimization* (pp. 311–336). John Wiley & Sons, Chichester, UK.

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Kohl, N., Desrosiers, J., Madsen, O. B. G., Solomon, M. M., & Soumis, F. (1997). K-path cuts for the vehicle routing problem with time windows. Technical Report IMM-REP-1997-12, Technical University of Denmark.

Kötzing, T., Neumann, F., Röglin, H., & Witt, C. (2010). Theoretical properties of two ACO approaches for the traveling salesman problem. In *Proceedings of the 7th ANTS conference*, (pp. 324–335).

Kytöjoki, J., Nuortio, T., Bräysy, O., & Gendreau, M. (2007). An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers and Operations Research*, *34*(9), 2743–2757.

Labbé, M., Laporte, G., & Mercure, H. (1991). Capacitated vehicle routing on trees. *Operations Research*, *39*(4), 616–622.

Laporte, G. & Louveaux, F. (1998). Solving stochastic routing problems with the integer L-shaped method. In T. Crainic & G. Laporte (Eds.), *Fleet Management and Logistics* (pp. 159–167). Boston, Massachusetts: Kluwer Academic Publishers.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). *The Travelling Salesman Problem*. John Wiley & Sons, Chichester, UK.

Li, F., Golden, B., & Wasil, E. (2005). Very large-scale vehicle routing: New test problems, algorithms, and results. *Computers and Operations Research*, *32*(5), 1165–1179.

Li, Y. & Chan Hilton, A. (2007). Optimal groundwater monitoring design using an ant colony optimization paradigm. *Environmental Modelling and Software*, *22*(1), 110–116.

Lin, F., Kao, C., & Hsu, C. (1993). Applying the genetic approach to simulated annealing in solving some np-hard problems. *IEEE Transactions on Systems, Man, and Cybernetics*, (23), 1752–1767.

Lin, S. (1965). Computer solutions for the traveling salesman problem. *Bell Systems Technology Journal*, *44*, 2245–2269.

Lin, S. & Kernighan, B. W. (1973). An effective heuristic algorithm for the travelling salesman problem. *Operations Research*, *21*, 498–516.

Lourenço, H., Martin, O., & Stützle, T. (2003). Iterated local search. In F. Glover & G. Kochenberger (Eds.), *Handbook of Metaheuristics* (pp. 321–353). Boston, Massachusetts: Kluwer Academic Publishers.

Malek, M., Guruswamy, M., & Pandya, M. (1989). Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research*, *21*, 59–84.

Maniezzo, V. & Carbonaro, A. (2000). ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, *16*(8), 927–935.

Maniezzo, V., Colorni, A., & Dorigo, M. (1994). The Ant System applied to the quadratic assignment problem. Technical Report IRIDIA/94-28, IRIDIA, Université Libre de Bruxelles, Belgium.

Maniezzo, V., Gambardella, L. M., & De Luigi, F. (2004). Ant colony optimization. In G. C. Onwubolu & B. V. Babu (Eds.), *New Optimization Techniques in Engineering* (pp. 101–117). Springer-Verlag Berlin Heidelberg.

Martin, O. & Otto, S. W. (1996). Combining simulated annealing with local search heuristics. *Annals of Operations Research*, *63*, 57–75. 10.1007/BF02601639.

Martin, O., Otto, S. W., & Felten, E. W. (1991). Large-step Markov chains for the traveling salesman problem. *Complex Systems*, *5*(3), 299–326.

Martin, O., Otto, S. W., & Felten, E. W. (1992). Large-step markov chains for the tsp incorporating local search heuristics. *Operations Research Letters*, *11*, 219–224.

Merz, P. & Freisleben, B. (1997). Genetic local search for the TSP: New results. In Bäck, T., Michalewicz, Z., & Yao, X. (Eds.), *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, (pp. 159–164). IEEE Press, Piscataway, NJ.

Mester, D. & Bräysy, O. (2005). Active guided evolution strategies for the large scale vehicle routing problem with time windows. *Computers & Operations Research*, *32*(6), 1593–1614.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., & Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, *21*, 1087–1092.

Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, Berlin, Germany.

Monmarché, N., Guinand, F., & Siarry, P. (Eds.). (2010). *Artificial Ants*. John Wiley & Sons.

Montemanni, R., Gambardella, L. M., Rizzoli, A., & Donati, A. (2005). Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, *10*, 327–343.

Montemanni, R., Smith, D., & Gambardella, L. M. (2007). Ant Colony Systems for large Sequential Ordering Problems. In *Proceedings IEEE Swarm Intelligence Symposium, SIS 2007, Honolulu, Hawaii, USA, April 1-5, 2007*, (pp. 60–67). IEEE.

Montemanni, R., Smith, D., & Gambardella, L. M. (2008). A heuristic manipulation technique for the sequential ordering problem. *Computers and Operations Research*, *35*(12), 3931–3944.

Montemanni, R., Smith, D., Rizzoli, A., & Gambardella, L. M. (2009). Sequential ordering problems for crane scheduling in port terminals. *International Journal of Simulation and Process Modelling*, *5*(4), 348–361.

Mühlenbein, H., Gorges-Schleuter, M., & Krämer, O. (1988). Evolution algorithms in combinatorial optimization. *Parallel Computing*, *7*, 65–85.

O. Schoonderwoerd, J. Holland, J. B. & Rothkrantz, L. (1996). Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, *5*(2), 169–207.

Or, I. (1976). *Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking*. PhD thesis, Department of Industrial and Engineering and Management Science, Northwestern University, Evanston, IL.

Osman, I. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, *41*, 421–451.

Papadimitriou, C. H. & Steiglitz, K. (1982). *Combinatorial Optimization – Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ.

Potvin, J.-Y. (1993). The traveling salesman problem: a neural network perspective. *ORSA Journal of Computing*, *5*, 328–347.

Potvin, J.-Y. & Bengio, S. (1996). The vehicle routing problem with time windows - part ii: Genetic search. *INFORMS Journal of Computing*, *8*, 165–172.

Potvin, J.-Y., Xu, Y., & Benyahia, I. (2006). Vehicle routing and scheduling with dynamic travel times. *Computers and Operations Research*, *33*(4), 1129–1137.

Prim, R. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, *36*, 1389–1401.

Psaraftis, H. (1983). K-interchange procedures for local search in a precedence-constrained routing problem. *European Journal of Operational Research*, *13*, 341–402.

Psaraftis, H. (1988). Dynamic vehicle routing problems. In B. Golden & A. Assad (Eds.), *Vehicle Routing: Methods and Studies* (pp. 223–248). Amsterdam, The Netherlands: North-Holland.

Psaraftis, H. (1995). Dynamic vehicle routing: status and prospects. *Annals of Operations Research*, *61*, 143–164.

Pulleyblank, M. & Timlin, M. (1991). Precedence constrained routing and helicopter scheduling: heuristic design.

Rego, C. & Roucairol, C. (1996). *A Parallel Tabu Search Algorithm Using Ejection Chains for the Vehicle Routing Problem, in Meta-heuristics: Theory and applications (I.H. Osman, J. Kelly eds.)*, (pp. 661–675). Kluwer Academic Publishers.

Reimann, M., Doerner, K., & Hartl, R. (2002). A savings based ant system for the vehicle routing problem. In et al., W. L. (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, (pp. 1317–1325)., San Francisco, California. Morgan Kaufmann Publishers.

Reimann, M., Doerner, K., & Hartl, R. (2003). Analyzing a unified ant system for the VRP and some of its variants. In G. R. et al. (Ed.), *Applications of Evolutionary Computing: EvoWorkshops 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM, Essex, UK, April 14-16, 2003. Proceedings*, volume 2611 of *Lecture Notes in Computer Science* (pp. 300–310). Berlin, Germany: Springer.

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

Reinelt, G. (1991). TSPLIB — A traveling salesman problem library. *ORSA Journal on Computing*, *3*, 376–384.

Reinelt, G. (1994). *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany.

Resende, M. & Ribeiro, C. (2003). Greedy randomized adaptive search procedures. In F. Glover & G. Kochenberger (Eds.), *Handbook of Metaheuristics* (pp. 219–249). Boston, Massachusetts: Kluwer Academic Publishers.

Resende, M. & Ribeiro, C. (2005). Grasp with path-relinking: Recent advances and applications. In *Metaheuristics: Progress as Real Problem Solvers*, (pp. 29–63). Springer.

Resende, M., Ribeiro, C., Glover, F., & Martì, R. (2010). Scatter search and path-relinking: fundamentals, advances, and applications. In *Handbook of Metaheuristics*, (pp. 87–107). Springer.

Rizzoli, A., Montemanni, R., Lucibello, E., & Gambardella, L. M. (2007). Ant colony optimisation for real world vehicle routing problems: from theory to applications. *Swarm Intelligence*, *1*(2), 135–151.

Rochat, Y. & Taillard, É. D. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, *1*, 147–167.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error backpropagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition – Volume 1: Foundations* (pp. 318–362). MIT Press, Cambridge, MA.

Savelsbergh, M. (1985). Local search in routing problems with time windows. *Annals of Operations Research*, *4*, 285–305.

Savelsbergh, M. W. P. (1990). An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, *47*, 75–85.

Seo, D.-I. & Moon, B. (2003). A hybrid genetic algorithm based on complete graph representation for the sequential ordering problem. In *Proceedings of GECCO 2003*, (pp. 69–680). Springer-Verlang.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *roceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP '98), M. Maher and J.-F. Puget (eds.)*, (pp. 417–431). Springer-Verlag.

Solomon, M. (1987). Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, *35*, 254–365.

Stützle, T. (1999). *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications*, volume 220 of *DISKI*. Infix, Sankt Augustin, Germany.

Stützle, T. & Dorigo, M. (1999a). ACO algorithms for the quadratic assignment problem. In D. Corne, M. Dorigo, & F. Glover (Eds.), *New Ideas in Optimization* (pp. 33–50). McGraw Hill, London, UK.

Stützle, T. & Dorigo, M. (1999b). ACO algorithms for the traveling salesman problem. In K. Miettinen, M. M. Mäkelä, P. Neittaanmäki, & J. Périaux (Eds.), *Evolutionary Algorithms in Engineering and Computer Science* (pp. 163–183). John Wiley & Sons, Chichester, UK.

Stützle, T. & Hoos, H. (2000). MAX-MIN ant-system. *Future Generation Computer Systems*, *16*(8), 889–914.

Stützle, T. & Hoos, H. H. (1997). The $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System and local search for the traveling salesman problem. In Bäck, T., Michalewicz, Z., & Yao, X. (Eds.), *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, (pp. 309–314). IEEE Press, Piscataway, NJ.

Stützle, T. & Hoos, H. H. (1998). Improvements on the Ant System: Introducing the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. In G. D. Smith, N. C. Steele, R. F. A. (Ed.), *Artificial Neural Networks and Genetic Algorithms*, (pp. 245–249). Springer Verlag, Vienna, Austria.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Taillard, É. D. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing*, *17*, 443–455.

Taillard, É. D. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, *23*, 66117673.

Taillard, É. D., Badeau, P., Gendreau, M., Guertin, F., & Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, *31*, 170–186.

Taillard, É. D., Gambardella, L. M., Gendreau, M., & Potvin, J.-Y. (1998). Adaptive memory programming: A unified view of metaheuristics. Technical Report IDSIA-19-98, IDSIA.

Thangiah, S. R., Osman, I. H., & Sun, T. (1994). Hybrid genetic algorithm simulated annealing and tabu search methods for vehicle routing

problem with time windows. Technical Report 27, Computer Science Department, Slippery Rock University.

Toth, P. & Vigo, D. (2001a). Branch-and-bound algorithms for the capacitated VRP. In P. Toth & D. Vigo (Eds.), *The Vehicle Routing Problem* (pp. 29–51). SIAM, Society for Industrial and Applied Mathematics.

Toth, P. & Vigo, D. (2001b). An overview of vehicle routing problems. In P. Toth & D. Vigo (Eds.), *The Vehicle Routing Problem* (pp. 1–26). SIAM, Society for Industrial and Applied Mathematics.

Toth, P. & Vigo, D. (2003). The granular tabu search and its application to the vehicle routing problem. *INFORMS Journal on Computing*, *15*(4), 333–346.

Tsubakitani, S. & Evans, J. R. (1998). Optimizing tabu list size for the traveling salesman problem. *Computers and Operations Research*, *25*(2), 91–97.

Ulder, N. L. J., Aarts, E. H. L., Bandelt, H.-J., van Laarhoven, P. J. M., & Pesch, E. (1991). Genetic local search algorithms for the travelling salesman problem. In Schwefel, H.-P. & Männer, R. (Eds.), *Proceedings of PPSN-I, First International Conference on Parallel Problem Solving from Nature*, number 496 in Lecture Notes in Computer Science, (pp. 109–116). Springer Verlag, Berlin, Germany.

Van Breedam, A. (1996). An analysis of the effect of local improvement operators in genetic algorithms and simulated annealing for the vehicle routing problem. RUCA Working Paper 96/14, University of Antwerp, Belgium.

Van der Bruggen, L., Lenstra, L., & Schuur, P. (1993). Variable depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Science*, *27*, 298–311.

Watkins, C. J. (1989). *Learning with Delayed Rewards*. PhD thesis, Psychology Department, University of Cambridge, UK.

Whitley, D., Starkweather, T., & Fuquay, D. (1989). Scheduling problems and travelling salesman: The genetic edge recombination operator. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference*

*on Genetic Algorithms (ICGA'89)*, (pp. 133–140). Morgan Kaufmann Publishers, Palo Alto, CA.

Xu, J. & Kelly, J. (1996). A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science*, *30*, 379–393.

Zecchin, A., Maier, H. R., Simpson, A. R., Leonard, M., & Nixon, J. B. (2007). Ant colony optimization applied to water distribution system design: Comparative study of five algorithms. *Journal of Water Resources Planning and Management*, *133*(1), 87–92.

Zeimpekis, V., Tarantilis, C., Giaglis, G., & Minis, I. (Eds.). (2007). *Dynamic Fleet Management – Concepts, Systems, Algorithms & Case Studies*. Operations Research/Computer Science Interfaces. Berlin, Germany: Springer.