Université Libre de Bruxelles
Faculté des Sciences Appliquées
IRIDIA - *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*

Ant Colony Optimization for the Vehicle Routing Problem

Max Manfrin

Directeur de Mémoire:

Prof. Marco Dorigo

Mémoire présentée en vue de l'obtention du Diplôme d'Etudes Approfondies en Sciences Appliqueées

Année Académique 2003/2004

Abstract

The work presented in this thesis is part of the research carried out in the *Metaheuristics Network*, a Research Training Network sponsored by the Improving Human Potential Program of the European Community (HPRN-CT-1999-00106).

This thesis consist of two parts. The first part (Chapters 1 and 2) presents an introduction to stochastic routing problems followed by the description and the mathematical model of the Vehicle Routing Problem with Stochastic Demands (VRPSD). As stochastic routing problems are $\mathcal{NP}$-hard combinatorial optimization problems, quite a lot of research has been devoted to the development of metaheuristic methods to tackle them, so a literature review is also given. Metaheuristic methods are approximate methods which combine basic heuristic methods in a higher level framework aimed at efficiently exploring a search space. Chapter 2 introduces the concept of metaheuristic and shows some criteria for classification of metaheuristics. Successively it gives a description of nowadays most important metaheuristics.

The second part of the thesis (Chapters 3 and 4) summarizes the research results of the *Metaheuristics Network* on the development and comparison of metaheuristics to tackle the VRPSD. The *Metaheuristics Network* aims at the comparison of metaheuristics on different combinatorial optimization problems. For each combinatorial optimization problem considered, five metaheuristics are implemented by different persons in different sites involved in the *Metaheuristics Network*. The five metaheuristics considered are: Ant Colony Optimization, Evolutionary Computation, Iterated Local Search, Tabu Search, and Simulated Annealing. The comparison of the experimental results of the implemented metaheuristics [11] will be presented at the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII) that will be held in Birmingham, UK, on 18-22 September 2004 and will be published in a forthcoming volume of the series *Lecture Notes in Computer Science*.

Statement

This work was supported by the *Metaheuristics Network*, a Research Training Network funded by the Improving Human Potential Program of the CEC, grant HPRN-CT-1999-00106 and by *COMP*^{2}*SYS*, a Marie Curie Early Stage Training Site, funded by the the European Commission through the Human Resources and Mobility Program, grant MEST-CT-2004-505079.

The information provided is the sole responsibility of the author and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

This work is an original research and has been done in collaboration with the researchers of the *Metaheuristics Network*. Part of this work as already been published in Bianchi et al. [14, 11] and Manfrin et al. [64, 65].

Acknowledgments

I wish to express my gratitude to my supervisor, Prof. Marco Dorigo, for the opportunity to work in an environment where I had all the necessary and even more. His confidence in me, his patience and his precision have been essential factors to the success of this research. In this context my gratitude goes also to Prof. Hugues Bersini , the director of IRIDIA.

Special thanks to Mauro Birattari, for his constant encouragement, his great help in clarifying and synthesizing the results of this work, and the fruitful collaboration we had throughout the *Metaheuristics Network* project.

I wish to thank all the participants to the *Metaheuristics Network*, with whom I shared the entusiasm and the weight of the work, in particular I wish to thank Leonora Bianchi, Marco Chiarandini, Olivia Rossi-Doira, Tommaso Schiavinotto, Luis Paquete and Monaldo Mastrolilli for the fruitful discussions and the numerous exchange of ideas.

I'm in debt with people in IRIDIA that have always demonstrated sympathy and kindness to me: Carlotta Piscopo, Halva Labella, Shervin Nouyan, Vito Trianni, Elio Tuci, Ampatzis Christos, Roderich Groß and Bruno Marchal.

Special thanks to my family, particularly to my sister Ester, for always having encouraged me and supported me in following my path.

Last but not least I wish to thank Roberta Billai...words make no justice to the love I have for her.

This work was supported by the *Metaheuristics Network*, a Research Training funded by the the European Commission.

Contents

Abstract **i**

Statement **iii**

Acknowledgments **v**

Contents **vii**

List of figures **ix**

List of algorithms **xi**

Introduction **1**
Original contributions 3
Structure of the thesis 3

1 Definition of the problem **5**
1.1 Introduction to stochastic routing problems 5
1.2 A mathematical model of the VRPSD 7
1.2.1 Threshold and expected cost evaluation 8
1.3 Examples of real-world applications 9
1.4 Literature review 10

2 Metaheuristics **13**
2.1 Classification of metaheuristics 14
2.2 Trajectory methods 16
2.2.1 Iterated Local Search 17
2.2.2 Simulated Annealing 19
2.2.3 Tabu Search 21
2.3 Population-based methods 22
2.3.1 Ant Colony Optimization 23
2.3.2 Evolutionary Computation 26

3 Preliminary experiments **29**
3.1 The guidelines of the *Metaheuristics Network* 29
3.1.1 The Starter Kit 31
3.2 The Algorithms 33
3.2.1 Iterated Local Search 33
3.2.2 Flim-Flam 34

3.2.3 Simulated Annealing . . . 34
3.2.4 Tabu search . . . 36
3.2.5 Ant Colony Optimization . . . 37
3.2.6 Evolutionary Computation . . . 38
3.3 Results of the preliminary experiments . . . 39
3.3.1 How to read the tables and the graphics . . . 40
3.4 Relation between TSP and VRPSD . . . 51
3.4.1 Pathological cases . . . 51

4 Further experiments **53**
4.1 Modification to the Starter Kit . . . 53
4.1.1 The Or-Opt local search . . . 54
4.1.2 The randomized farthest insertion heuristic . . . 55
4.2 The Algorithms . . . 55
4.2.1 Iterated Local Search . . . 56
4.2.2 Simulated Annealing . . . 56
4.2.3 Tabu Search . . . 57
4.2.4 Ant Colony Optimization . . . 60
4.2.5 Evolutionary Computation . . . 60
4.3 Results of the successive experiments . . . 60
4.4 Future works . . . 61
4.4.1 Homogeneous Failure and Restocking . . . 61
4.4.2 Possible Extensions of the Problem . . . 74

Conclusions **77**

Bibliography **79**

List of Figures

1.1 Expected cost-to-go of the vehicle under restocking strategy . . . 9

3.1 Example of Or-Opt forward insertion 35
3.2 Preliminary experiments: results aggregated over the 4 classes of instances. 41
3.3 Preliminary experiments: results on the **rand_unif_n** class of instances. 43
3.4 Preliminary experiments: results on the **rand_unif_r** class of instances. 45
3.5 Preliminary experiments: results on the **rand_clust_n** class of instances. 47
3.6 Preliminary experiments: results on the **rand_clust_r** class of instances. 49
3.7 Pathological instance for VRPSD 52

4.1 Arcs involved in the computation of the cost move in the Or-Opt-tsp local search. 54
4.2 Example of tabu moves. 59
4.3 Results aggregated over the 4 classes of instances. 62
4.4 Results on the **rand_unif_n** class of instances. 64
4.5 Results on the **rand_unif_r** class of instances. 66
4.6 Results on the **rand_clust_n** class of instances. 68
4.7 Results on the **rand_clust_r** class of instances. 70
4.8 Cost of including an arc in the *a priori* tour 73

List of Algorithms

1 Computation of the VRPSD objective function 9
2 Local Search 17
3 Iterated Local Search 18
4 Simulated Annealing 20
5 Simple Tabu Search 21
6 Tabu Search 22
7 Ant Colony Optimization 24
8 Evolutionary Computation 27
9 Or-Opt Local Search 32
10 Tabu Search - preliminary experiments 37
11 Evolutionary Computation - preliminary experiments 39
12 Or-Opt-tsp Local Search 55
13 Tabu Search - TS-0 57
14 Tabu Search - TS-tsp 58

Introduction

Combinatorial optimization (CO) problems are growing in importance for the scientific and the industrial world. In many real life settings, high quality solutions to CO problems such as VEHICLE ROUTING are required in a very short amount of time. In those cases, especially when large scale problems are considered, metaheuristics are one of the best alternatives. Ant Colony Optimization is one of the newest metaheuristic approaches. It was developed in the early '90 by Marco Dorigo and colleagues. This thesis provides contributions to the practical applicability of Ant Colony Optimization.

The work presented in this thesis is part of the research carried out in the *Metaheuristics Network*,[1] a Research Training Network sponsored by the *Improving Human Potential* Program of the European Community.[2] Initially composed of six participants, it now comprises five institutions:

- **IRIDIA - Univeristé Libre de Bruxelles - Brussels - Belgium**
- **INTELLEKTIK - Technische Universität Darmstadt - Darmstadt - Germany**
- **ECRG - Napier University - Edinburgh - UK**
- **IDSIA - Manno - Switzerland**
- **ANTOPTIMA - Lugano - Switzerland** (joined the network in 2001).

Two institutions have left the Network after one and two years, respectively:

- **COG - Technische Universiteit Eindhoven - Eindhoven - The Netherlands** (left the Network on the 31.08.2001);
- **EUROBIOS - Paris - France** (left the Network on the 31.08.2002).

The *Metaheuristics Network* has goals of scientific, engineering and training nature. These are explained in the following.

- Scientific goals: the main scientific goal of the *Metaheuristics Network* is to improve the understanding of how metaheuristics work through theoretical and experimental research. Other scientific goals are the definition of new high-performing hybrid metaheuristics, that is, metaheuristics that combine components taken from existing metaheuristics, as well as the

[1] `http://www.metaheuristics.org`
[2] Contract number HPRN-CT-1999-00106.

definition and use of a strictly controlled, machine independent, experimental methodology which allows a fair and meaningful comparison of experimental results.

- Engineering goals: the first engineering oriented goal consists in the definition of guidelines that can be used to help choosing which metaheuristics, or metaheuristic components, to use when a new problem is attacked. A second engineering oriented goal of the network is the testing and validation of the developed ideas on very challenging problems taken from the industrial world.

- Training goals: The main training goals of the network are: (i) to let young researchers learn about metaheuristic techniques as well as about a number of important optimization problems, (ii) to let them learn how to design experiments in a rigorous way, and (iii) to develop their project management skills by means of participation in a strictly co-ordinated international team activity.

The *Metaheuristics Network* has spent fourteen months (from January 2003 to February 2004) to study the Vehicle Routing Problem with Stochastic Demands (VRPSD), a real-world problem whose study is the topic of this DEA thesis. The interest in this problem is motivated by both its practical relevance and its considerable difficulty.

The general Vehicle Routing Problem calls for the determination of the optimal set of routes to be performed by a fleet of vehicles to serve a given set of customers. The VRPSD arises in practice whenever one faces the problem of deliveries to (or collection from) a set of customers, whose demands are uncertain.
Unlike its deterministic equivalent, the VRPSD is ambiguously defined since it belongs to a class of *a priori* optimization problems (see Bertsimas et al. [9]) for which it is impractical to consider an *a posteriori* approach (such that an optimal solution is computed each time the value of a stochastic demand is known). Instead, an *a priori* solution attempts to obtain the best solution, over all possible problem scenarios, before the realization of any single scenario. Roberts and Hadjiconstantinou [80] evaluated the computational performance of such a solution method and showed that an *a priori* solution for a Vehicle Routing Problem where demand is uncertain lies, on average, within 8% of the solution obtained by a reoptimization-based, *a posteriori* strategy.

The *Metaheuristics Network* has considered a particular version of the VRPSD, and has proposed the comparison of the experimental results obtained by five metaheuristics under the same experimental conditions, on instances of the problem. The metaheuristics studied include: Ant Colony Optimization (ACO), Evolutionary Computation (EC), Iterated Local Search (ILS), Simulated Annealing (SA), and Tabu Search (TS). To guarantee meaningful comparisons and the same experimental conditions, all the implemented algorithm use the same direct representation of the solution and of the search space (thanks to the definition of a common neighborhood structure and a common local search routine). Moreover, to avoid a bias due to the choice of the programming language, all the metaheuristics have been implemented in a common programming language (C++) using a library of common objects with the same basic data

structures. All the source codes have been compiled using the same compilation environment, and the experiments have been run on a same machine.

The time frame of fourteen months was divided into an initial phase and a second phase. In the initial phase, which lasted three months, a "basic" version of each metaheuristic was implemented, and a first comparison of the basic metaheuristics was performed. The work done during the initial phase showed that for this particular problem the important element was to find a way to reduce the computational complexity of the procedure used to compute the objective function. For this reason, in the remaining eleven months, the results of the initial phase were exploited, and we proceeded with the research of approximation schemas for the objective function

Original contributions

The original contributions are:

- Our Ant Colony System algorithm is the first implementation of an ACO approach to the Vehicle Routing Problem with Stochastic Demands appeared in the literature.
- The comparison of the experimental results of the five implemented metaheuristics [11] will be presented at the international conference PPSN VIII 2004 and will be published in a forthcoming volume of the series *Lecture Notes in Computer Science*.

Structure of the thesis

The thesis has the following structure:

- Chapter 1 presents an introduction to stochastic routing problems followed by the description and the mathematical model of the Vehicle Routing Problem with Stochastic Demands. As stochastic routing problems are $\mathcal{NP}$-hard combinatorial optimization problems, quite a lot of research has been devoted to the development of metaheuristic methods to tackle them, so a literature review is also given.
- In Chapter 2 we introduce the concept of metaheuristic, we show some criteria for classification of metaheuristics and we give a description of nowadays most important metaheuristics.
- In the first part of Chapter 3 we describe the guidelines given by the *Metaheuristics Network* to implement the metaheuristics, and we show the implemented algorithms. In the last part of the chapter we show the comparison of the experimental results obtained during the preliminary experiments, and we analyze how the Traveling Salesman Problem and the Vehicle Routing Problem with Stochastic Demands are related, and how we can exploit this relation to implement better algorithms.
- In Chapter 4 we focus on how to exploit the similarity between the Traveling Salesman Problem and the Vehicle Routing Problem with

STOCHASTIC DEMANDS. We describe the implemented metaheuristics and the comparison of the experimental results done in the further experiments.

- In the Conclusions we analyze the results that we obtained with the use of the different studied approaches and we discuss possible future developments.

Chapter 1

Definition of the problem

The field of metaheuristics for the application to CO problems is a growing field of research. This is due to the importance of CO problems for the scientific as well as the industrial world. In many real life settings, high quality solutions to CO problems such as vehicle routing or timetabling are required in a very short amount of time. In those cases, especially when large scale problems are considered, metaheuristics are one of the best alternatives.[1]

According to Papadimitriou and Steiglitz [76], a CO problem $\mathcal{P} = (\mathcal{S}, f)$ is an optimization problem in which a finite set of objects $\mathcal{S}$ is given along with an objective function $f : \mathcal{S} \rightarrow \Re^+$ that assigns a positive cost to each of the objects $s \in \mathcal{S}$. The goal is to find an object of minimal cost.[2] The objects are typically integer numbers, subsets of a set of items, permutations of a set of items, or graph structures. Due to the practical importance of CO problems, many algorithms to tackle them have been developed. These algorithms can be classified as either *complete* or *approximate* algorithms. Complete algorithms are guaranteed to find for every finite size instance of a CO problem an optimal solution in bounded time (see Papadimitriou and Steiglitz [76], Nemhauser and Wolsey [73]). Yet, for CO problems that are $\mathcal{NP}$-hard [41], no polynomial time algorithm exists, assuming that $\mathcal{P} \neq \mathcal{NP}$. Therefore, complete methods might need exponential computation time in the worst-case. This often leads to computation times too high for practical purposes. In approximate methods we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time.

1.1 Introduction to stochastic routing problems

The deterministic Vehicle Routing Problem (VRP) is well known in the operations research literature (see Fisher [36], Bertsimas and Simchi-Levi [10], Bramel and Simchi-Levi [17], Crainic and Laporte [23], Golden and Assad [48] for reviews). In the classical definition of VRP it is assumed that the associated parameters, concerning factors such as cost, customer demands, and ve-

[1] The increasing importance of metaheuristics is underlined by the biannual Metaheuristics International Conference (MIC). The 5th was held in Kyoto in August 2003 (*http://www-or.amp.i.kyoto-u.ac.jp/mic2003/).*

[2] Note that minimizing over an objective function f is the same as maximizing over $-f$. Therefore, every CO problem can be described as a minimization problem.

hicle travel times, are deterministic. The work done for this thesis within the *Metaheuristics Network* deals with a variation where each customer demand is assumed to follow a given probability distribution, instead of having a single known value. The actual customer demand is known only upon arrival at the customer's location. Such a problem is known in the literature as Vehicle Routing Problem with Stochastic Demands. The VRPSD arises in practice whenever one faces the problem of deliveries to (or collections from) a set of customers, whose demands are uncertain.

In a deterministic VRP, routes are planned in such a way that vehicles always have enough capacity to satisfy all customers' demands on their pre-planned routes. When demands are stochastic, this is possible only by loosing the concept of 'pre-planned routes', and by adding a set of decisions rules (a *routing strategy* or *routing policy*). In fact, suppose a vehicle visits customers in a certain pre-planned order. It is possible that at some customer location the vehicle capacity is reached or exceeded, and a route failure is said to occur. In such a situation, the following decision (or 'recourse action') may be taken: the vehicle returns to the depot, replenishes, and either goes back to the customer where demand has not been fully satisfied or resumes its tour with the next customer. Therefore, the vehicle will always be able to serve all demands of customers on its pre-planned tour, but the distance traveled is a random quantity, and the locations where recurse actions will be taken are not known in advance. Indeed, the actual tour followed by the vehicle might be in general different by the pre-planned one. The goal, in the VRPSD, is to determine a routing policy such that demand at each customer is met, and the expected distance traveled is minimized.

We can think about the following types of routing policies:

- off-line (or *a priori*, static, open-loop)
- on-line (or real-time, dynamic, closed-loop)
- mixed

Off-line policies prescribe for a vehicle a sequence τ of customers to be visited in the sequence order. Such policy is then executed in real-time, supplemented by simple recourse actions to accommodate failures that may occur when one actually observes the realizations of the random variables involved in the problem. On the other extreme are on-line policies. An on-line policy tells, for each possible state of the system, which location to visit next. In practice, the application of an on-line policy consists in optimally re-sequencing the unvisited customers whenever a vehicle arrives at a new customer location. Mixed policies combine elements of both; for instance, the vehicle follows the prescribed tour, but it is enabled with state dependent rules that allow for preventive returns to the depot, before a route failure actually occurs.

In this DEA thesis we focus on the mixed policy of *preventive restocking*, which works in the following way. After a customer has been served, we take a decision on whether to return to the depot for replenishment, or to go on with the next customer. The decision is taken in order to trade off the extra cost of returning to the depot after a stock-out with the cost of returning to the depot for reloading before an out-of-stock actually occurs at a customer. Thus, the point at which the vehicle returns to the depot may be before a stock-out

actually takes place. *Preventive restocking* is done if the residual capacity of the vehicle after serving a customer is below a certain threshold, which is calculated in advance, at the moment of the routes planning.

Motivation of our approach (mixed policy) For a given VRPSD instance, and for a given set of recourse actions, the set of off-line policies is a subset of the set of mixed policies, that, in turn, is included in that of on-line ones. Since all of them aim at the minimization of the same quantity (i.e., the expected distance traveled), in principle better solution quality is achieved by on-line policies. Secomandi [81] provided an estimation of the gap between the performance of optimal on-line policies and optimal off-line/mixed policies. This gap is below 3% for optimal off-line policies, and below 1% for optimal mixed policies, in small instances with up to 8 customers. Moreover, there is theoretical evidence [10, 81] that the size of this gap tends to zero, as the number of customers tends to infinite, and they are uniformly and independently distributed on the unit square. Therefore, off-line, mixed and on-line policies are asymptotically equivalent. Even if slightly better performing, the on-line approach has several difficulties [7]: the dispatch company might not have the resources for dynamically re-sequencing unvisited customers; the redesign of tours might be not sufficiently good to justify the required effort and cost; the operator might have other priorities, such as regularity and personalization of service. Instead, off-line and mixed policies are preferable from a computational point of view since they entail solving only one instance of an $\mathcal{NP}$-hard problem. They also produce a more stable and practically predictable solution, since the customer sequence of each route is predetermined. The implication of these observations is that algorithmic effort should be directed on computing close to optimal mixed policies, as also noted by Secomandi [81].

1.2 A mathematical model of the VRPSD

The VRPSD is defined on a complete graph $G = (V, A, C)$, where:

V = $\{0, 1, ..., n\}$, a set of nodes with node 0 denoting the depot.
A = $\{(i, j) : i, j \in V, i \neq j\}$, the set of arcs joining the nodes.
C = (c_{ij}), denoting the travel cost (distance) between nodes i and j.

The cost matrix C is symmetric and satisfies the triangular inequality. Also, all customers have stochastic demands ξ_i, $i = 1, ..., n$, which are independently distributed with known distributions and are known only upon arriving at the customer location. For implementation purposes, it is assumed that ξ_i does not exceed the vehicles capacity Q, and follows a discrete probability distribution $p_{ik} = \text{Prob}(\xi_i = k)$, $k = 0, 1, 2, ..., K \leq Q$. We assume a single vehicle. This is equivalent to assuming that a set of customers has been assigned to receive service by a given vehicle. The vehicle is initially located at the depot.

We adopt the mixed routing policy of *preventive restocking*, described in Section 1.1. The goal is to find a vehicle route and a restocking policy at each node (a threshold), to minimize the total expected cost. The costs under consideration are:

- Cost of traveling from one customer to another as planned.

- Restocking cost: the cost of traveling back to the depot for restocking, before going to the next planned customer.
- Route failure cost: the cost of returning to the depot for restocking caused by insufficient remaining stock in the vehicle to satisfy demand upon arrival at a customer location. This cost is a fixed nonnegative cost b plus a cost of traveling to the depot and back to the route.

Let $0 \to 1 \to 2 \to \cdots \to n$ be a particular planned vehicle route. After the service completion at customer j, suppose the vehicle has a remaining load q, and let $f_j(q)$ denote the total expected cost from node j onward. The expected cost of the planned route (that is, the objective function value) is then $f_0(Q)$, and the problem consists in finding a route with the least expected total cost. If S_j represents the set of all possible loads that a vehicle can have after service completion at customer j, then, $f_j(q)$ for $q \in S_j$ satisfies

$$f_j(q) = \min\{f_j^p(q), f_j^r(q)\}, \tag{1.1}$$

where

$$f_j^p(q) = c_{j,j+1} + \sum_{k:k\leq q} f_{j+1}(q-k)p_{j+1,k} + \sum_{k:k>q} [b+2c_{j+1,0}+f_{j+1}(q+Q-k)]p_{j+1,k}, \tag{1.2}$$

and

$$f_j^r(q) = c_{j,0} + c_{0,j+1} + \sum_{k=1}^{K} f_{j+1}(Q-k)p_{j+1,k}, \tag{1.3}$$

with the boundary condition

$$f_n(q) = c_{n,0},\ q \in S_n. \tag{1.4}$$

In equations (1.1-1.3), $f_j^p(q)$ represents the expected cost of proceeding directly to the next node, and $f_j^r(q)$ represents the expected cost of the restocking action. These equations are used to recursively determine the objective value of the planned vehicle route and the optimal sequence of decisions after customers are served. The dynamic programming [5] recursion for the calculation of $f_0(Q)$ is implemented in Algorithm 1 .

1.2.1 Threshold and expected cost evaluation

The expected cost-to-go in case of restocking, $f_j^r(q)$, is constant in q, since in case of restocking the vehicle will have full capacity Q before serving the next customer, whatever the current capacity q is. On the other hand, $f_j^p(q)$ is a monotonically non-increasing function of q (proof in Yang et al. [93]), for every fixed customer j. Therefore there is a capacity threshold value h_j such that, if the vehicle has more than this value of residual goods, then the best policy is to proceed to the next planned customer, otherwise it is better to go back to the depot for restocking (see Figure 1.1).

Algorithm 1 is an implementation of the dynamic programming recursion (1.1-1.4) for the calculation of $f_0(Q)$ and of the thresholds. Let us assume the customers demands may take values $\xi \in \{0, 1, ..., K\}$. Then, the algorithm runs in $O(nKQ)$ time; the memory required is $O(nQ)$, if one is interested in memorizing all intermediate values $f_j(q)$, for $j = 1, 2, ..., n$ and $q = 0, 1, ..., Q$, and $O(Q)$ otherwise.

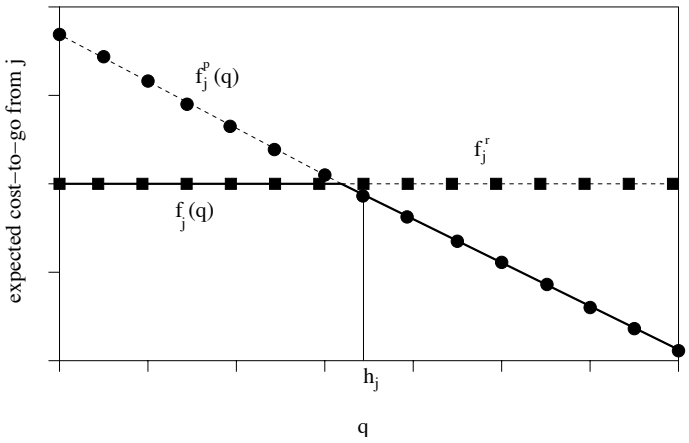


Figure 1.1: The bold line represents the cost function $f_j(q)$, that is, the expected cost-to-go of the vehicle from customer j on, under the restocking strategy. The quantity q is the residual capacity of the vehicle just after having serviced customer j. The function $f_j^p(q)$ would be the expected cost-to-go in case the vehicle always proceeded directly to customer $j+1$ without first going to de depot for replenishment. The function (constant in q) $f_j^r(q)$ would be the expected cost-to-go in case the vehicle always went to the depot for restocking, before going to the customer $j+1$. The figure shows that if the residual capacity q is under the threshold h_j, then it is more convenient to restock, otherwise it is more convenient to proceed to the next customer.

Algorithm 1 Computation of the VRPSD objective function $\mathbf{f_0(Q)}$

for $(q = Q, Q-1, ..., 0)$ **do**
  $f_n(q) = c_{n,0}$
  for $(j = n-1, n-3, ..., 1)$ **do**
    compute f_j^r using $f_{j+1}(\cdot)$
    for $(q = Q, Q-1, ..., 0)$ **do**
      compute $f_j^p(q)$
      compare f_j^r and $f_j^p(q)$ for finding the threshold h_j
      compute $f_j(q)$ using $f_{j+1}(\cdot)$
    end for
  end for
end for
compute $f_0(Q)$
return $f_0(Q)$

1.3 Examples of real-world applications

The VRPSD finds application in all those cases in which the orders of the costumers are modified upon arrival or where it is impossible to predict the demand of the costumers. We can think of two specific cases for which an optimization that keeps into account the stochasticity of the problem could help in obtaining improved routes and to reduce costs. The pick up of garbage is one of them: it is, indeed, impossible to know *a priori* how much garbage has to

be collected at each place. The delivery of petrol to petrol stations is another case subject to stochasticity of demand. When a customer issues the order it is still unknown how much he will sell in the time between the order and the new delivery. Other real-world applications can certainly be found. Nevertheless, it appears that companies are not used to consider demands expressed as probability distributions.

It is important to mention that in any real life VRP, besides the explicit constraints of the problem itself, there is always a set of implicit (and sometimes not formally defined) constraints, whose handling may not be trivial. An example is the shape of the tours in the solution: some people want them to be circle-like, others want them to be more radial, or they don't want to have tours including customers of two distinct regions (even if those regions are adjacent one another...). Another example of not formally defined constraint is to have, "when this is possible", certain customers in the last position of their tour. Furthermore, while in academic problems the vehicle capacity constraint is mono-dimensional, in real problems this is a multi-dimensional constraint in the sense that each vehicle has a maximum capacity in pallets, a maximum capacity in kilograms and a maximum capacity in cubic meters. So, when building solutions, one should check that none of the capacities are exceeded.

1.4 Literature review

One of the first approaches to stochastic routing problems are the works of Jaillet [54, 55] on the traveling salesman problem with random customers, and recently addressed by Bianchi et al. [12, 13]. The capacitated case is first analyzed by Jaillet and Odoni [56]. The works of Jaillet [54, 55] and Jaillet and Odoni [56] show how to efficiently compute the expected length of a solution. Bertsimas [7] and Bertsimas et al. [8] further analyze the VRPSD with one vehicle. Bertsimas [7] proposed the cyclic heuristic, by adapting to a stochastic framework one of the heuristics presented by Haimovitch and Rinnooy Kan [49] in a deterministic context. The algorithm proceeds in two phases. In the first phase a Traveling Salesman Problem (TSP) is solved, without considering customers' demands. This phase provides a tour $\tau = (0, 1, ..., n, 0)$ starting and ending at the depot. The second phase considers the n tours $\tau_i = (0, i, ..., n, 1, ..., i-1, 0)$, for $i = 1, ..., n$, and computes their expected length, $E[L_{\tau_i}]$. Out of these n tours, the best tour τ^* is chosen:

$$\tau^* = \arg \min_{i=1,...,n} E[L_{\tau_i}]. \tag{1.5}$$

Bertsimas et al. [8] improved the cyclic heuristic by applying dynamic programming, to supplement the *a priori* tour with rules for selecting returns trips to the depot, thus obtaining a mixed policy. Their computational experience suggests that the two versions of the cyclic heuristic provide good quality solutions. This heuristic is also shown to be asymptotically optimal under a random Euclidean model. Bertsimas [7] has also proposed a heuristic for the Vehicle Routing Problem with Stochastic Customers and Demands (VRPSCD). This problem is very similar to the VRPSD, but here one supposes to know whether the next customer will require zero demand in advance, before traveling to that customer. All these approaches fall within the *a priori* (and mixed) optimization framework, discussed in Bertsimas et al. [9]. Secomandi [81, 82] focuses

on computing on-line policies for the VRPSD, by applying a roll-out dynamic programming algorithm to sequentially improving a given *a priori* solution.

In case of a fleet of m vehicles, if balanced tours are not needed, one can arrange to have each of the $m-1$ customers that are closer to the depot served by one of the $m-1$ available vehicles. Then the remaining customers form a single vehicle instance that can be solved by the cyclic heuristic [36]. This adaptation of the cyclic heuristic has also shown to be asymptotically optimal in a random Euclidean model (see Fisher [36] and references cited therein). Nevertheless it does not seem to be a practical approach in a real situation, because one obtains $m-1$ single customer routes, and one route with $n-m+1$ customers. Yang et al. [93] investigate more sophisticated approaches to solve the multi vehicle problem, with a constraint on the expected distance traveled by each vehicle. They test two heuristic algorithms, the route-first-cluster-next and the cluster-first-route-next, which separately solve the problem of clustering customers which must be served by different vehicles and the problem of finding the best route for each cluster. Both algorithms seem to be efficient and robust.

Other authors have proposed different approaches to compute *a priori* policies for VRPSD with a fleet of vehicles. Gendreau et al. [42] present a stochastic integer programming method for VRPSD. This is the first exact method for this class of policies presented in the literature. By means of the integer L-shaped method [60] they solve instances with up to 46 and 70 customers and 2 vehicles, for the VRPSCD and VRPSD, respectively. The same authors also develop a tabu search heuristic algorithm called TABUSTOCH, for the same problem [43]. This algorithm is to be employed when instances become too large to be solved exactly by the L-shaped method. They report obtaining optimal solutions in 89.45% of the instances. The average deviation from optimality is only 0.38%, and in 97.8% is smaller than 5%. Computational times are considered reasonable.

Chapter 2

Metaheuristics

The term *metaheuristic* comes from the composition of two greek words. *Heuristic*, which derives from the verb *heurísco* ($\epsilon\upsilon\rho\acute{\iota}\sigma\kappa\omega$) that means "to search", and *meta* ($\mu\epsilon\tau\alpha$), which means "beyond, on a higher level".

The term metaheuristic, first proposed by Glover [45], has been used in the literature with different meanings. Only in the last years some researcher have proposed a general definition [75, 92]. We can cite for example the one given by Stützle:

> *Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local optima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made during the search. But, the main difference to pure random search is that in metaheuristic algorithms randomness is not used blindly but in an intelligent, biased form.* Stützle [86]

The *Metaheuristics Network* has adopted the following definition:

> *A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem.* Metaheuristics Network (2000)

Blum and Roli [16] have extrapolated a series of characteristic properties of metaheuristics:

- Metaheuristics are strategies that "guide" the search process. Their goal is to efficiently explore the search space in order to find (near-)optimal solutions.
- Metaheuristics may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics can be described on an abstract level (i.e., not tied to a specific problem).
- Metaheuristics often use the experience gained in previous searches (memory) to guide new searches.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy. Those strategies must be chosen in such a way to balance dynamically the exploitation of previously gained experience, called **intensification**, and the exploration of the search space, called **diversification**. This balancement is necessary, on one side, to quickly identify region in the search space where good solutions are, on the other side, to not loose to much time in searching inside regions that have already been explored or that seem not to have good solutions.

The rest of the chapter is organized as follows. There are several approaches to classify metaheuristics with respect to their properties. In Section 2.1 we briefly list and summarize different classification approaches; Sections 2.2 and 2.3 are devoted to a description of nowadays most important metaheuristics. We first describe the most relevant trajectory methods and then we outline population-based methods. In particular in Section 2.2.1 we analyze Iterated Local Search, in Section 2.2.2 we analyze Simulated Annealing, in Section 2.2.3 we analyze Tabu Search, in Section 2.3.1 we analyze Ant Colony Optimization and in Section 2.3.2 we analyze Evolutionary Computation.

2.1 Classification of metaheuristics

In the literature a great deal of criteria have been proposed to classify metaheuristics. Here we show the ones adopted by Stützle [86]. In our opinion none of these criteria, if considered singly, is capable of providing a neat and clear classification for the many methods that are present in the literature, especially considering the many hybrid algorithms that exist, anyway we find it most natural to describe metaheuristics following the "single point vs. population-based" search classification, which divides metaheuristics into trajectory methods and methods based on populations. This is motivated by the fact that this categorization permits a clearer description of the algorithms. Moreover, a current trend is the hybridization of methods in the direction of the integration of single point search algorithms in population-based ones.

- **Single point vs. population-based search**. One criterion that can be used to classify the algorithms is the number of solutions that are used at the same time: does the algorithm work on a population or on a single solution at any time? The algorithms that work with a single

solution are called *trajectory methods*. Methods that belong to this family are Tabu Search, Simulated Annealing and Local Search (LS). They all share the property that the search process describes a trajectory in the search space. Population-based methods, on the contrary, either perform search processes which can be described as the evolution of a set of points in the search space (as for example in Evolutionary Computation), or they perform search process which can be described as the evolution of a probability distribution over the search space (as for example in Ant Colony Optimization).

- **Dynamic vs. static objective function**. Another criterion used for the classification of metaheuristics concern the way they make use of the objective function. On one hand there are algorithms who do not change the objective function during run-time, on the other hand, there are methods like Guided Local Search, that modify it during the search. The idea behind this approach is to have chances to explore new area of the search space, even after a local optima is found. Modifications in the objective function introduce modification in the landscape of the search space, helping the diversification process.

- **One vs. various neighborhood structures**. Many metaheuristics work with a single neighborhood structure of a solution (a definition of neighborhood structure of a solution is given in Section 2.2). In other words, the search landscape topology does not change in the course of the algorithm. There are metaheuristics, like the Variable Neighborhood Search, that use a set of neighborhood structures which provides the possibility to diversify the search by swapping between different search landscapes.

- **Memory-based vs. memory-less methods**. An important criterion used in the classification of metaheuristics is the use they make of the search history, that is, whether they use memory or not.[1] Methods that take in consideration the part of the search already done are usually called methods with memory. Memoryless algorithms are by definition Markovian processes, since they rely only on the current solution to decide where to search in the next iteration. Among algorithms with memory we can make a distinction between the one with short term memory and the one with long term memory. Short term memory algorithms keep trace of a few visited solutions. Long term memory algorithms build indexes or accumulate a great deal of data on the visited solutions. Nowadays, memory is considered a fundamental component for the creation of a highly effective metaheuristic.

- **Nature-inspired methods vs. non-nature inspired methods**. The origins of the algorithms is a minor criterion, but probably one of the most intuitive to classify. Many methods takes inspiration from the natural world. Methods like Evolutionary Computation, Simulated Annealing and Ant Colony Optimization are clearly inspired by the natural world, while this is not true for methods like Tabu Search or Local Search. Practical

[1] Here we refer to the use of adaptive memory, in contrast to rather rigid memory, as used for instance in Branch & Bound.

problems may arise when we need to classify hybrid methods that do not fit either class (or, in a sense, they fit both at the same time).

It should be noticed that in the description of many nature-inspired methods the same terminology that is used for those phenomena is used for the description of the algorithms. For example, in Evolutionary Computation, more precisely in Genetic Algorithms (GA), it's common to call *gene* the codification scheme of a solution component, and to call *chromosome* the codification scheme of a solution. Again, the operators for random-variation of the solutions are called with the same terms used in the biological sciences, like *cross-over* and *mutation*. For what concern Ant Colony Optimization, we speak of *artificial ant colonies* that build solutions putting *pheromone* on solution components. Even though this terminology is not rigorous, we must realize that is in broad use, also in the literature, probably because the parallel with the natural world makes the comprehension much more intuitive in many cases.

2.2 Trajectory methods

In this section we outline metaheuristics referred to as *trajectory methods*. The term trajectory methods is used because the search process performed by these methods is characterized by a trajectory in the search space. Let's recall the concept of neighborhood structure and local minimum:

Definition 2.1 *A* **neighborhood structure** *is a function $\mathcal{N}$: $\mathcal{S} \rightarrow 2^{\mathcal{S}}$ that assigns to every $s \in \mathcal{S}$ a set of neighbors $\mathcal{N}(s) \subseteq \mathcal{S}$. $\mathcal{N}(s)$ is called the neighborhood of s. Often, neighborhood structures are implicitly defined by specifying the changes that must be applied to a solution s in order to generate all its neighbors. The application of such an operator that produces a neighbor $s' \in \mathcal{N}(s)$ of a solution s is commonly called a* **move**.

A neighborhood structure together with a problem instance define the topology of a so called search (or fitness) landscape [84, 85]. A search landscape can be visualized as a labeled graph in which the nodes are solutions (labels indicate their objective function value) and arcs represent the neighborhood relation between solutions. A solution $s^* \in \mathcal{S}$ is called a *globally minimal solution* (or global minimum) if for all $s \in \mathcal{S}$ it holds that $f(s^*) \leq f(s)$. The set of all globally minimal solutions is henceforth denoted by $\mathcal{S}^*$. The introduction of a neighborhood structure enables us to additionally define the concept of *local* minimum.

Definition 2.2 *A* **local minimum** *with respect to a neighborhood structure $\mathcal{N}$ is a solution s^* such that $\forall s \in \mathcal{N}(s^*) : f(s^*) \leq f(s)$. We call s^* a strict local minimum if $f(s^*) \leq f(s) \forall s \in \mathcal{N}(s^*)$.*

The search process of trajectory methods can be seen as the evolution of a discrete dynamical system [4, 26]. The algorithm starts from an initial state (the initial solution) and describes a trajectory in the state space. The system dynamics depends on the strategy used; simple algorithms generate a trajectory composed of two parts: a *transient* phase followed by an *attractor* (a fixed point, a cycle or a complex attractor). Algorithms with advanced strategies generate

more complex trajectories which can not be subdivided in those two phases. The characteristics of the trajectory provide information about the behavior of the algorithm and its effectiveness with respect to the problem instance that is tackled.

The performance of simple Local Search procedures (as outlined in Section 2.2.1) when applied to CO problems is usually quite unsatisfactory. Therefore, several techniques have been developed to prevent algorithms from getting trapped in local minima, which is done by adding mechanisms that allow them to escape from local minima. This implies the necessity of termination criteria other than simply reaching a local minimum. Commonly used termination criteria are a maximum CPU time, a maximum number of iterations, a solution s of sufficient quality is found, or the maximum number or iterations without improvement is reached.

2.2.1 Iterated Local Search

Local Search starts from a solution s, often randomly generated, and explores the neighborhood $\mathcal{N}(s)$. The pseudo-code is shown in Algorithm 2. There are two major ways of implementing the function `ChooseImprovingNeighbor`(). The first way is called *first-improvement*. A first-improvement function scans the neighborhood $\mathcal{N}(s)$ and returns the first solution that is better than s. In contrast, a *best-improvement* function exhaustively explores the neighborhood and returns one of the solutions with the lowest objective function value. An iterative improvement procedure that uses a first-improvement function is called *first-improvement local search*, respectively *best-improvement local search* (or steepest descent local search) in the case of a best-improvement function. Both methods stop at a local minima. Therefore, their performance strongly depends on the definition of a neighborhood structure $\mathcal{N}$. A Local Search algorithm partitions the search space $\mathcal{S}$ into so-called *basins of attraction* of local minima. The basin of attraction of a local minimum $s^* \in \mathcal{S}$ is the set of all solutions s for which a deterministic iterative improvement local search terminates in s^* when started from the initial solution s.

In practice, LS define a correspondence between the set $\mathcal{S}$ and the subset $\mathcal{S}^*$ of globally minimal solutions.

One of the most evolved method of LS is the metaheuristic ILS [69, 66, 86, 61, 62]. In general, an explorative local search method is effective if it is able to find high quality local minima, i.e., if it can find the basins of attraction of high quality local minima. However, when the search space is huge or when the

Algorithm 2 Local Search

input: an instance x of a CO problem
$s \leftarrow$ `GenerateInitialSolution`()
repeat
  $s^* \leftarrow$ `ChooseImprovingNeighbor`($\mathcal{N}(s)$)
  $s \leftarrow s^*$
until no improvement is possible
$s_{best} \leftarrow s$
output: s_{best}, "candidate" to optimal solution for x

Algorithm 3 Iterated Local Search

input: an instance x of a CO problem
$s \leftarrow$ `GenerateInitialSolution`()
$s^* \leftarrow \mathrm{LS}(s)$
repeat
  $s' \leftarrow$ `Perturbation`(s^*, *history*)
  $s'^* \leftarrow$ `LS`(s')
  $s^* \leftarrow$ `ApplyAcceptanceCriterion`(s^*, s'^*, *history*)
until termination conditions not met
$s_{best} \leftarrow s^*$
output: s_{best}, "candidate" to optimal solution for x

basins of attraction of high quality local minimum are small, this goal is difficult to reach. In these cases an effective local search method might be defined only on the set of local minima $\mathcal{S}^*$, instead of on the whole set $\mathcal{S}$. Unfortunately, in most cases there is no feasible way of introducing a neighborhood structure for $\mathcal{S}^*$. Instead, ILS algorithms perform a trajectory along local minima $s_1^*, s_2^*, \cdot, s_t^*$ without explicitly introducing a neighborhood structure on $\mathcal{S}^*$ by applying the scheme that is shown in Algorithm 3. At each iteration the current solution (which is a local minimum) is perturbed and a local search method is applied to the perturbed solution. Then, the local minimum that is obtained by applying the local search method is either accepted as the new current solution, or not. The importance of the *perturbation* is obvious: too small a perturbation might not enable the system to escape from the basin of attraction of the local minimum just found. On the other side, too strong a perturbation would make the algorithm similar to a random restart local search. Therefore, the requirement on the perturbation method is to produce a starting point for local search such that a local minimum different from the current solution is reached. However, this new local minimum should be *closer* to the current solution than a local minimum produced by the application of the local search to a randomly generated solution. The *acceptance criterion* acts as a counterbalance, as it filters and gives feedback to the perturbation action, depending on the characteristics of the new local minimum.

The design of ILS algorithms has several degrees of freedom in the generation of the initial solution, the choice of the perturbation method and the acceptance criterion. Furthermore, the *history* of the search process can be exploited both in form of short and long term memory. In the following we describe the three main algorithmic components of ILS.

`GenerateInitialSolution`(): The construction of initial solutions should be fast (computationally not expensive), and initial solutions should be a good starting point for local search. The fastest way of producing an initial solution is to generate it at random. Another possibility is to use constructive heuristics. It is worth underlining that an initial solution is considered a good starting point depending on the particular local search method applied and on the structure of the problem instance under consideration, thus the algorithm designer's goal is to find a good trade-off between speed and quality of solutions.

`Perturbation`(s^*, *history*): The perturbation is usually non-deterministic in order to avoid cycling. Its most important characteristic is the *strength*,

roughly defined as the amount of changes inflicted on the current solution. The strength can be either fixed or variable. In the first case, the distance between s^* and s' is kept constant, independently of the problem size. However, a variable strength is in general more effective, since it has been experimentally found that, in most of the problems, the bigger the problem instance size, the larger should be the strength. A more sophisticated mechanism consist of adaptively changing the strength. For example, the strength might be increased when more diversification is needed or decreased when intensification seems preferable.[2] A second choice is the mechanism to perform perturbations. This may be a random mechanism, or the perturbation may be produced by a (semi-)deterministic method (e.g., by a local search that is based on a neighborhood different from the one used in the main algorithm).

`ApplyAcceptanceCriterion`(s^*, s'^*, *history*): The third important component is the acceptance criterion. Two extreme examples are (1) accepting the new local minimum only in case of improvement and (2) always accepting the new solutions. In-between there are several possibilities. For example, it is possible to adopt an acceptance criterion that is similar to the one of Simulated Annealing (explained in Section 2.2.2). Non-monotonic cooling schedules might be particularly effective if they exploit the history of the search process. For example, when the recent history of the search process indicates that intensification seems no longer effective, a diversification phase is needed and the temperature is increased.

Reference of successful applications of ILS can be found in Lourenço et al. [62].

2.2.2 Simulated Annealing

Simulated Annealing is commonly said to be the oldest among the metaheuristics and surely one of the first algorithms that had an explicit strategy to escape from local minima. The origins of the algorithm are in statistical mechanics (see the Metropolis algorithm [67]). The idea of SA was provided by the annealing process of metal and glass, which assume a low energy configuration when cooled with an appropriate cooling schedule. SA was first presented as a search algorithm for CO problems in Kirkpatrick et al. [59] and Cerný [20]. In order to avoid getting trapped in local minima, the fundamental idea is to allow moves to solutions with objective function values that are worse than the objective function value of the current solution. Such a kind of move is often called an *uphill-move*. At each iteration a solution $s' \in \mathcal{N}(s)$ is randomly chosen. If s' is better than s (i.e., has a lower objective function value), then s' is accepted as new current solution. Otherwise, if the move from s to s' is an uphill-move, s' is accepted with a probability which is a function of a temperature parameter T_k and $f(s') - f(s)$. Usually this probability is computed following the Boltzmann distribution:

$$p(s'|T_k, s) = e^{-\frac{f(s')-f(s)}{T_k}}. \tag{2.1}$$

The dynamic process described by SA is a *Markov chain* [35], as it follows a trajectory in the state space in which the successor state is chosen depending only on the incumbent one. This means that basic SA is memory-less. However, the

[2] An ILS algorithm with such a perturbation scheme is very similar to a Variable Neighborhood Search (VNS) algorithm.

Algorithm 4 Simulated Annealing

input: an instance x of a CO problem
$s \leftarrow$ `GenerateInitialSolution`()
$k \leftarrow 0$
$T_k \leftarrow$ `SetInitialTemperature`()
while termination conditions not met **do**
  $s' \leftarrow$ `PickNeighborAtRandom`($\mathcal{N}(s)$)
  if $f(s') \leq f(s)$ **then**
    $s \leftarrow s'$;
  else
    accept s' as new solution with probability $p(s'|T_k, s)$
  end if
  `AdaptTemperature`(T_k)
end while
$s_{best} \leftarrow s$
output: s_{best}, "candidate" to optimal solution for x

use of memory can be beneficial for SA approaches (see for example Chardaire et al. [21]). The algorithmic framework of SA is described in Algorithm 4. The components are explained in more detail in the following.

`GenerateInitialSolution`(): The algorithm starts by generating an initial solution that may be randomly or heuristically constructed.

`SetInitialTemperature`(): The initial temperature is chosen such that the probability for an uphill-move is quite high at the start of the algorithm.

`AdaptTemperature`(T_k): The temperature T_k is adapted at each iteration according to a *cooling scheme*. The cooling scheme defines the value of T_k at each iteration k. The choice of an appropriate cooling scheme is crucial for the performance of the algorithm. At the beginning of the search the probability of accepting uphill-moves should be high. Then, this probability should be gradually decreased during the search. Note that this is not necessarily done in a monotonic fashion.

Theoretical results on non-homogeneous Markov chains [1] state that under particular conditions on the cooling schedule, the algorithm converges in probability to a global minimum for $k \to \infty$. A particular cooling scheme that fulfills the hypothesis for the convergence is the one that follows a logarithmic law. Hereby, T_k is determined as $T_k \leftarrow \frac{r}{\log k + c}$ (where c is a constant). Unfortunately, cooling scheme which guarantee the convergence to a global optimum are not feasible in applications, because they are too slow for practical purposes. Therefore, faster cooling scheme are adopted in applications. One of the most popular ones follows a geometric law: $T_k \leftarrow \alpha \times T_{k-1}$, where $\alpha \in (0, 1)$, which corresponds to an exponentially decay of the temperature.

The cooling scheme can be used for balancing between diversification and intensification. For example, at the beginning of the search, T_k might be constant or linearly decreasing in order to sample the search space; then, T_k might follow a rule such as the geometric one in order to make the algorithm converge to a local minimum at the end of the search. The cooling scheme and the initial temperature should be adapted to the particular problem instance considered, since the cost of escaping form local minima depends on the structure of the

search landscape. A simple way of empirically determining the starting temperature T_0 is to initially sample the search space with a random walk to roughly evaluate the average and the variance of objective function values. Based on the samples the starting temperature can be determined such that uphill-moves have a high probability.

Reference of successful applications of SA can be found in Fleischer [37], Ingber [53], Aarts et al. [1].

2.2.3 Tabu Search

Tabu Search is one of the most successful metaheuristic for the application to CO problems. The basic ideas of TS were introduced by Glover [45] in 1986, based on his earlier ideas [44]. A description of the method and its concepts can be found in Glover and Laguna [46]. The basic idea of TS is the explicit use of search history, both to escape from local minima and to implement an explorative strategy. We first describe a simple version of TS in order to introduce the basic concepts; then, we explain a more applicable algorithm.

A simple TS algorithm (see Algorithm 5) is based on a best-improvement local search (see Section 2.2.1) and uses a *short term memory* to escape from local minima and to avoid cycles.[3] The short term memory is implemented as a *tabu list* that keeps track of the most recently visited solutions and excludes them from the neighborhood of the current solution. In the following we will refer to the restricted neighborhood of a solution s as the *allowed set*, which we will denote by $\mathcal{N}_a(s)$. At each iteration the best solution from the allowed set is chosen as the new current solution. Furthermore, in procedure `Update`$(TabuList, s, s')$ this solution is added to the tabu list and—if the tabu list has reached its maximum capacity—one of the solutions that were already in the tabu list is removed. Tabu lists are usually handled in a FIFO manner. The algorithm stops when a termination criterion is met. It might also terminate if the allowed set is empty.

The use of a tabu list prevents from returning to recently visited solutions, therefore it prevents from endless cycling[4] and forces the search to accept even

[3] A cycle is a sequence of moves that constantly repeats itself.

[4] Since the tabu list has a finite length l which is smaller than the cardinality of the search space, cycles of higher period than l are still possible.

Algorithm 5 Simple Tabu Search

input: an instance x of a CO problem
$s \leftarrow$ `GenerateInitialSolution`()
$TabuList \leftarrow \emptyset$
while termination conditions not met **do**
  $\mathcal{N}_a(s) \leftarrow \mathcal{N}(s) \setminus TabuList$
  $s' \leftarrow \arg\min\{f(s") | s" \in \mathcal{N}_a(s)\}$
  `Update`$(TabuList, s, s')$
  $s \leftarrow s'$
end while
$s_{best} \leftarrow s$
output: s_{best}, "candidate" to optimal solution for x

Algorithm 6 Tabu Search

input: an instance x of a CO problem
$s \leftarrow$ `GenerateInitialSolution`()
`InitializeTabuList`($TabuList_1, \cdots, TabuList_r$)
while termination conditions not met **do**
  $\mathcal{N}_a(s) \leftarrow \{s' \in \mathcal{N}(s) | s'$ does not violate a tabu condition, or satisfies at least one aspiration criterion
  $s' \leftarrow \arg\min\{f(s") | s" \in \mathcal{N}_a(s)\}$
  `UpdateTabuList`($TabuList_1, \cdots, TabuList_r, s, s'$)
  $s \leftarrow s'$
end while
$s_{best} \leftarrow s$
output: s_{best}, "candidate" to optimal solution for x

uphill-moves. The length l of the tabu list—known in the literature as the *tabu tenure*—controls the memory of the search process. With small tabu tenures the search will concentrate on small areas of the search space. On the opposite, a large tabu tenure forces the search process to explore larger regions, because it forbids revisiting a higher number of solutions. The tabu tenure can be varied during the search, leading to more robust algorithms.

The implementation of short term memory in terms of a list that contains complete solutions is not practical, because managing a list of complete solutions is highly inefficient. Therefore, instead of the solutions themselves, the *solution components* that are involved in moves are stored in the tabu list. Since different types of moves that work on different types of solutions components can be considered, a tabu list is usually introduced for each type of solution component. The different types of solution components and the corresponding tabu lists define the *tabu conditions* which are used to filter the neighborhood of a solution and generate the allowed set. Storing solution components instead of complete solutions is much more efficient, but it introduces a loss of information, as forbidding for example the introduction of a certain solution component in a solution means assigning the tabu status to probably more than one solution. Thus, it is possible that unvisited solutions of high quality are excluded from the allowed set. To overcome this problem, *aspiration criteria* are defined which allow to include a solution in the allowed set even if it is forbidden by tabu conditions. The most commonly used aspiration criterion applies to solutions which are better than the best solution found so far. This more applicable tabu search algorithm is shown in Algorithm 6. Reference of successful applications of TS can be found in Glover and Laguna [46].

2.3 Population-based methods

Population-based methods deal in every iteration of the algorithm with a set (i.e., a population) of solutions rather than with a single solution. From this set of solutions the population of the next iteration is produced by the application of genetic operators that generally consider two or more solutions. Therefore, population-based algorithms provide a natural, intrinsic way for the exploration of the search space. Yet, the final performance strongly depends on the way

the population is manipulated. The most studied population-based methods in CO are Evolutionary Computation and Ant Colony Optimization. In EC algorithms, a population of solutions is modified by recombination and mutation operators, and in ACO a colony of artificial ants is used to construct solutions guided by the pheromone trails and heuristic information.

2.3.1 Ant Colony Optimization

Ant Colony Optimization is one of the newest metaheuristic for the application to CO problems. The basic ideas of ACO were introduced in Dorigo [27] and successively extended in Dorigo et al. [31, 29], Stützle and Dorigo [88], Dorigo and Stützle [32]. In this section we present the description of ACO given in Dorigo and Di Caro [28].

ACO was inspired by the foraging behavior of real ants. This behavior—as described by Deneubourg et al. [25]—enables ants to find shortest paths between food sources and their nest. Initially, ants explore the area surrounding their nest in a random manner. As soon as an ant finds a source of food, it evaluates quantity and quality of the food and carries some of this food to the nest. During the return trip, the ant deposits a pheromone trail on the ground. The quantity of pheromone deposited, which may depend on the quantity and quality of the food, will guide other ants to the food source. The indirect communication between the ants via the pheromone trails allows them to find the shortest path between their nest and food sources. This functionality of real ant colonies is exploited in artificial ant colonies in order to solve CO problems.

In ACO algorithms the pheromone trails are simulated via a parametrized probabilistic model that is called the *pheromone model*. The pheromone model consists of a set of model parameters whose values are called the *pheromone values*. The basic ingredient of ACO algorithm is a constructive heuristic that is used for probabilistically constructing solutions using the pheromone values. In general, the ACO approach attempts to solve a CO problem by iterating the following two steps:

- Solutions are constructed using a pheromone model, that is, a parametrized probability distribution over the solution space.
- The solutions that were constructed in earlier iterations are used to modify the pheromone values in a way that is deemed to bias the search toward high quality solutions.

The ACO metaheuristic framework is shown in Algorithm 7. It consists of three algorithmic components that are gathered in the `ScheduleActivities` construct. The `ScheduleActivities` construct does not specify how these three activities are scheduled and synchronized. This is up to the algorithm designer. In the following we explain these three algorithm components in more detail.

`AntBasedSolutionConstruction`(): As mentioned above, the basic ingredient of ACO algorithm is a constructive heuristic for probabilistically constructing solutions. A constructive heuristic assembles solutions as sequences of solution components taken from a finite set of solution components $\mathcal{C} = \{c_1, \cdots, c_n\}$. A solution construction starts with an empty partial solution $s^p = <>$. Then, at each construction step the current partial solution s^p is extended by adding a feasible solution component from the set $N(s^p) \subseteq \mathcal{C} \setminus s^p$, which is defined by the

Algorithm 7 Ant Colony Optimization

input: an instance x of a CO problem
while termination conditions not met **do**
  ScheduleActivities
    `AntBasedSolutionConstruction`()
    `PheromoneUpdate`()
    `DaemonActions`()
  end ScheduleActivities
  $s_{best} \leftarrow$ best solution in the population of solutions
end while
output: s_{best}, "candidate" to optimal solution for x

solution construction mechanism. The process of constructing solutions can be regarded as a walk (or a path) on the so-called *construction graph* $\mathcal{G}_{\mathcal{C}} = (\mathcal{C}, \mathcal{L})$ whose vertexes are the solution components $\mathcal{C}$ and the set $\mathcal{L}$ are the connections. The allowed walks on $\mathcal{G}_{\mathcal{C}}$ are hereby implicitly defined by the solution construction mechanism that defines the set $N(s^p)$ with respect to a partial solution s^p. The choice of a solution component from $N(s^p)$ is at each construction step done probabilistically with respect to the pheromone model, which consists of *pheromone trail parameters* $\mathcal{T}_i$ that are associated to components $c_i \in \mathcal{C}$. The set of all pheromone trail parameters is denoted by $\mathcal{T}$. The values of these parameters—the *pheromone values*—are denoted by τ_i. In most ACO algorithms the probabilities for choosing the next solution component—also called the *transition probabilities*—are defined as follows:

$$\mathbf{p}(c_i|s^p) = \frac{\tau_i^\alpha \cdot \eta(c_i)^\beta}{\sum_{c_j \in N(s^p)} \tau_j^\alpha \cdot \eta(c_j)^\beta}, \forall c_i \in N(s^p), \tag{2.2}$$

where η is a weighting function, which is a function that, sometimes depending on the current partial solution, assigns at each construction step a heuristic value $\eta(c_j)$ to each feasible solution component $c_j \in N(s^p)$. The values that are given by the weighting function are commonly called the *heuristic information*. Furthermore, α and β are positive parameters whose values determine the relation between pheromone information and heuristic information.

`PheromoneUpdate`(): In ACO algorithms we can find different types of pheromone updates. First, we outline a pheromone update that is used by basically every ACO algorithm. This pheromone update consists of two parts. First, a *pheromone evaporation*, which uniformly decreases all the pheromone values, is performed. From a practical point of view, pheromone evaporation is needed to avoid a too rapid convergence of the algorithm toward a sub-optimal region. It implements a useful form of *forgetting*, favoring the exploration of new areas in the search space. Then, one or more solutions from the current and/or from earlier iterations are used to increase the values of pheromone trail parameters on solution components that are part of these solutions. As a prominent example, we outline in the following the pheromone update rule that was used in Ant System (AS) [27, 31], which was the first ACO algorithm proposed. This update rule, which we henceforth call *AS-update*, is defined by

$$\tau_i \leftarrow (1-\rho) \cdot \tau_i + \rho \cdot \sum_{\{s \in \mathcal{G}_{iter} | c_i \in s\}} F(s), \tag{2.3}$$

for $i = 1, \cdots, n$, where $\mathcal{G}_{iter}$ is the set of solutions that were generated in the current iteration. Furthermore, $\rho \in (0, 1]$ is a parameter called evaporation rate, and $F : \mathcal{G} \rightarrow \Re^+$ is a function such that $f(s) < f(s') \Rightarrow F(s) \geq F(s'), \forall s \neq s' \in \mathcal{G}$. $F(\cdot)$ is commonly called the *quality function*. Note that in Dorigo [27], Dorigo et al. [31] the update rule in equation 2.3 was introduced without multiplying the added amount of pheromone by ρ. Later (as for example in Dorigo and Gambardella [30]) this was often done. However, as ρ is a constant it does not change the qualitative behavior of the algorithm. Other types of pheromone update are rather optional and mostly aim at the intensification or the diversification of the search process. An example is a pheromone update in which during the solution construction, when adding a solution component c_i to the current partial solution s^p, the pheromone value τ_i is immediately decreased. This kind of pheromone update aims at a diversification of the search process.

`DaemonActions`(): Daemon actions can be used to implement centralized actions which cannot be performed by single ants. Examples are the application of local search methods to the constructed solutions, or the collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a non-local perspective. As a practical example, the daemon may decide to deposit extra pheromone on the solution components that belong to the best solution found so far.

In general, different versions of ACO algorithms differ in the way they update the pheromone values. This also holds for the two currently best-performing ACO variants in practice, which are Ant Colony System (ACS) [30] and $\mathcal{MAX} - \mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [89]. In the following we briefly outline the peculiarities of these two algorithms.

Ant Colony System. The ACS algorithm was introduced to improve over the performance of AS. ACS is based on AS but shows some important differences. First, after each iteration it applies a pheromone update only using the best-so-far solution (i.e., the best solution found so far). Note that also the pheromone evaporation is only applied to the solutions components that are in the best-so-far solution. Second, the transition probabilities are defined by a rule that is called *pseudo-random-proportional* rule. With this rule, some construction steps are performed in a deterministic manner, whereas in others the transition probabilities are defined as in Equation 2.2. Third, during the solution construction the pheromone value of each added solution component is slightly decreased.

$\mathcal{MAX} - \mathcal{MIN}$ **Ant System**. $\mathcal{MM}$AS algorithms are characterized as follows. First, the pheromone values are limited to an interval $[\tau_{min}, \tau_{max}]$, with $0 < \tau_{min} < \tau_{max}$. Explicit limits on the pheromone values prevent that the probability for constructing a solution falls below a certain value greater than 0. This means that the chance of finding a global optimum never vanishes. Second, in case the algorithm detects that the search is too much confined to a certain area in the search space, a restart is performed. This is done by re-initializing all the pheromone values. Third, the pheromone update is always performed with either the iteration-best solution, the restart-best solution (i.e., the best solution found since the last restart was performed), or the best-so-far solution.

Recently, researchers have been dealing with finding similarities between ACO algorithms and other probabilistic learning algorithms such as Estimation of distribution algorithms (EDA) [71]. An extensive study on this subject has

been presented in Zlochin et al. [94], where the authors present a unifying framework for so-called model-based search (MBS) algorithms. An MBS algorithm is characterized by the use of a (parametrized) probabilistic model $M \in \mathcal{M}$ (where $\mathcal{M}$ is the set of all possible probabilistic models) that is used to generate solutions to the problem under consideration.

Reference of successful applications of ACO can be found in Dorigo and Stützle [32, 33].

2.3.2 Evolutionary Computation

Evolutionary computation algorithms are inspired by nature's capability to evolve living beings well adapted to their environment that cooperate or compete with other members of the population. EC algorithms can be characterized as computational models of evolutionary process that takes inspiration from the natural genetic variety and natural selection. At each iteration a number of operators is applied to the individuals of the current population to generate the individuals of the population of the next generation (iteration). Usually, EC algorithms use operators called *recombination* or *crossover* to recombine two or more individuals to produce new ones. They also use *mutation* or *modification* operators which cause a self-adaptation of individuals. The driving force in evolutionary algorithms is the *selection* of individuals based on their *fitness* (which can be based on the objective function or some other kind of quality measure). Individuals with a higher fitness have a higher probability to be chosen as members of the population of the next iteration (or as parents for the generation of new individuals). This corresponds to the principle of *survival of the fittest* in natural evolution. It is the capability of nature to adapt itself to a changing environment, which gave the inspiration for EC algorithms. There has been a variety of slightly different EC algorithms proposed over the years. Basically they fall into three different categories which have been developed independently of each other. These are **evolutionary programming** (EP) as introduced by Fogel [39], Fogel et al. [40], **evolutionary strategies** (ES) proposed by Rechenberg [79] and **genetic algorithms** initiated by Holland [51]. EP arose from the desire to generate machine intelligence. While EP originally was proposed to operate on discrete representation of finite state machines, most of the present variants are used for continuous optimization problems. The latter also holds for most present variants of ES, whereas GA are mainly applied to solve CO problems. Over the years there have been quite a few overviews and surveys about EC methods. Among those are the ones by Bäck [2], by Fogel [38], by Spears et al. [83] and by Michlewicz and Michalewicz [68]. In Calegari et al. [18] a taxonomy of EC algorithms is proposed. In the following we provide a CO-oriented introduction to EC algorithms. For doing this, we follow a work by Hertz and Kobler [50], which gives a good overview of the different components of EC algorithms and of the possibilities to define them. Algorithm 8 shows the basic structure of EC algorithms. In this algorithm, P, denotes the population of individuals. A population of offspring is generated by the application of *recombination* and *mutation* operators and the individuals for the next population are *selected* from the union of the old population and the offspring population. The main features of an EC algorithm are outlined in the following.

Description of the individuals: EC algorithms deal with populations of in-

Algorithm 8 Evolutionary Computation

input: an instance x of a CO problem
$P \leftarrow$ `GenerateInitialPopulation`()
`Evaluate`(P)
while termination conditions not met **do**
  $P' \leftarrow$ `Recombine`(P)
  $P'' \leftarrow$ `Mutate`(P')
  `Evaluate`(P'')
  $P \leftarrow$ `Select`(P'', P')
  $s_{best} \leftarrow$ best solution in P
end while
output: s_{best}, "candidate" to optimal solution for x

dividuals. These individuals are not necessarily solutions to the considered instance. They may be partial solutions, or sets of solutions, or any object which can be transformed into one or more solutions in a structured way. Mos commonly used in CO is the representation of solutions as bit-strings or as permutations of n integer numbers. Tree-structures or other complex structures are also possible. In the context of GA, individuals are called *genotypes*, whereas the solutions that are encoded by individuals are called *phenotypes*. This is to differentiate between the representation of solutions and solutions themselves. The choice of an appropriate representation is crucial for the success of an EC algorithm. Holland's schema analysis [51] and Radcliffe's generalization to formae [78] are examples of how theory can help to guide representation choices.

Evolution process: At each iteration it has to be decided which individuals will enter the population of the next iteration. This is done by a selection scheme. To choose the individuals for the next population exclusively from the offspring is called *generational replacement*. In some schemes, such as *elitist strategies*, successive generations overlap to some degree, i.e., some portion of the previous generation is retained in the new population. The fraction of new individuals at each generation is called the *generational gap* [24]. In a *steady state* selection, only a few individuals are replaced by offspring. Most EC algorithms deal with populations of constant size. However, it is also possible to have a variable population size. In case of a continuously shrinking population size, the situation in which only one individual is left in the population (or no crossover partners can be found for any member of the population) might be one of the stopping conditions of the algorithm.

Neighborhood function: A neighborhood function $\mathcal{N}_{EC} : I \rightarrow 2^I$ assigns to each individual $i \in I$ a set of individuals $\mathcal{N}_{EC}(i) \subseteq I$ whose members are permitted to act as recombination partners for i to create offspring. If an individual can be recombined with any other individual we talk about *unstructured* populations, otherwise we talk about *structured* populations.

Information sources: The most common form of information sources to create offspring (i.e., new individuals) is a couple of parents (two-parent crossover). But there are also recombination operators that operate on

more than two individuals to create a new individual (multi-parent crossover). More recent developments even use population statistics for generating the individuals of the next population. Examples are the recombination operators called Gene Pool Recombination [72] and Bit-Simulated Crossover [91] which make use of a probability distribution over the search space given by the current population to generate the next population.

Infeasibility: An important characteristic of an EC algorithm is the way it deals with infeasible individuals which might be produced by the genetic operators. There are basically three different ways to handle such a situation. The simplest action is to *reject* infeasible individuals. Nevertheless, for some highly constrained problems (e.g., for timetabling problems) it might be very difficult to find feasible individuals. Therefore, the strategy of *penalizing* infeasible individuals in the function that measure the quality of an individual is sometimes more appropriate (or even unavoidable). The third possibility consists in trying to *repair* an infeasible solution (see Eiben and Ruttkay [34] for an example).

Intensification strategy: In many applications it proved to be quite beneficial to use improvement mechanisms to increase the fitness of individuals. EC algorithms that apply a local search algorithm to each individual of a population are often called Memetic Algorithms [69, 70]. While the use of a population ensures an exploration of the search space, the use of local search techniques helps to quickly identify "good" areas in the search space.

Diversification strategy: One of the major difficulties of EC algorithms (especially when applying local search) is the premature convergence toward sub-optimal solutions. The simplest mechanism to diversify the search process is the use of a mutation operator. The simplest form of a mutation operator just performs a small random perturbation of an individual, introducing a kind of *noise*. In order to avoid premature convergence there are also a number of other ways of maintaining the population diversity. Probably the oldest strategies are *crowding* [24] and its close relative *preselection* [19]. Newer strategies are *fitness sharing* [47] and *niching* [63] which is a collective name, whereby the reproductive fitness allocated to and individual in a population is reduced proportionally to the number of other individuals that share the same region of the search space.

For an extensive collection of references to EC applications we refer to Bäck et al. [3].

Chapter 3

Preliminary experiments

The *Metaheuristics Network*[1] has spent fourteen months (from January 2003 to February 2004) to study the Vehicle Routing Problem with Stochastic Demands, a formulation of which has been given in Section 1.2. The author of the thesis participated to the research conducted at IRIDIA, and was in charge of implementing an algorithm of the metaheuristic ACO. In the initial three-months phase of the research, we compared, over four classes of instances, seven algorithms. A basic Random Restart Local Search (RRLS) that was used to set the minimum performance level to be achieved by the other algorithms,[2] an Iterated Local Search, whose implementation is described in Section 3.2.1, a Simulated Annealing, described in Section 3.2.3, a Tabu Search, described in Section 3.2.4, an Ant Colony Optimization, described in Section 3.2.5 and an Evolutionary Computation, described in Section 3.2.6. Moreover, an algorithm called Flim-Flam (FF) was also implemented (description in Section 3.2.2). All algorithms but the last were implemented following the implementation guidelines provided by the *Metaheuristics Network* that are describe in Section 3.1.

The analysis of the experimental results showed that the best performing algorithm was using a TSP approximation of the objective function. Hence in Section 3.4 we analyzed the similarities between TSP and VRP in order to exploit them for designing better algorithms.

3.1 The guidelines of the *Metaheuristics Network*

In this section we analyze the guidelines given by the *Metaheuristics Network* [87] for the production of the software to solve the Vehicle Routing Problem with Stochastic Demands, whose formulation was given in Section 1.2.

Several studies have been published which compare metaheuristics against each other, but sometimes it is rather difficult to interpret the results of the computational experiments. In particular, it is very difficult to do a fair and meaningful comparison among different metaheuristic approaches. Fairness is

[1] A description of the Research Training Network has been given in the Introduction

[2] To fail in achieve better performance than RRLS is to be considered a major failure

not guaranteed, for example, when the compared approaches are not coded in the same programming language or when their execution takes place on computers with different processors or different operating systems. Even when using the same programming language and the same platform, a comparison might be quite meaningless, due to potentially quite different data structures that are used to implement the different approaches.

Another big issue is reproducibility: as it is well established in natural sciences, an experiment should be described in a way which easily permits to repeat the same experiment at a different place. Yet, in computer science this approach is rather impracticable, because this would mean that the whole computer system on which the experiments are run is described in detail in both hardware and software—from processor and memory types, data storage to the version of the software: operating system, compiler, linker and libraries. And even if there was an accurate description of a computational experiment, a reconstruction could be impossible for example because of hardware or software which is no more available. As a consequence of this, the reader of an experimental study of algorithms usually has to rely on the published material.

In order to have a fair comparison of the implemented algorithms, the *Metaheuristics Network* has defined the following criteria:

- **Hardware and Operating System**: the experiments were performed at IRIDIA on a cluster of 26 Linux workstations. The system consists of 1 master and 25 slave machines, 6 of which were available for the experiments of the *Metaheuristics Network*. The configuration of the system is as follows: the master node is equipped with a CPU AMD Athlon XP 2400+ and 768 MB of RAM, while the 6 slaves have a CPU AMD Athlon XP 1800+ and 512 MB of RAM each. The computers are connected with a Gigabit Ethernet network 1000Mbit/s using network interface cards Intel Pro 1000 and a switch D-Link DGS-1016T. All computer run Debian GNU/Linux "Woody" with kernel 2.4.20 and DQS (distributed queuing system) 3.3.2.

- **Programming language**: to avoid a bias due to the choice of the programming language, all the metaheuristics are written in a common programming language (C++). The executables have been generated at IRIDIA on the cluster, in the following compilation environment: **gcc** 2.95.4, **glibc** 2.2.4, **binutils** 2.10.0.18-1.

- **Starter kit**: all nodes of the *Metaheuristics Network* were provided with the source code of the common C++ classes to manage the problem data structure, the input-output routines, the neighborhood structure (to be used by SA and TS) and the local search routine (to be used by ILS, ACO and EC).

- **Evaluation function**: a function that returns the "quality" of a solution is provided within the Starter Kit. The objective function value $f_0(Q)$ (the expected cost of the planned route) is calculated according to the equations 1.1–1.4 as explained in Section 1.2.

3.1.1 The Starter Kit

The Starter Kit contains the source code of some C++ classes to be used in the implementation of the algorithms.

- **Class `Problem`:** this class implements methods and data structures to load problem instances to be solved.
- **Class `Solution`:** this class implements methods and data structures to manage solutions in the VRPSD with preventive restocking policy (described in Section 1.1). Moreover it provides methods for computing the objective value of a solution and for exploring the neighborhood structure defined by the Or-Opt local search.
- **Class `Control`:** this class implements methods and data structures to parse the command line and to manage stopping criterion based on time or number of iterations. Moreover, it takes care to output solutions.
- **Classes `Timer`, `Random` and `Utility`:** these classes are service classes that implement methods and data structures to manage time, pseudo-random number[3] and mathematical structures like matrices.

The Or-Opt Local Search

The Or-Opt algorithm [74], has been quite successfully applied to the VRPSD in Yang et al. [93]. Given a starting tour, the basic operation of the Or-Opt algorithm consists of moving a string of size 3, 2, or 1 from one position to another in the tour. Checking if an Or-Opt move leads to a better tour may be done in two stages. First compute the saving from extracting the string from the tour, and second compute the cost of inserting it back somewhere else in the tour, after the extraction point.

Computing these costs and savings in the deterministic case is quite simple, since they can usually be computed in constant time. In the stochastic case, however, it is computationally demanding, because it requires the dynamic programming recursion of equations (1.1–1.4). This leads to an $O(nKQ)$ time for the computation of the cost and saving of just one Or-Opt move. In order to reduce the computational time, the following approximation scheme suggested in Yang et al. [93] have been used.

Given a string S of consecutive nodes in the a-priori tour, the approximate saving of extracting it from the tour is computed as follows. Let l and t be the nodes immediately preceding, resp. following, S in the tour, and let $f_l^{beforeExt}(q)$ and $f_t^{beforeExt}(q)$ be the corresponding cost vectors before the extraction of S. Apply one dynamic programming recursion step starting with cost vector $f_t^{beforeExt}(q)$ at node t back to node l, without considering the string S. Let $f_l^{afterExt}(q)$ be the resulting cost vector at node l, that is, after extracting S from the tour. Then, define the approximate extraction saving as a simple average over q of $f_l^{afterExt}(q) - f_l^{beforeExt}(q)$, that is,

$$\text{Approximate Extraction Saving} = \frac{\sum_{q=0}^{Q}(f_l^{afterExt}(q) - f_l^{beforeExt}(q))}{Q+1}, \quad (3.1)$$

[3]Class `Random` implements the algorithm `ran0` from the Numerical Recipes in C [77] .

The computation of the insertion cost of S between nodes i and j in the tour, is done analogously, if we assume that the insertion point (node i) is after the extraction point (node l). Let $f_i^{beforeIns}(q)$ be the cost vector at node i for the current tour, that is, before inserting S in the tour. Apply dynamic programming recursion starting with cost vector $f_j^{beforeIns}(q)$ at node j through nodes of the inserted string, and back to node i. Let $f_i^{afterIns}(q)$ be the resulting cost vector at node i, that is, after inserting the sting in the tour. Then, define the approximate insertion cost as a simple average over q of $f_i^{afterIns}(q) - f_j^{beforeIns}(q)$. That is,

$$\text{Approximate Insertion Cost} = \frac{\sum_{q=0}^{Q}(f_i^{afterIns}(q) - f_j^{beforeIns}(q))}{|S_i|}. \qquad (3.2)$$

The total approximate cost of an Or-Opt move is computed by subtracting the Approximate Extraction Saving from the Approximate Insertion Cost:

$$\text{Approximate Or-Opt Move Cost} = \text{equation (3.2) - equation (3.1).} \qquad (3.3)$$

Note that the cost vectors are assumed to be already available from the computation of the expected cost for the starting tour, thus, they do not need to be computed when evaluating the Approximate Insertion Cost. The only computations that must be done here are the evaluation of cost vectors $f_l^{afterExt}(q)$ and $f_i^{afterIns}(q)$, and the averages in equations (3.1) and (3.2). The dynamic programming recursion for the computation of the cost vectors requires $O(KQ)$, while equations (3.1) and (3.2) require $O(Q)$ time. Therefore, with the proposed approximation, the cost of an Or-Opt move can be computed in $O(KQ)$ time.

Algorithm 9 Or-Opt Local Search

1: **input:** an *a-priori* tour T
2: compute $f_0(Q)$ of tour T (Algorithm 1)
3: let $k = 3$
4: **for all** S^k, a set of successive nodes from T **do**
5: select at random a node from T after the node immediately preceding S^k and compute the Approximate Or-Opt Move Cost (equation 3.3).
6: **end for**
7: **if** there is no negative Approximate Or-Opt Move Cost **then**
8: go to Step 13.
9: **else**
10: select the set S^k that results in the most negative Approximate Or-Opt Move Cost, and perform the corresponding Or-Opt move
11: go to Step 4
12: **end if**
13: **if** $k = 1$ **then**
14: stop
15: **else**
16: decrease k by 1, and go to Step 4.
17: **end if**
18: **outoup:** an *a-priori* tour T'

The proposed approximation scheme has some potential drawbacks though. It neglects the influence of the inserted string (or deleted string) on the nodes

before node i. In principle it is thus possible that a certain Or-Opt move seems good, when evaluated by the approximation scheme, but it is actually a worsening move, when evaluated by the exact objective function computation. Therefore if the approximation scheme is used to evaluate moves in the Or-Opt algorithm (or in any other local search), there is no guarantee that a better tour than the starting one will be found. In practice this approximation should behave quite well since, as reported in Yang et al. [93], it should find the same route as the one obtained if the exact costs were computed, with significantly less computational effort (less than 10%). The algorithmic framework Or-Opt algorithm is described in Algorithm 9.

3.2 The Algorithms

In this section we give a brief description of the algorithms that were implemented for the preliminary experiments on the VRPSD. We want to highlight that for the first phase we have implemented very basic versions of the metaheuristics, in order to have a quick "preview" of the performance of the methods. We follow the classification given in Section 2.1, which divides metaheuristics into trajectory methods and methods based on populations.

3.2.1 Iterated Local Search

The ILS algorithm has been implemented by the INTELLEKTIK node[4] in Darmstadt. ILS is based on the simple yet powerful idea of improving a local search procedure by providing new starting solutions obtained from perturbations of the current solution, often leading to far better solutions than if using random restart. The local search is applied to the perturbed solution and a locally optimal solution is reached. If it passes an acceptance criterion, it becomes the new current solution; otherwise, one returns to the previous current solution. The perturbation must be sufficiently strong to allow the local search to explore new solutions, but also weak enough so that not all the good information gained in the previous search are lost. A detailed description of ILS algorithms can be found in Section 2.2.1, along with the pseudo-code in Algorithm 3.

Given the relations between TSP and VRPSD (analized in detail in Section 3.4), the `GenerateInitialSolution`() procedure consisted in solving the problem as a TSP, i.e., simply minimizing the distances. The algorithm used was an ILS algorithm which is described in Stützle and Hoos [90] and is one of the most high performance metaheuristics for the TSP. The initial solution was the best over $\frac{n}{log(n)}$ random restarts of the latter algorithm, where n is the number of customers. The `LS`() procedure was fixed to be the common Or-Opt Local Search provided with the Starter-Kit. Since only a random sample of the Or-Opt neighborhood was examined, we let `LS`() run for n times. The `ApplyAcceptanceCriterion`(s^*, s'^*) consisted in accepting s'^* if it is better than s^*, i.e., the best solution found so far. The `Perturbation`(s^*) consisted in a n random moves of a 2-exchange neighborhood, i.e., sub-tour inversion between two randomly chosen customers. This operator is broke if, within the n

[4] A description of the composition of the *Metaheuristics Network* is given in the Introduction.

moves, a solution is found having an objective function smaller than the objective function of the best solution plus a certain value ε. Given the instances tackled, $\varepsilon = \frac{n}{10}$ was empirically the best value found on some preliminary runs.

3.2.2 Flim-Flam

The FF algorithm has been implemented by the INTELLEKTIK node in Darmstadt. It's main principles are similar to ones from ILS. Therefore, the same name given to the ILS procedures are used here. However, this approach didn't follow the guidelines described in Section 3.1. Its purpose was to see how far we could go by using strategies for solving the VRPSD as a TSP. In fact, it consists in a very straightforward adaptation of the ILS described in Stützle and Hoos [90].

In this approach, the `GenerateInitialSolution`() procedure consisted in generating an initial solution by means of a Nearest Neighbor heuristic. The `LS`() procedure was a typical 3-opt first improvement local search. In addition, it applied two speed-up techniques, which are a fixed radius nearest neighbor within candidate lists [58] of 40 nearest neighbors for each city and *don't look bits* [6].
The `ApplyAcceptanceCriterion`(s^*, s'^*), as in the ILS version, consisted in accepting s'^* if it is better than the best solution found so far. Finally, the `Perturbation`(s^*) consisted in applying a double-bridge move [66] that cuts the current tour at four appropriately chosen edges into four sub-tours and reconnects these in a different order to yield a new starting tour for the local search.

3.2.3 Simulated Annealing

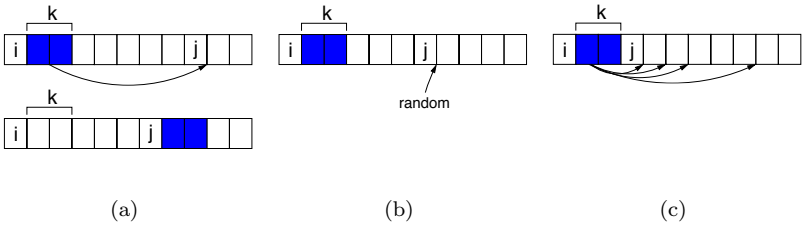
The SA algorithm has been implemented by the INTELLEKTIK node in Darmstadt. A detailed description of SA algorithms can be found in Section 2.2.2, along with the pseudo-code in Algorithm 4.

We now give a more detailed description on how the components have been implemented.

`GenerateInitialSolution`() : The following alternatives were considered:

- a random sequence of customers;
- the sequence of customers generated by a constructive heuristic for VRP;
- the sequence of customers generated by constructive heuristics for TSP (nearest neighbor heuristics, farthest insertion heuristic) [6];
- a sequence of customers obtained by solving the TSP by ILS.

From preliminary results it was noticed that the better the initial solution, the better will be the final solution for the VRPSD. Since good solutions for the TSP seemed to be good also for the VRPSD, it was decided to adopt the last alternative. The ILS algorithm used to find a tour that visit all the customers exactly once and that minimize the total length without caring of any capacity constraints, is one of the highest performing metaheuristics for TSP and is described in Stützle and Hoos [90]. Although it is not optimal it does provide good quality solutions to the TSP.



(a) (b) (c)

Figure 3.1: (a) An Or-Opt move of length, $k = 2$, two. Given a position i in the sequence of customers, the k elements after it are shifted after position j. In our case j has to be bigger than i (forward insertion). (b) (c) Two examination strategies. In (b), given k and i, j is selected randomly; next k and i are updated (examination alternative 1). In (c), j is let varying in label order until the end of the sequence is reached; next k and i are updated (examination alternative 2).

`PickNeighborAtRandom`($\mathcal{N}(s)$): The Or-Opt neighborhood appears particularly appealing for the VRPSD mainly because an Or-Opt move does not reverse any path in the route. Therefore its cost evaluation is relatively fast. Only Or-Opt of length 1, 2 and 3 were considered. As for the local search provided with the Starter Kit, backwards insertions were not considered, only forward. See Figure 3.1(a) for an example. For the neighborhood examination strategy three alternatives were considered. Let k be the length of the sequence to move (k in $\{1, 2, 3\}$) and i and j respectively the index of starting position and index of the ending position in the sequence. k, i, j are selected in the following way:

1. k looping in 3,2,1; i in label order; j randomly;
2. i in label order; j in label order; for each pair i and j $k = \{1, 2, 3\}$;
3. k randomly; i randomly; j randomly;

`AdaptTemperature`(T_k): The temperature profile is determined by three main features: the initial temperature, the cooling schedule and the re-heating schedule.

Initial Temperature. A sample of 100 elements in the neighborhood of the initial solution is used to compute the average variation in the evaluation function. The value found is then multiplied by a given factor, f, which is a parameter of the algorithm.

Cooling schedule. A non monotonic temperature schedule, realized by the interaction of two strategies, was used: a standard geometric cooling and a temperature re-heating. In the standard geometric cooling the temperature T_k is computed from T_{k-1} as $T_k = \alpha \times T_{k-1}$ where α is a parameter called the "cooling rate" ($0 < \alpha < 1$). The number of iterations at which the temperature remains constant is kept proportional to the size of the neighborhood as suggested in Johnson et al. [57]. Hence, Temperature lenght (TL) $= q \cdot |\mathcal{N}|$, where q is a parameter and $|\mathcal{N}|$ is the size of the neighborhood, which depends on the examination strategy selected.

Re-heating. When the search appears to stagnate, the temperature is increased by adding a quantity T^i to the current temperature. This is done when no improvement is found for a number of steps given by $r \cdot TL$ where r is a parameter. The solution that was consider for testing improvements is the best since the last re-heating occurred.

For selecting which neighborhood examination strategy and which move evaluation method are better and for tuning the parameters f, α, q and r, an experimental phase was set up. It involved 74 candidate versions of Simulated Annealing, obtained by combination of the elements described above, and 24 heterogeneous instances of the VRPSD. The times for each run of the algorithms were determined by 30 random restarts of the common local search on the instance under examination. The evaluation methodology used is the following: each algorithm was run once on each instance, the algorithms were ranked on the instances and the ranks were averaged over the instances. The algorithm with the best average rank was selected as representative of Simulated Annealing applied to VRPSD and submitted to compete against the other metaheuristics.

The algorithm selected uses the second neighborhood examination scheme described above and the approximated method to evaluate a move. But, surprisingly, it has temperature null. This means, that the acceptance criterion of Simulated Annealing is merely reduced to accepting all non worsening moves.

3.2.4 Tabu search

The TS algorithm has been implemented by the IDSIA node in Manno. A detailed description of TS algorithms can be found in Section 2.2.3, along with the pseudo-code in Algorithm 6. Like all TS algorithms, it is composed by three main elements: the starting solution, the neighborhood structure and the strategy of selection of new solutions. The starting solution was generated by the nearest neighbor traveling salesman problem heuristic. The neighborhood structure is derived from the OrOpt algorithm described in Section 3.1.1. The strategy of selection of new solutions is a descent-ascent method with the use of tabu list. A high level pseudo-code description of the implemented TS can be found in Algorithm 10. In the following some observation to explain the algorithm. The neighborhood is explored in such a way that at the beginning of the search, only moves with string length equal to 3 are tried. The string length is decreased each time that no best move can be found. Each time the string length has reached the minimum length (that is, 1), it is re-initialized to the maximum value (that is, 3).

In this TS the notion of neighbour_best_move is important, because the neighbour_best_move is the move that is actually performed and leads to a new solution. Each time a new potential move is considered, the procedure `update_neighbour_best`() evaluates it and decides if the move is a new neighbour_best_move or not. The decision criterion differs in the case of tabu or non-tabu moves. If the move is tabu, then, in order to be promoted to new neighbour_best_move, the move should lead to a new global best solution (according to the approximated evaluation done with `compute_proxy_neighbour_cost`()). If the move is not tabu, then, it's enough that it leads to a new best solution with respect to the the current neighborhood. Not all moves are evaluated, in

Algorithm 10 Tabu Search - preliminary experiments

```
k = 3
while termination conditions not met do
  if (k < 1) then
    k = 3
  end if
  set the length of tabu list as a random number in the interval [0.8 · (n − k − 1), n − k − 1], where n is the number of customers
  for all potential move among those with string length k do
    compute_proxy_neighbour_cost()
    check_if_move_is_tabu()
    update_neighbour_best()
  end for
  if there is a neighbour_best then
    currSol.shift(neighbour_best_move)
    update_true_global_best(start_solution)
    if not a new_true_global_best then
      decrease k by 1
    end if
    set_tabu_move()
  else
    tabu_move = neighbour_best_move
  end if
  tabu_list.push_back(tabu_move)
  prune_tabu_list()
end while
```

fact, tabu moves are evaluated only with a probability of the 30%, and non-tabu moves with a probability of the 80%. Also for this reason, sometimes no move is selected as new neighbour_best_move.

Each time a new neighbour_best_move is selected, it is put in the tabu list, in the following way: tabuMove[0] = currSol[i+1], tabuMove[1] = k, tabuMove[2] = currSol[j]. Moreover, the move is performed, and the exact objective value of the new solution is computed. In case there is no new neighbour_best_move, the last selected neighbour_best_move is put again in the tabu list. After, the oldest tabu move is removed from the list.

3.2.5 Ant Colony Optimization

The ACO metaheuristic has been implemented by the IRIDIA node, from the author of the thesis, and for this preliminary phase a basic Ant Colony System [30] has been chosen. A detailed description of ACO algorithms can be found in Section 2.3.1, along with the pseudo-code in Algorithm 7.

In the ACS, a set of agents, called ants, build solutions to the VRPSD cooperating through pheromone-mediated indirect and global communication. Here we give only a brief description of the basic principles that lie beneath the ACS. At each iteration of the algorithm, each of the m ants constructs an *a priori* tour one customer after the other so that starting with the depot all customers are visited one and only one time, ending again with the depot. The ants

choose the next customer to be visited probabilistically, guided by stigmergic information. No heuristic information is used in this first implementation.
The stigmergic information is in the form of a matrix of 'pheromone' values $\tau : C \times C \rightarrow \Re_{\geq 0}$, where C is the set of customers, which are an estimate of the utility of going from one customer i to a second one j in the *a priori* tour, as judged by previous iterations of the algorithm. In ACS at the beginning of the algorithm the values in the matrix are initialized to a parameter τ_0, except for those elements that belong to the starting solution, generated by the farthest insertion heuristic, who receive a "reinforcement" equal to r iterations of global update rule. After each construction step a local update rule is applied to the element of the matrix corresponding to the chosen customers pairs (i,j):

$$\tau(i,j) \leftarrow (1-\psi) \cdot \tau(i,j) + \psi \cdot \tau_0 \tag{3.4}$$

The parameter $\psi \in [0,1]$ is the pheromone decay parameter, which controls the diversification of the construction process. The aim of the local update rule is to permit to the m ants to choose different customers j for the same given customer i.
After all the customers have been visited, the Or-Opt local search described in Section 3.1.1 is applied to the candidate solution s. At the end of the iteration the global update rule is applied to all the entries in the pheromone matrix:

$$\tau(i,j) \leftarrow \begin{cases} (1-\rho) \cdot \tau(i,j) + \rho \cdot \frac{Q}{q(s_{best})} & \text{if } (i,j) \text{ is in } s_{\text{best}} \\ (1-\rho) \cdot \tau(i,j) & \text{otherwise} \end{cases} \tag{3.5}$$

where Q is a parameter controlling the amount of pheromone laid down by the update rule, and the function q measures the expected cost of a candidate solution s.
The parameters we have used for the algorithm are the following:

m	τ_0	α	ψ	ρ	Q	r
5	0.5	1	0.3	0.1	10^7	100

3.2.6 Evolutionary Computation

The EC algorithm has been implemented by the ECRG node in Edinburgh. A detailed description of EC algorithms can be found in Section 2.3.2, along with the pseudo-code in Algorithm 8.

Given the relations between TSP and VRPSD with the *a priori* strategy, it was decided to implement a simple memetic algorithm based on work on the TSP, and crossover and mutation operators were chosen from permutation based operators which work well for the TSP.

A small population size of 10 individuals is used. Initial solutions are built with the randomized farthest insertion heuristic, which builds a solution starting from a random customer and then shifts the tour to start at the depot, as required by the problem. Or-Opt local search is than applied to each member of the initial population. In the Steady State evolution process only one couple of parents reproduces at each generation. Tournament selection of size 2 is used to select which parents are going to be given the chance to reproduce. The crossover used in the final implementation is the Edge Recombination crossover (ER) [68] re-adapted to always start at the depot. OX crossover [68] re-adapted

to always start from the depot, and PMX crossover [68] were also implemented and tried, but ER crossover seems to work better. It tries to build an offspring exclusively from the edges present in both parents, as follows: Swap-based mu-

Algorithm 11 Evolutionary Computation - preliminary experiments

1: Create an edge list from both parents, that provides for each customer all the other customers connected to it in at least one of the parent;
2: start from the depot as current customer;
3: select the customer in the edge list of the current customer, with the smallest number of customers left in its edge list;
4: **if** there is no customer left in the edge list of the current customer **then**
5: select a non yet visited customer;
6: **end if**
7: the selected customer becomes the current customer;
8: update the edge list by removing the current customer from each adjacent customer list;
9: **if** the tour is complete **then**
10: stop.
11: **else**
12: go to step 3.
13: **end if**

tation swaps two adjacent customers, never the depot. It is applied with an adaptive mutation rate with a maximum probability of 0.5. Order based mutation that picks 2 customers at random and exchanges them, and inversion were also tried. Or-Opt local search is again applied at each generation to improve the quality of the offspring. The improved offspring then replaces the worst member of the population.

3.3 Results of the preliminary experiments

The algorithms compared in the initial phase are: a basic RRLS, an ILS, whose implementation is described in Section 3.2.1, an SA, described in Section 3.2.3, a TS, described in Section 3.2.4, an ACO, described in Section 3.2.5, an EC, described in Section 3.2.6, and FF, described in Section 3.2.2.

Instances

We generated a number of VRPSD instances that we classified into four classes:

(i) 45 instances were generated in which the coordinates of the customers are random integers chosen from a uniform distribution in the interval $[0, 99]$ and the spread is equal to 50 **(class rand_unif_n)**,

(ii) 50 instances were generated in which the coordinates of the customers are random integers chosen from a uniform distribution in the interval $[0, 99]$ and the spread varies in the interval $[0, 100]$ **(class rand_unif_r)**,

(iii) 45 instances were generated in which the coordinates of the customers are random integers chosen from a normal distribution around 2 centers

(so we have 2 clusters) in the interval $[0, 99]$ (so the customers can have coordinates that are negative or bigger than 99) and the spread is equal to 50 **(class rand_clust_n)**,

(iv) 50 instances were generated in which the coordinates of the customers are random integers chosen from a normal distribution around 2 centers (so we have 2 clusters) in the interval $[0, 99]$ (so the customers can have coordinates that are negative or bigger than 99) and the spread varies in the interval $[0, 100]$ **(class rand_clust_r)**.

All the instances are such that the probability distributions of the customers' demand is uniform over the spread.

Each algorithm was tested once on each instance for a time equal to the time needed by RRLS to complete 30 iterations. So the execution time varied on an instance-by-instance basis.

Figure 3.2 illustrates the results aggregated over the four classes of instances. Figures 3.3, 3.4, 3.5, and 3.6 illustrate the results obtained by the algorithms under analysis on the classes of instances **rand_unif_n**, **rand_unif_r**, **rand_clust_n**, and **rand_clust_r**, in this order.

3.3.1 How to read the tables and the graphics

For every class of instances and then for the entire set of instances we verify if differences in solutions found by the algorithms are statistical significant. We use the *Pairwise Wilcoxon rank sum* test [22] with *p-values*[5] adjustment method by Holm [52]. In the tables associated to the graphics we report for every pair of algorithms (A,B) the *p-values* for the null hypothesis "*The distributions of the solutions generated by A and by B are the same*".

The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

The diagrams show the distribution of the solutions found by the algorithms during independent executions. The results of all executions on the same instance are ordered by quality of the solution (expected cost) to determine the rank. Solutions are grouped by algorithm. In the *box-plot* the median is represented by the bar, the *box* represent the interval included in the quantile 25% e 75%; the whiskers extend to the more extreme points that are no more than 1,5 times the interquartile; the circles represent points who are outside this interval.

[5]The *p-value* of a test is the probability of obtaining a value more extreme than the one that is obtained, in the direction of the alternate hypothesis, if the null hypothesis is true.

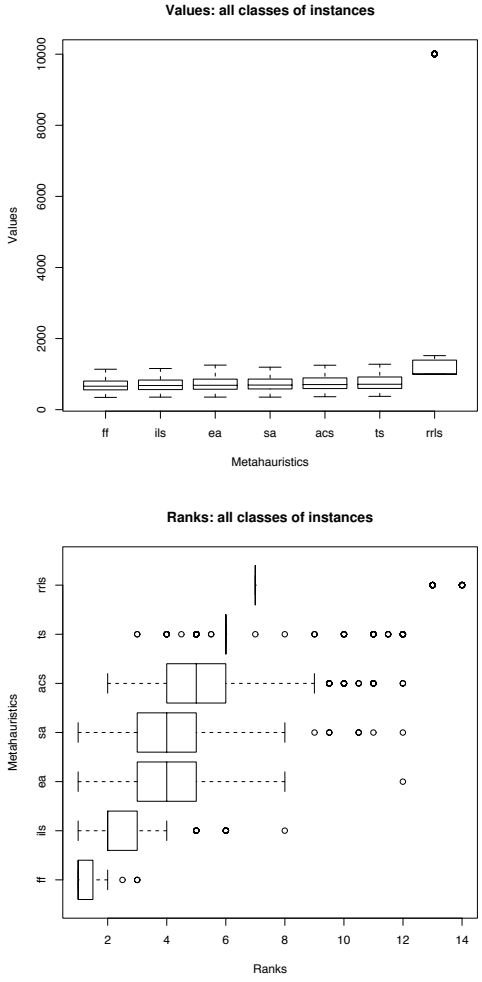


Figure 3.2: Preliminary experiments: results aggregated over the 4 classes of instances.

Table 3.1: *p-values* for the null hypothesis "The distributions of the solutions are the same" for aggregated classes of instances. The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

	ff	ils	ea	sa	acs	ts
ils	$< 2e-16$	—	—	—	—	—
ea	$< 2e-16$	$< 2e-16$	—	—	—	—
sa	$< 2e-16$	$< 2e-16$	0.45	—	—	—
acs	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$	—	—
ts	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$	—
rrls	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$	$< 2e-16$

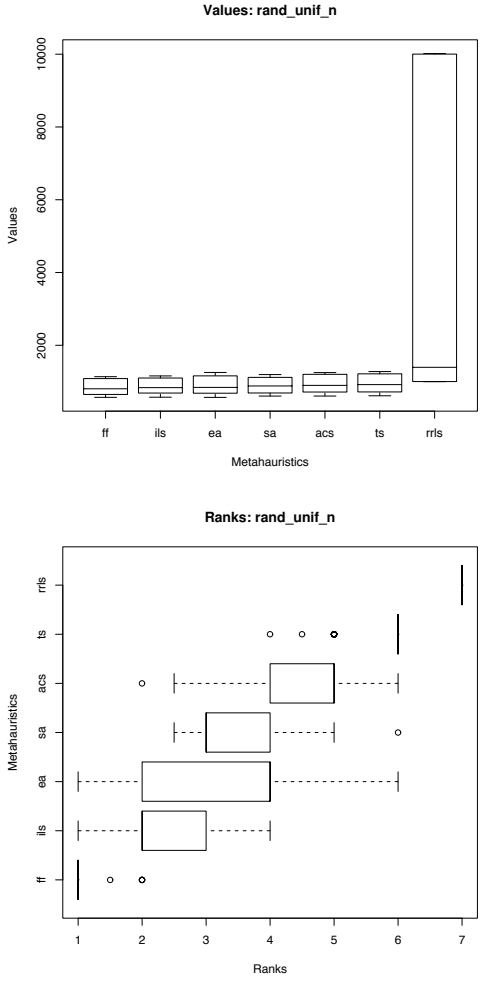


Figure 3.3: Preliminary experiments: results on the **rand_unif_n** class of instances.

Table 3.2: *p-values* for the null hypothesis "The distributions of the solutions are the same" for **rand_unif_n** class of instances. The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

	ff	ils	ea	sa	acs	ts
ils	$8.9e-08$	—	—	—	—	—
ea	$1.3e-12$	$2.2e-05$	—	—	—	—
sa	$1.2e-12$	$3.8e-07$	0.49	—	—	—
acs	$1.2e-12$	$1.4e-07$	$8.4e-06$	$2.1e-06$	—	—
ts	$1.2e-12$	$1.2e-12$	$4.6e-10$	$4.0e-11$	$8.4e-06$	—
rrls	$1.2e-12$	$1.2e-12$	$1.2e-12$	$1.2e-12$	$1.2e-12$	$1.2e-12$

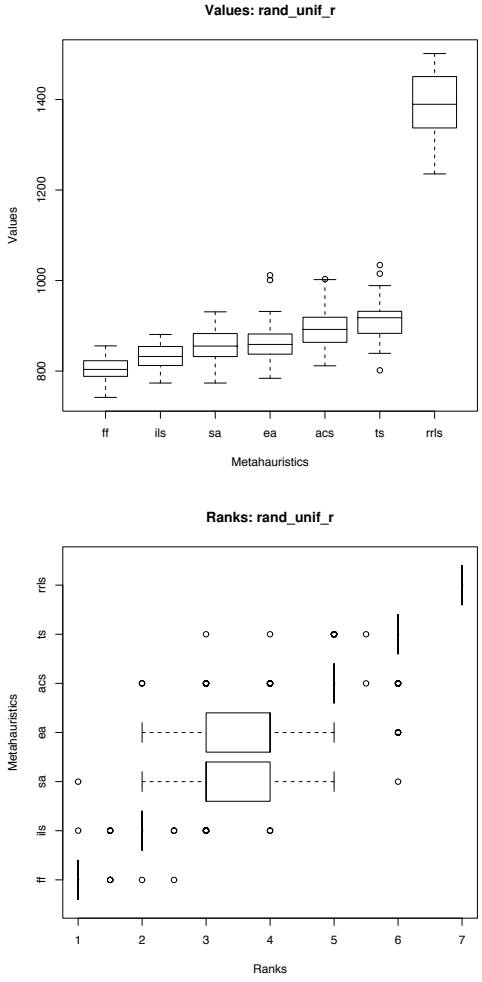


Figure 3.4: Preliminary experiments: results on the **rand_unif_r** class of instances.

Table 3.3: *p-values* for the null hypothesis "The distributions of the solutions are the same" for **rand_unif_r** class of instances. The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

	ff	ils	sa	ea	acs	ts
ils	$2.7e-08$	—	—	—	—	—
sa	$1.6e-08$	$2.6e-08$	—	—	—	—
ea	$1.6e-08$	$7.5e-07$	0.46	—	—	—
acs	$1.6e-08$	$1.6e-08$	$4.1e-07$	$1.4e-05$	—	—
ts	$1.6e-08$	$1.6e-08$	$1.8e-08$	$7.5e-07$	$6.6e-06$	—
rrls	$1.6e-08$	$1.6e-08$	$1.6e-08$	$1.6e-08$	$1.6e-08$	$1.6e-08$

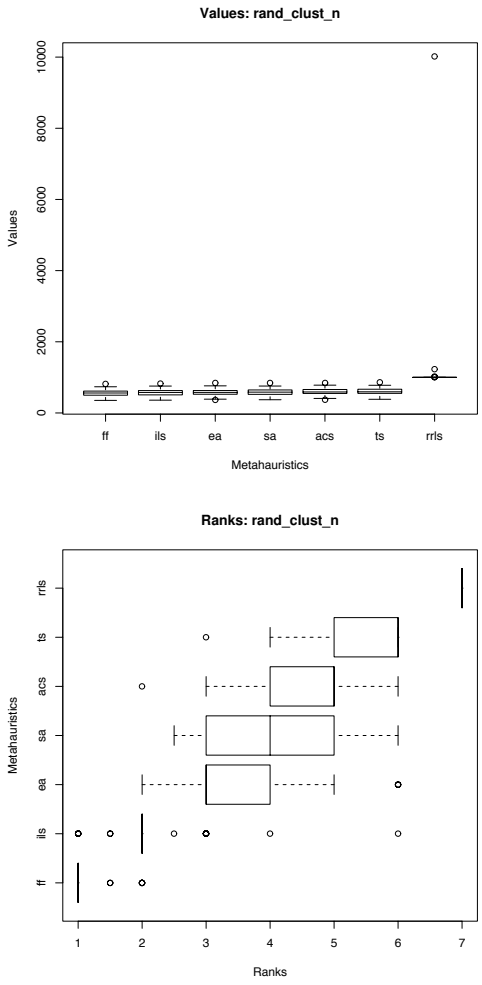


Figure 3.5: Preliminary experiments: results on the **rand_clust_n** class of instances.

Table 3.4: *p-values* for the null hypothesis "The distributions of the solutions are the same" for **rand_clust_n** class of instances. The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

	ff	ils	ea	sa	acs	ts
ils	$6.0e-07$	—	—	—	—	—
ea	$1.2e-12$	$1.7e-06$	—	—	—	—
sa	$1.2e-12$	$2.4e-07$	0.14	—	—	—
acs	$1.2e-12$	$3.9e-10$	$2.4e-07$	$9.7e-05$	—	—
ts	$1.2e-12$	$3.4e-11$	$6.3e-11$	$2.3e-08$	$1.1e-05$	—
rrls	$1.2e-12$	$1.2e-12$	$1.2e-12$	$1.2e-12$	$1.2e-12$	$1.2e-12$

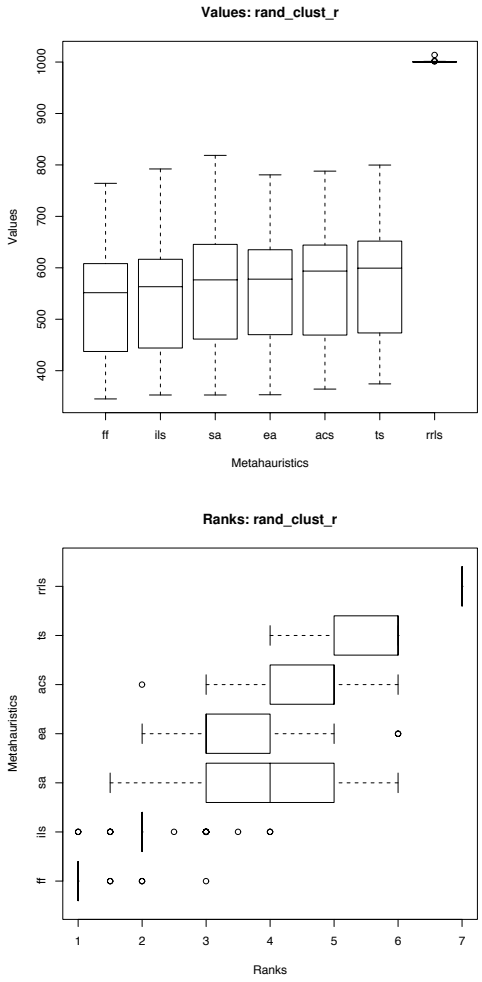


Figure 3.6: Preliminary experiments: results on the **rand_clust_r** class of instances.

As it can be observed from the plots, FF is by far the best algorithms. This shows that, at least on the instances considered in this research, a TSP approximation of the cost function is particularly effective. A tentative analysis of the reasons of the success of such approach is given in Section 3.4. On the other hand, Section 3.4.1, discusses the limitations and possible risks of such approach.

Another element that is worth noticing in the results is that the relative performance of the algorithms under analysis is similar across the different classes of instances: FF is always the best performing metaheuristic followed by ILS; the distribution of the solutions generated by EC and SA over the aggregated classes of instances is the same. Last come ACO and TS. In any case, all the metaheuristics considered here obtain significantly better results than RRLS.

Table 3.5: *p-values* for the null hypothesis "The distributions of the solutions are the same" for **rand_clust_r** class of instances. The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

	ff	ils	sa	ea	acs	ts
ils	$2.9e-08$	—	—	—	—	—
sa	$1.6e-08$	$5.1e-08$	—	—	—	—
ea	$1.6e-08$	$5.6e-06$	0.06245	—	—	—
acs	$1.6e-08$	$2.3e-08$	0.00036	$8.6e-07$	—	—
ts	$1.6e-08$	$1.6e-08$	$2.3e-06$	$2.9e-08$	$1.6e-06$	—
rrls	$1.6e-08$	$1.6e-08$	$1.6e-08$	$1.6e-08$	$1.6e-08$	$1.6e-08$

3.4 Relation between TSP and VRPSD

By definition, the TSP and the VRP are closely related, since both are concerned with the determination of routes on a graph such that a certain cost is minimized. In fact, if there is only one vehicle with infinite capacity, the VRP can be seen as a *simple* TSP.

Given the fact that very fast algorithms for solving the TSP to optimality are known, one should expect that they can also be applied in the same manner to solve the VRP. However, the addition of a constraint on the vehicle's capacity makes impossible their direct usage. Therefore, when the vehicle's capacity is low, an optimal TSP tour can not be anymore optimal for the corresponding VRP. Anyway, if capacity is not so extremely low when compared with the existing demand, an optimal TSP tour can be a good starting solution for applying a heuristic-based algorithm for solving the VRP.

In the case of the VRPSD, we observed a higher performance of the metaheuristics using search strategies typical for solving the TSP, i.e, minimizing total traveling distance. This could mean that, due to the stochasticity of the demand, the minimization of constraint violations considering the capacity of the vehicle becomes less important than minimizing its total traveling distance. Hence, intuitively, it would be preferable to reduce the cost of traveling between nodes, since one is not sure when has to go back to the depot. Another simpler hypothesis is concerned with the extremely fast evaluation for TSP tours which allows far more extensive exploration of solutions than performed by the approximation to the VRPSD objective function.

We conjecture that TSP-like strategies are the main key for tackling the VRPSD instances that were generated. However, we also believe that not all VRPSD instances could be tackled by a TSP approach. For instance, an instance which has a depot far away from all customers would be hardly solved by using such approach. More research with different kind of instances has to be done to verify our conjecture.

3.4.1 Pathological cases

Given the similarities between the TSP and the VRPSD with the *a priori* strategy, and the fact that the algorithms relying on TSP strategies seem to work quite well on the VRPSD, one could wonder if an optimal solution for the TSP can be arbitrarily far from optimal for the VRPSD. Indeed some pathological instances can be built for which the difference between an optimal solution for the TSP, regarded as an *a priori* tour for the VRPSD, and an optimal solution for the VRSPD can be arbitrarily large.

A very simple example is shown in Figure 3.7 where we can move the depot D far enough from the customers 1, 2, 3 and 4 as to meet any arbitrary difference between the expected cost of the *a priori* tour given by an optimal solution for the TSP represented by the continuous line, and the expected cost of an optimal solution for the VRPSD not based on TSP, where the *a priori* tour is given by the dotted line. Other more complex examples, involving a higher number of customers and/or larger demand spread, can be thought of.

On the other hand one could question the *a priori* strategy for the VRPSD and ask if the expected cost of an *a priori* tour is a good estimate for the cost of an actual tour realization. Here again pathological cases can be found where

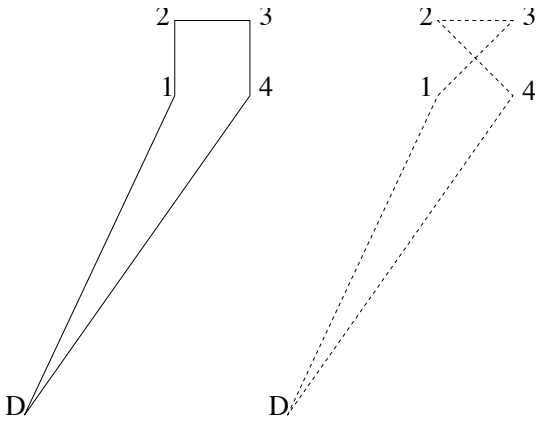


Figure 3.7: In this VRPSD instance the capacity Q of the vehicle is a multiple of 6, the demand of the customers is $2/3Q$, $Q/2$, $1/3Q$ and $Q/2$ respectively for customers 1, 2, 3 and 4, with spread 0 for each of them. The distance between the depot D and the other customers must be much bigger than the distance between any two customers. On the left the optimal tour for the TSP is shown, but under the above assumptions it is far from optimal for the VRPSD because the vehicle has to restock twice with this a-priori tour. On the right is shown a tour where the vehicle has to restock only once, making its expected cost as far from the one of the tour on the right as twice the distance between the depot and the customers.

the difference between the expected cost of an *a priori* tour can be arbitrarily far from the actual cost of the tour realization once the customers' demands are known. An idea is that of designing an instance where the probability of restocking at a given customer sufficiently far from the depot is very small, and therefore the cost of restocking gives a very small contribution to the expected cost formula, but indeed restocking can occur and when it does the cost of the actual tour is far from the expected cost. Of course in average the expected cost formula still gives a good estimate in this case.

Chapter 4

Further experiments

After the initial three-months phase of research, described in Chapter 3, the results of the preliminary experiments showed that higher performances were observed for those metaheuristics that were using search strategies typical for solving the TSP. This suggested further research in the direction of exploiting the similarities between TSP and VRPSD. For this purpose we modified the original Starter Kit described in Section 3.1.1 adding new components that are described in Section 4.1. It was decided to implement each metaheuristic using two different approximated objective functions during the local search (the implementations are described in Section 4.2), hence we compared two versions of the "usual" five metaheuristics (ILS, SA, TS, ACO and EC) and two versions of the RRLS . Two state-of-the-art algorithms to solve the TSP and the Capacitated Vehicle Routing Problem (CVRP) have also been used for the comparison. The analysis of the experimental results showed that the implemented algorithms performed poorly compared to the RRLS, sometimes even worse. Even if the algorithms that were using the TSP objective function performed better than those that were using the VRPSD objective function, the TSP state-of-the-art algorithm performed quite poorly compared to the other metaheuristics. Section 4.3 gives a tentative analysis of the reasons of this lack of success and Section 4.4 propose a new approximation of the VRPSD cost function called "homogeneous failure and restocking approximation" (Section 4.4.1), and a possible formulation of the VRP with deterministic demand (Section 4.4.2) to be used as the core of different real-world cases.

4.1 Modification to the Starter Kit

Starting from the results of the preliminary experiments, we decided to to design an experimental framework that allows us to answer the following research questions:

- how the metaheuristics compare each other on a fair basis?
- how much exploitation of the similarities between TSP and VRPSD can improve the algorithms performance?
- how TSP and CVRP solutions are related to VRPSD ones?

The Starter Kit was modified adding this new components:

- a method to generate solutions via the randomized farthest insertion heuristic;
- an objective function that evaluates the solution in a TSP-like way (it calculates tour length instead of expected tour length).

It was decided to implement every metaheuristic using the two different approximated objective functions during the local search:

- VRPSD proxy objective function (Or-Opt-0);
- TSP objective function (Or-Opt-tsp).

In the following we will refer the metaheuristic of these groups as respectively *-0* and *-tsp*. Within each group the metaheuristics are fairly comparable. Furthermore the differences in performance between the two groups for the same metaheuristic gives information about the exploitation of TSP similarities.

4.1.1 The Or-Opt local search

The two versions of the local search, Or-Opt-0 and Or-Opt-tsp, differ in the way the neighboring solutions are evaluated. The first version exploits the VRPSD objective function (that is, the expected cost of the a-priori tour), while the second version exploits the TSP objective function, that is, the length of the a-priori tour. Or-Opt-0 has been already described in Section 3.1.1, while in the following we give a description of Or-Opt-tsp.

The Or-Opt-tsp local search This version of the Or-Opt local search computes the cost of an Or-Opt move by just making the difference between the length of the tour after the move has been applied and the length of the original tour. Such difference may be computed in constant time, by considering only the arcs involved in the move. For instance, suppose we want to move a string S of consecutive nodes to another position in the tour, like in figure 4.1.1. Let l and t be the nodes immediately preceding, respectively following, S in the original tour, let i and j be the nodes between whom S is to be inserted, and let m and k be the first and the last nodes of S. Then, the move cost in the Or-Opt-tsp sense may be computed as follows:

$$\text{Move Cost} = d(l,t) + d(k,j) + d(i,m) - d(l,m) - d(k,t) - d(i,j). \qquad (4.1)$$

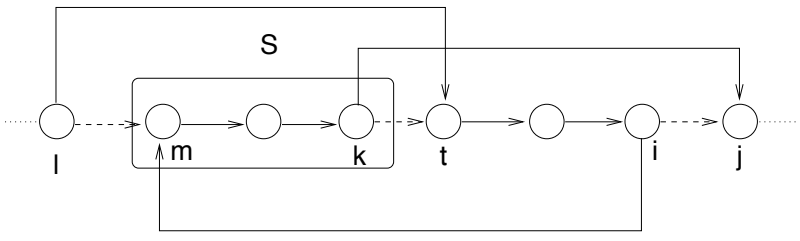


Figure 4.1: Arcs involved in the computation of the cost move in the Or-Opt-tsp local search.

Algorithm 12 Or-Opt-tsp Local Search

1: **input:** an *a-priori* tour T
2: compute $f_0(Q)$ of tour T (Algorithm 1)
3: let $k = 3$
4: **for all** S^k, a set of successive nodes from T **do**
5: select at random a node from T after the node immediately preceding S^k and compute the Move Cost (equation 4.1).
6: **end for**
7: **if** there is no negative Approximate Or-Opt Move Cost **then**
8: go to Step 13.
9: **else**
10: select the set S^k that results in the most negative Approximate Or-Opt Move Cost, and perform the corresponding Or-Opt move
11: go to Step 4
12: **end if**
13: **if** $k = 1$ **then**
14: stop
15: **else**
16: decrease k by 1, and go to Step 4.
17: **end if**
18: **outoup:** an *a-priori* tour T'

Apart from the way the move cost is evaluated, the Or-Opt-tsp explores the neighborhood of a solution in the same way as the Or-Opt-0, as outlined in Algorithm 12.

4.1.2 The randomized farthest insertion heuristic

The farthest insertion heuristic is a very simple heuristic for the TSP problem [6]. It builds a tour by choosing as next customer the not-yet-visited customer which is farthest, that is, the feasible customer that corresponds to the biggest distance from the current customer.

The randomized farthest insertion heuristic for the VRPSD builds a solution starting from a customer chosen at random. After the tour has been completed, it shifts the tour to start at the depot, as required by the problem.

4.2 The Algorithms

In this section we give a brief description of the algorithms that were implemented for the further experiments on VRPSD. Most of the algorithms are equal to the one implemented for the preliminary experiments (described in Section 3.2) with the exceptions of the components to generate initial solutions and the two local searches. In addition to the five "usual" metaheuristics (ILS, SA, TS, ACO and EC), two state-of-the-art algorithms to solve TSP and CVRP have also been used for the comparison. Unluckily the source codes of those last two algorithms were not public available, so we cannot provide a description of their internal working.

4.2.1 Iterated Local Search

The two ILS algorithms have been implemented by the INTELLEKTIK node in Darmstadt. A detailed description of ILS metaheuristic can be found in Section 2.2.1, along with the pseudo-code in Algorithm 3.

The main algorithmic components of ILS were implemented as follows:

`GenerateInitialSolution`(): this procedure consist in applying the farthest insertion heuristic described in Section 4.1.2.

`LS`(): this procedure was fixed to be the common Or-Opt local search (Or-Opt-0 for ILS-0 and Or-Opt-tsp for ILS-tsp).

`ApplyAcceptanceCriterion`(s^*, s'^*): The acceptance criterion consisted in accepting s'^* if it is better than s^*, i.e., the best solution found so far, according to the exact VRPSD objective function.

`Perturbation`(s^*): the perturbation consisted in a loop of n random moves of a 2-exchange neighborhood, i.e., subtour inversion between two randomly chosen customers. The loop is broken if, within n moves, a solution is found having an objective value smaller than the objective value of the best solution according to the exact VRPSD objective function, plus a certain value ε. Given the instances tackled, $\varepsilon = \frac{n}{10}$ was empirically the best value found on some preliminary runs. Otherwise, the best perturbed solution found is returned.

4.2.2 Simulated Annealing

The two SA algorithms have been implemented by the INTELLEKTIK node in Darmstadt. A detailed description of SA metaheuristic can be found in Section 2.2.2, along with the pseudo-code in Algorithm 4.

The main algorithmic components of SA were implemented as follows:

`GenerateInitialSolution`(): this procedure consist in applying the farthest insertion heuristic described in Section 4.1.2.

`PickNeighborAtRandom`($\mathcal{N}(s)$): the examination of the neighborhood uses the strategy adopted in the common Or-Opt local search.

`Acceptance Criterion`: the acceptance criterion is the element that differentiates the two version of SA. In SA-0, $f(s') - f(s)$ is the value given by the the Approximate Or-Opt Move Cost as computed described in Section 3.1.1; while in SA-tsp $f(s') - f(s)$ is simply the difference between the length of tour s' and the length of tour s.

`AdaptTemperature`(T_k): the same procedure used in the preliminary experiments was used (see details in Section 3.2.3).

For tuning the parameters f, α, q and r the same procedure used in the preliminary experiments was used (see details in Section 3.2.3).

For SA-tsp the tuning procedure indicated the following values: $f = 0.05$, $\alpha = 0.98$, $q = 1$ and $r = 20$.

For SA-0, instead, we had a surprising result. The best configuration has temperature null. This means, that the acceptance criterion of SA is merely reduced to accepting all non worsening moves. Hence, SA does not provide any improvement to the common local search.

4.2.3 Tabu Search

The two TS algorithms have been implemented by the IDSIA node in Manno. A detailed description of TS metaheuristic can be found in Section 2.2.3, along with the pseudo-code in Algorithm 6.

The main algorithmic components of TS were implemented as follow:

`GenerateInitialSolution`(): the starting solution was generated by the randomized farthest insertion heuristic described in Section 4.1.2.

The neighborhood structure is derived from the Or-Opt local search, described in Section 3.1.1. The strategy of selection of new solutions is a descent-ascent method with the use of tabu list. Algorithms 13 and 14 outline the two versions of TS that we have developed, TS-0 based exclusively on the VRPSD objective function, and TS-tsp also exploiting the TSP objective for speeding up the neighborhood exploration. The emphasized lines of the two procedures correspond to instructions that are different in TS-0 and in TS-tsp. Now some

Algorithm 13 Tabu Search - TS-0

```
set the length k of the strings to be moved as k = 3;
while time is not over do
  if (k < 1) then
    set k = 3
  end if
  set the length of tabu list as a random number in the interval [0.8 · (n − k − 1), n − k − 1], where n is the number of customers;
  for all potential moves among those with string length k do
    compute the VRPSD-like approximate move cost by equation (3.3);
    update the best-move-of-the-neighborhood;
    check if the move is tabu;
    update the best-move-of-the-neighborhood;
  end for
  if a best-move-of-the-neighborhood exists then
    modify the current solution by performing the best-move-of-the-neighborhood;
    update the best-so-far solution since the start of the TS;
    if the new current solution is not a new best-so-far solution then
      decrease the string length k by 1;
    else
      set k = 3;
    end if
    set the tabu-move;
  else
    tabu-move = best-move-of-the-neighborhood;
  end if
  add the tabu-move at the end of the tabu list;
  prune the tabu list from the beginning, so that its length is equal to the tabu length set at the beginning;
end while
```

observations in order to explain the pseudocode.

Neighborhood exploration The neighborhood is explored in such a way that

Algorithm 14 Tabu Search - TS-tsp

set the length k of the strings to be moved as $k = 3$;
while time is not over **do**
  if $(k < 1)$ **then**
    set $k = 3$
  end if
  set the length of tabu list as a random number in the interval $[0.8 \cdot (n-k-1), n-k-1]$, where n is the number of customers;
  for all potential moves among those with string length k **do**
    compute a tsp-like move cost by equation (4.1);
    check if the move is tabu;
    update the best-move-of-the-neighborhood;
  end for
  if *a best-move-of-the-neighborhood exists* **and** *performing the move would lead to a better solution in terms of the VRPSD objective function* **then**
    modify the current solution by performing the best-move-of-the-neighborhood;
    update the best-so-far solution since the start of the TS;
    if the new current solution is not a new best-so-far solution **then**
      decrease the string length k by 1;
    else
      set $k = 3$;
    end if
    set the tabu-move;
  else
    tabu-move = best-move-of-the-neighborhood;
  end if
  add the tabu-move at the end of the tabu list;
  prune the tabu list from the beginning, so that its length is equal to the tabu length set at the beginning;
end while

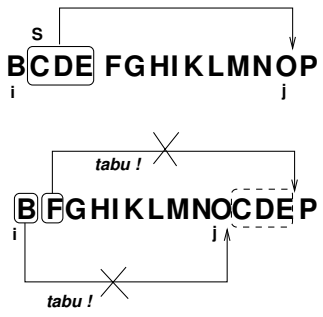


Figure 4.2: Example of tabu moves.

at the beginning of the search, only moves with string length equal to 3 are tried. The string length is decreased each time that no best move can be found, and reset equal to 3 each time a new best-so-far solution is found. Each time the string length has reached the minimum length (that is, 1), it is re-initialized to the maximum value (that is, 3).

Selection of the move to perform In TS it's important the notion of best-move-of-the-neighborhood because this is the move that is actually performed and leads to a new solution. Each time a new potential move is considered, it is evaluated and a decision is taken as if is a new best-move-of-the-neighborhood or not. The decision criterion differs in the case of tabu or non-tabu moves. If the move is tabu, then, in order to be promoted to be new best-move-of-the-neighborhood, the move should lead to a new best-so-far solution (according to the approximated move cost in TS-0 and according to the tour-length difference in TS-tsp). If the move is not tabu, then, it is enough that it leads to a new best solution with respect to the the current neighborhood. Not all moves are evaluated, in fact, tabu moves are evaluated only with a probability of the 30%, and non-tabu moves with a probability of the 80%. Also for this reason, sometimes no move is selected as new best-move-of-the-neighborhood. Once a new best-move-of-the-neighborhood is selected, it may be performed, but with a different mechanism in TS-0 and in TS-tsp. in TS-0, the move is performed, and the exact objective value of the new solution is computed. In TS-tsp, the move is performed only if it really leads to an improving solution with respect to the VRPSD objective function.

Tabu moves In general, there are several possibilities for defining what is a tabu move. The aim is always to avoid going back to a solution that has been visited in the last few iterations of the local search. In these TS, the concept of tabu move is simple. After a string S has been extracted from position i and re-inserted at position j, there are two moves that become tabu: those that would re-establish the relative position of the customer immediately preceding, respectively following S in the original tour. The two tabu moves are exemplified in fig. 4.2.3.

Tabu list Each time a new best-move-of-the-neighborhood is selected, it is put in the tabu list. In case there is no best-move-of-the-neighborhood, the

last selected best-move-of-the-neighborhood is put again in the tabu list. After, the oldest tabu move is removed from the list.

4.2.4 Ant Colony Optimization

The two algorithms of the ACO metaheuristic has been implemented by the IRIDIA node, from the author of the thesis. A detailed description of ACO algorithms can be found in Section 2.3.1, along with the pseudo-code in Algorithm 7.

The only difference with the ACS algorithm implemented for the preliminary experiments (for the details see Section 3.2.5) are:

`GenerateInitialSolution`(): the starting solution was generated by the randomized farthest insertion heuristic described in Section 4.1.2.

`LS`(): ACS-0 adopts the Or-Opt-0 local search, while ACS-tsp adopt the Or-Opt-tsp local search.

4.2.5 Evolutionary Computation

The two EC algorithms have been implemented by the ECRG node in Edinburgh. A detailed description of EC metaheuristic can be found in Section 2.3.2, along with the pseudo-code in Algorithm 8.

The only difference with the GA algorithm implemented for the preliminary experiments (for the details see Section 3.2.6) is the local search: EA-0 adopts Or-Opt-0 local search with approximated cost function while EA-tsp adopts the Or-Opt-tsp local search with TSP cost function.

4.3 Results of the successive experiments

The algorithms compared in the further experiments are: a randomized farthest insertion Random Restart Local Search that adopts the Or-Opt-0 local search (FR-0) that was used to set the minimum performance level to be achieved by the other algorithms,[1] a randomized farthest insertion Random Restart Local Search that adopts the Or-Opt-tsp local search (FR-tsp), two ILS algorithms (ILS-0 and ILS-tsp), whose implementations are described in Section 4.2.1, two SA algorithms (SA-0 and SA-tsp), whose implementations are described in Section 4.2.2, two TS algorithms (TS-0 and TS-tsp), whose implementations are described in Section 4.2.3, two ACO algorithms (ACS-0 and ACS-tsp), whose implementations are described in Section 4.2.4, two EC algorithms (EA-0 and EA-tsp), whose implementations are described in Section 4.2.5. Moreover, a state-of-the-art algorithms for solving TSP (RUN) was provided by the INTELLEKTIK node and a state-of-the-art algorithm to solve the CVRP (RUN.CVRP) was provided by the IDSIA node.

Instances

We generated a number of VRPSD instances that we classified into the same four classes described in Section 3.3.

[1] To fail in achieve better performance than FR-0 is to be considered a major failure.

Each algorithm was tested once on each instance for a time equal to the time needed by the algorithm FR-0 to complete 250, 500 or 1000 iterations, depending on the number of customers respectively 50, 100 or 200. So the execution time varied on an instance-by-instance basis.

Figure 4.3 illustrates the results aggregated over the four classes of instances. Figures 4.4, 4.5, 4.6, and 4.7 illustrate the results obtained by the algorithms under analysis on the classes of instances **rand_unif_n**, **rand_unif_r**, **rand_clust_n**, and **rand_clust_r**, in this order.

The statistical test used to verify if differences in solutions found by the algorithms are significant is the *Pairwise Wilcoxon rank sum* test with *p-values* adjustment method by Holm described in Section 3.3.1.

From the statistical analysis we observe the following:

1. when restricting the comparison to the -0 algorithms (the ones that only use the VRPSD objective function), only ILS-0 is better than FR-0;

2. when restricting the comparison to the -tsp algorithms, only ILS-tsp and EA-tsp are better than FR-tsp;

3. the -tsp version of the metaheuristics are statistically significantly better than the -0 version, we have verified this with a Wilcoxon signed rank test with continuity correction;

4. the TSP state-of-the-art algorithm is worse than the algorithm that has obtained the best results in our experiments;

It seems that totally ignoring the stochasticity of the demands and trying to solve the instances like TSP instead of VRPSD is not a good idea. The starting solution seems to contribute in a major manner to the final results of the algorithms, in fact as we said in Section 3.1.1, the approximation schemed used in Or-Opt local search might judge moves as "good", but they are worsening moves when evaluated with the exact objective function.

For this reason in Section 4.4.1 we proposed a new approximation of the VRPSD cost function.

4.4 Future works

Possible future works are researching different approximation of the cost function (we propose a new one in Section 4.4.1), or investigating new formulations of the VRP in order to determine common features (we propose a new one in Section 4.4.2).

4.4.1 Homogeneous Failure and Restocking

What follows is the result of fruitful discussions among the researchers of the *Metaheuristics Network*, in particular the spark has been given by Mauro Birattari of the IRIDIA node.

The experiments have shown that the evaluation of the cost function is particularly expensive. Among the metaheuristics implemented, those that adopted

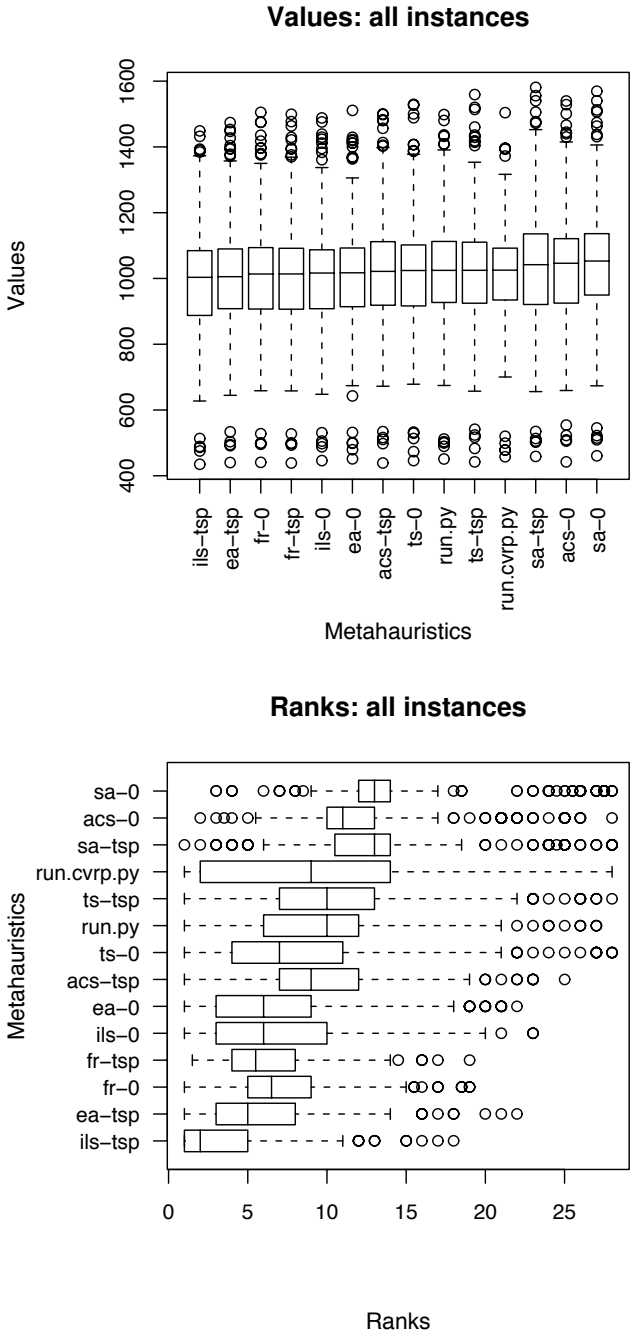


Figure 4.3: Results aggregated over the 4 classes of instances.

Table 4.1: *p-values* for the null hypothesis "The distributions of the solutions are the same" for aggregated classes of instances. The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

	ils-tsp	ea-tsp	fr-0	fr-tsp	ils-0	ea-0	acs-tsp	ts-0	run	ts-tsp	run.cvrp	sa-tsp	acs-0
ea-tsp	2e-09	—	—	—	—	—	—	—	—	—	—	—	—
fr-0	<2e-16	1e-03	—	—	—	—	—	—	—	—	—	—	—
fr-tsp	1e-13	1e-01	1e-08	—	—	—	—	—	—	—	—	—	—
ils-0	2e-11	1	2e-01	1	—	—	—	—	—	—	—	—	—
ea-0	8e-10	1	1	1	1	—	—	—	—	—	—	—	—
acs-tsp	<2e-16	<2e-16	4e-13	<2e-16	1e-10	5e-10	—	—	—	—	—	—	—
ts-0	<2e-16	1e-4	1e-01	1e-02	1e-02	4e-02	5e-02	—	—	—	—	—	—
run	<2e-16	1e-14	7e-11	1e-13	5e-10	4e-11	1	8e-04	—	—	—	—	—
ts-tsp	<2e-16	<2e-16	1e-10	3e-13	1e-09	2e-11	1	2e-04	1	—	—	—	—
run.cvrp	4e-09	3e-02	7e-01	2e-01	4e-02	1e-01	1	1	1	1	—	—	—
sa-tsp	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	1e-14	<2e-16	1e-08	1e-09	2e-04	—	—
acs-0	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	6e-09	2e-14	2e-04	1e-04	2e-02	6e-04	—
sa-0	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	<2e-16	1e-08	1e-01	5e-12

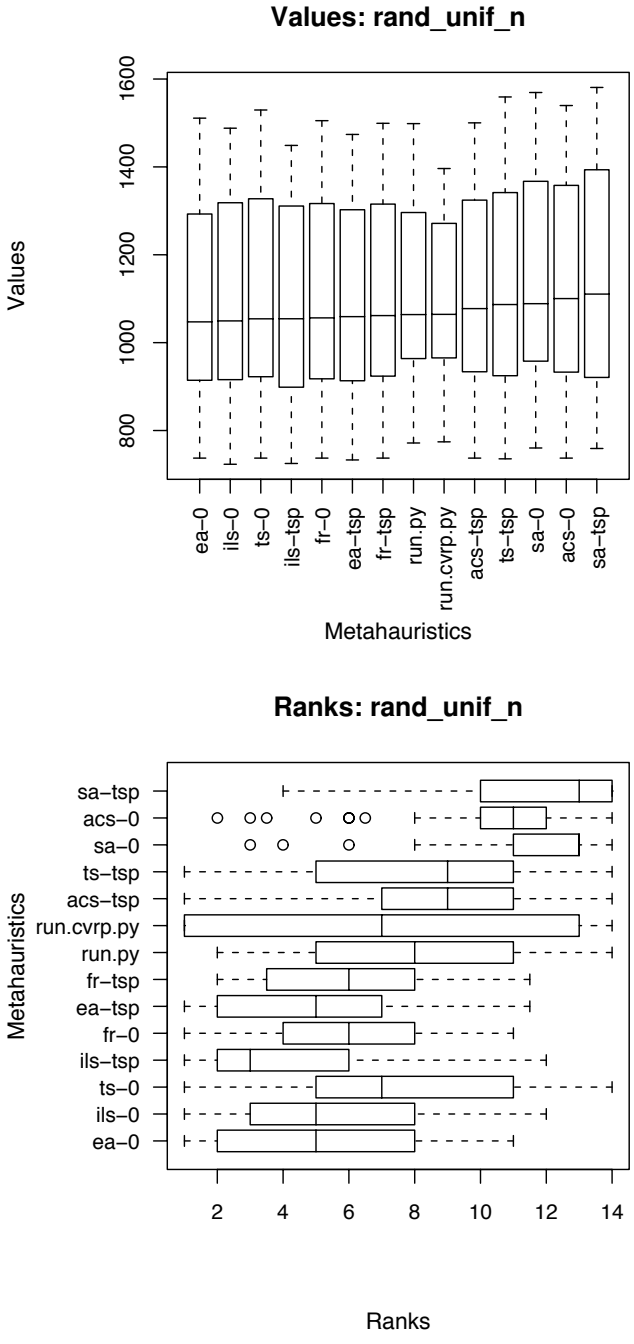


Figure 4.4: Results on the **rand_unif_n** class of instances.

Table 4.2: *p-values* for the null hypothesis "The distributions of the solutions are the same" for **rand_unif_n** class of instances. The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

	ea-0	ils-0	ts-0	ils-tsp	fr-0	ea-tsp	fr-tsp	run	run.cvrp	acs-tsp	ts-tsp	sa-0	acs-0
ils-0	1	—	—	—	—	—	—	—	—	—	—	—	—
ts-0	$1e-01$	$1e-01$	—	—	—	—	—	—	—	—	—	—	—
ils-tsp	1	7e-02	7e-04	—	—	—	—	—	—	—	—	—	—
fr-0	1	1	8e-01	2e-02	—	—	—	—	—	—	—	—	—
ea-tsp	1	1	6e-02	8e-01	8e-01	—	—	—	—	—	—	—	—
fr-tsp	1	1	4e-01	8e-02	4e-01	1	—	—	—	—	—	—	—
run	2e-02	1e-01	1	3e-05	7e-01	4e-02	3e-01	—	—	—	—	—	—
run.cvrp	1	1	1	1	1	1	1	1	—	—	—	—	—
acs-tsp	9e-03	2e-03	1	3e-05	3e-03	4e-04	1e-03	1	1	—	—	—	—
ts-tsp	4e-02	1e01	1	1e-04	1e-01	1e-02	4e-02	1	1	1	—	—	—
sa-0	9e-10	1e-11	4e-04	7e-11	7e-11	2e-10	2e-11	7e-04	6e-003	2e-06	7e-04	—	—
acs-0	6e-06	5e-06	4e-01	1e-08	3e-06	2e-06	3e-06	3e-01	3e-01	3e-01	1e-01	5e-02	—
sa-tsp	3e-09	3e-10	2e-03	1e-10	3e-09	2e-09	9e-09	8e-03	6e-02	3e-04	3e-03	1	1e-01

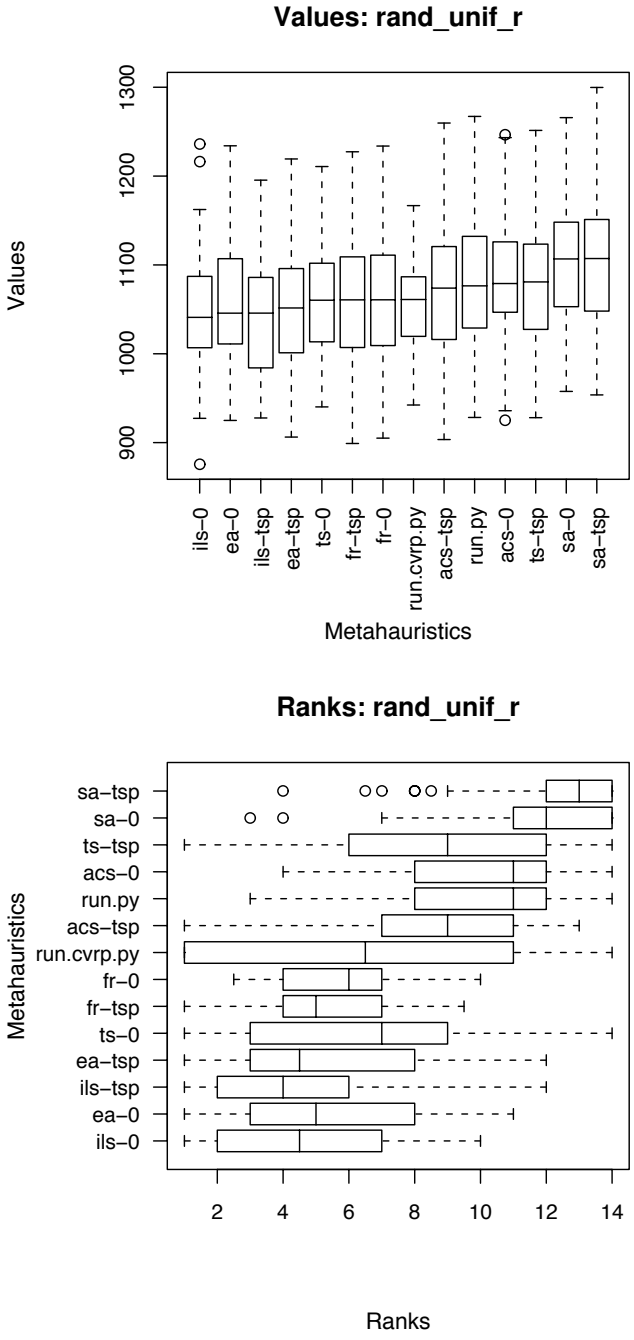


Figure 4.5: Results on the **rand_unif_r** class of instances.

Table 4.3: *p-values* for the null hypothesis "The distributions of the solutions are the same" for **rand_unif_r** class of instances. The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

	ils-0	ea-0	ils-tsp	ea-tsp	ts-0	fr-tsp	fr-0	run.cvrp	acs-tsp	run	acs-0	ts-tsp	sa-0
ea-0	1	—	—	—	—	—	—	—	—	—	—	—	—
ils-tsp	1	1	—	—	—	—	—	—	—	—	—	—	—
ea-tsp	1	1	1	—	—	—	—	—	—	—	—	—	—
ts-0	2e-01	1	1e-01	1	—	—	—	—	—	—	—	—	—
fr-tsp	7e-01	1	1	1	1	—	—	—	—	—	—	—	—
fr-0	3e-01	1	4e-01	1	1	2e-02	—	—	—	—	—	—	—
run.cvrp	1	1	1	1	1	1	1	—	—	—	—	—	—
acs-tsp	5e-04	8e-04	2e-04	2e-04	3e-01	3e-04	5e-04	1	—	—	—	—	—
run	9e-06	3e-06	6e-06	5e-06	4e-02	1e-06	2e-06	2e-01	1	—	—	—	—
acs-0	2e-07	3e-06	4e-07	2e-06	4e-05	1e-07	1e-07	4e-02	2e-01	1	—	—	—
ts-tsp	4e-04	6e-04	2e-04	3e-04	9e-02	3e-04	1e-03	4e-01	1	1	1	—	—
sa-0	7e-08	2e-07	7e-08	1e-07	2e-06	1e-07	1e-07	4e-04	3e-05	2e-03	2e-02	6e-04	—
sa-tsp	7e-08	1e-07	9e-08	2e-07	2e-07	9e-08	1e-07	2e-04	1e-05	6e-04	2e-02	6e-04	1

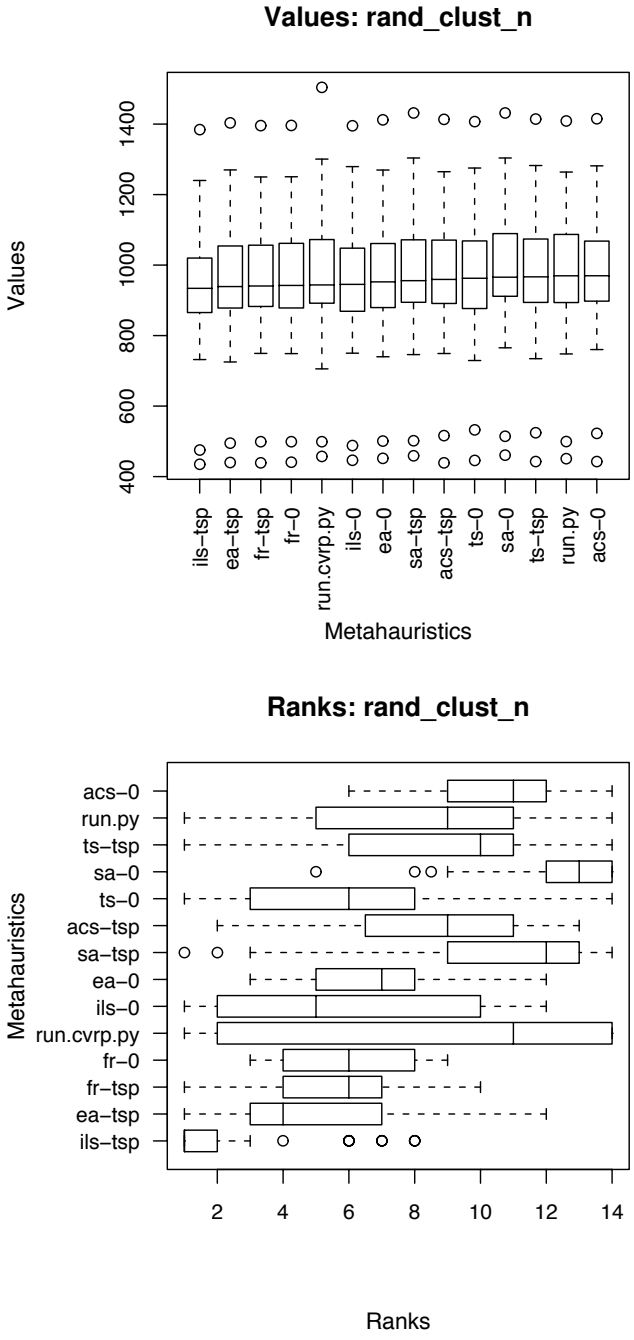


Figure 4.6: Results on the **rand_clust_n** class of instances.

Table 4.4: *p-values* for the null hypothesis "The distributions of the solutions are the same" for **rand_clust_n** class of instances. The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

	ils-tsp	ea-tsp	fr-tsp	fr-0	run.cvrp	ils-0	ea-0	sa-tsp	acs-tsp	ts-0	sa-0	ts-tsp	run
ea-tsp	2e-05	—	—	—	—	—	—	—	—	—	—	—	—
fr-tsp	1e-07	1	—	—	—	—	—	—	—	—	—	—	—
fr-0	4e-08	1	1	—	—	—	—	—	—	—	—	—	—
run.cvrp	8e-04	8e-01	1	1	—	—	—	—	—	—	—	—	—
ils-0	7e-08	1	1	1	1	—	—	—	—	—	—	—	—
ea-0	1e-08	2e-01	1	1	1	1	—	—	—	—	—	—	—
sa-tsp	1e-11	1e-06	1e-04	1e-05	1	9e-06	1e-04	—	—	—	—	—	—
acs-tsp	2e-10	3e-04	1e-02	1e-03	1	1e-01	1e-01	5e-02	—	—	—	—	—
ts-0	3e-07	1	1	1	1	1	1	2e-03	6e-01	—	—	—	—
sa-0	1e-11	1e-11	6e-07	6e-07	1	5e-12	2e-10	8e-02	4e-09	1e-08	—	—	—
ts-tsp	4e-10	9e-06	2e-03	1e-03	1	5e-01	2e-02	7e-01	1	4e-03	2e-05	—	—
run	7e-10	1e-03	3e-03	2e-02	1	1	4e-01	1	1	1e-01	4e-05	1	—
acs-0	1e-11	4e-10	5e-12	5e-12	1	9e-05	2e-06	1	4e-03	1e-05	5e-03	7e-01	1e-01

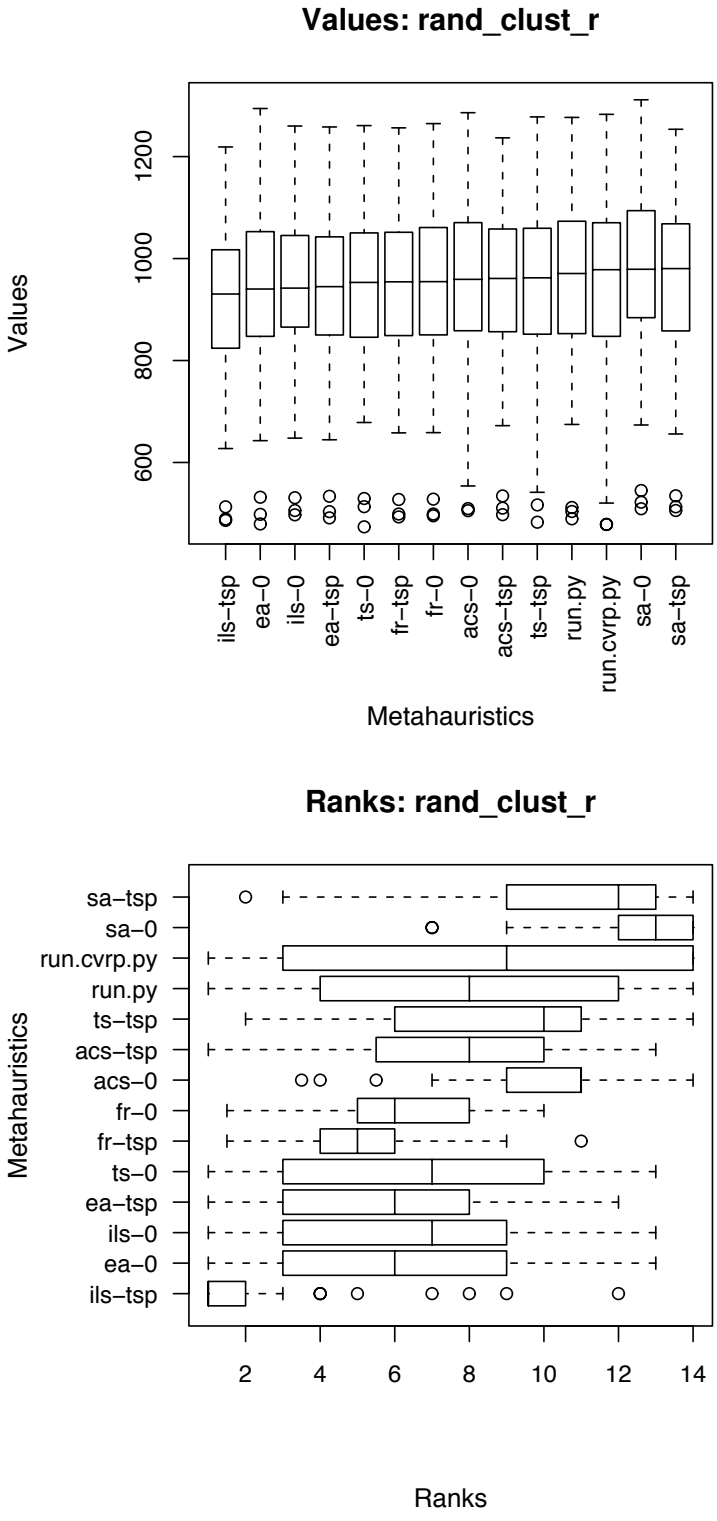


Figure 4.7: Results on the **rand_clust_r** class of instances.

Table 4.5: *p-values* for the null hypothesis "The distributions of the solutions are the same" for **rand_clust_r** class of instances. The significance level with which we reject the null hypothesis is 0.95. *p-values* smaller than 0.05 are sufficient to reject the null hypothesis in favor of the alternate hypothesis, while *p-values* greater than 0.05 do not allow us to reject the null hypothesis.

	ils-tsp	ea-0	ils-0	ea-tsp	ts-0	fr-tsp	fr-0	acs-0	acs-tsp	ts-tsp	run	run.cvrp	sa-0
ea-0	1e-07	—	—	—	—	—	—	—	—	—	—	—	—
ils-0	4e-08	1	—	—	—	—	—	—	—	—	—	—	—
ea-tsp	8e-07	1	1	—	—	—	—	—	—	—	—	—	—
ts-0	1e-07	1	1	4e-01	—	—	—	—	—	—	—	—	—
fr-tsp	8e-08	1	1	1	5e-01	—	—	—	—	—	—	—	—
fr-0	6e-09	1	1	2e-01	1	6e-05	—	—	—	—	—	—	—
acs-0	5e-12	3e-07	1e-04	1e-09	2e-04	1e-12	2e-07	—	—	—	—	—	—
acs-tsp	3e-10	5e-01	4e-01	2e-02	1	4e-04	3e-01	1e-03	—	—	—	—	—
ts-tsp	1e-10	1e-02	5e-02	1e-04	4e-01	3e-04	4e-02	5e-01	1	—	—	—	—
run	2e-09	5e-01	4e-01	1e-02	7e-01	2e-02	5e-01	4e-01	1	1	—	—	—
run.cvrp	1e-08	4e-01	2e-01	5e-02	4e-01	1e-01	4e-01	1	1	1	1	—	—
sa-0	3e-13	1e-11	3e-13	1e-12	1e-08	2e-12	3e-12	1e-05	1e-11	1e-07	4e-07	1	—
sa-tsp	6e-13	1e-04	1e-05	2e-07	2e-03	9e-08	3e-06	1	8e-03	5e-01	6e-01	1	3e-03

an approximation of the cost function, rather than the full evaluation, obtained significantly better results.

All implemented metaheuristics that have resorted to an approximation of the VRPSD cost function where TSP-based: the cost of an *a priori* tour, rather than being evaluated exactly, was approximated by its cost in the TSP sense. The rationale behind such approximation is related to the apparent similarities between VRPSD and TSP (as illustrated in Section 3.4). It could be further noticed that the approximation becomes exact in the degenerate case in which the truck capacity exceeds the sum of the maximum demand of all customers.

Another problem that presents similarities with VRPSD is the CVRP. Indeed, VRPSD degenerates into CVRP if the spread in the demand of the customers is null—in other words, if the demand is deterministic. None of the implementations proposed within the Metaheuristics networks exploited this similarity. The reason is that it seems quite complicated to predict with sufficient accuracy the load of the truck along the *a priori* tour in order to define locations at which a re-stocking will be needed.

The aim of this section is to describe a possible way to tackle this problem: The homogeneous failure and restocking approximation of the VRPSD cost function.

The homogeneity hypothesis

The hypothesis underlying the discussion we propose is the following:

> Hp_{hfr}: The probability of failure and restocking is uniform along the *a priori* tour.[2]

This hypothesis can be seen as a way of representing the total ignorance on the positions along the *a priori* tour at which failures and restockings will occur.

Under hypothesis Hp_{hfr}, and assuming that the probabilities of failure and restocking are known, say f and r respectively, the cost of a given *a priori* tour can be expressed as the sum of individual contributions, one per each arc composing the *a priori* tour itself. Namely, say that the arc $\langle i, j\rangle$ is included in the *a priori* tour. Its contribution to the cost is:

$$C_{i,j} = (1 - f - r)d_{i,j} + r(d_{i,0} + d_{0,j}) + f(d_{i,j} + d_{j,0} + d_{0,j}) \qquad (4.2)$$

where $d_{i,j}$ is the distance between i and j, $d_{i,0}$ is the distance between i and the depot, and $d_{j,0} = D_{0,j}$ is the distance between node j and the depot. See Figure 4.4.1 for a graphical representation of the quantities involved in Equation 4.2.

Possible search schemes

In this section we discuss possible search schemes based on the Hp_{hfr} hypothesis. To this end, let us first notice that the restocking and failure probabilities are clearly solution-dependent: In the general case, we should expect that two different *a priori* tours, say T_1 and T_2, are characterized by two different sets

[2]Some *border effect* exists. For instance, for what concerns the first and last link in the tour, i.e., depot-first_customer and last_customer-depot, the probability of failure and restocking are both null. This border effect will be considered in the following.

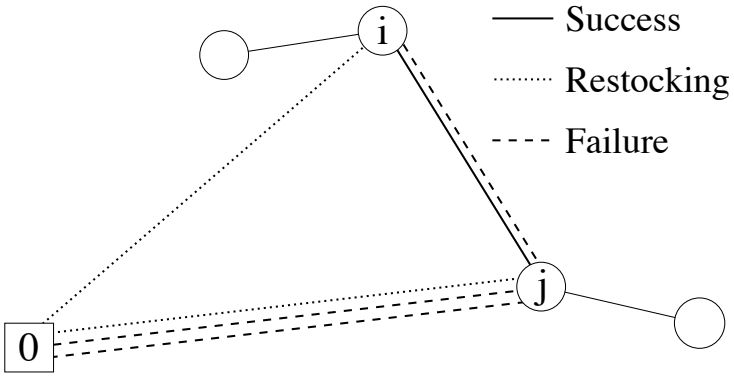


Figure 4.8: Inclusion on link $\langle i, j \rangle$ in the solution: The cost of including link $\langle i, j \rangle$ is a weighted average of the distance $\langle i, j \rangle$ (Success), the distance $\langle i, 0, j \rangle$ (Restocking), and the distance $\langle i, j, 0, j \rangle$ (Failure).

of restocking and failure probabilities, say r_1 and f_1 for the former, and r_2 and f_2 for the latter.

Moreover, it should be clear that if the restocking and failure probabilities associated with the optimal solution where known in advance, the optimal solution itself could be found by solving a TSP on a graph obtained from the original one where the cost of the generic arc $\langle i, j \rangle$ is set to C_i, j as defined in Equation 4.2. On the other hand, when a solution is given, possibly the optimal one, the corresponding restocking and failure probabilities can be obtained either analytically or empirically through Monte Carlo sampling.

In the typical case, however, neither the optimal solution is known, nor the associated restocking and failure probabilities. To tackle such "chicken and egg" situations in which if one of two elements where known, the other could be easily obtained, and vice-versa, but none is available, the typical approach is based on an iterative procedure. In the context of our VRPSD, an iterative solution scheme would be implemented as follows:

1. set current solution to some initial solution,
2. evaluate failure and restocking probabilities for current solution,
3. solve problem for current probabilities, and update current solution,
4. stop upon convergence; otherwise, go back to 2.

We expect this approach to be effective if the uncertainty is high on the positions along an *a priori* tour at which restocking and failures will occur. In such a case, the Hp_{hfr} hypothesis is not restrictive and indeed models correctly the information available.

On the contrary, we do not expect this approach to be extremely effective in the case in which the uncertainty is relatively low as for example if the capacity of the truck is very large compared to the average request of the customers, and

if the spread of the request is relatively small. In such a case, since the number of customers served with one single load of the truck is high, the actual realization of the sum of the requests of the customers served between two consecutive stops at the depot will tend to converge to its expected value (see law of large numbers) washing out, *de facto*, most of the uncertainty.

In such a case, the Hp_{hfr} hypothesis is possibly too restrictive and the solution schemes proposed in this document disregards some useful information that could possibly be profitably employed.

4.4.2 Possible Extensions of the Problem

One of the aims of the *Metaheuristics Network* is to analyze the behavior of metaheuristics on a variety of CO problems. Therefore we investigated other possible formulations of Vehicle Routing, mainly with deterministic demand, in order to distinguish common features. On one hand, real-world problems have much more constraints than the one present in our model, on the other hand, we must keep the model simple since we have limited computational power and limited time. The experience in real-world scenarios provided by the industrial partner Ant Optima was illuminating. We propose, therefore, the following formulation that tries to approximate several over-constrained real-world scenarios still maintaining a certain generality.

A non-homogeneous fleet of vehicles has to be routed to deliver or pickup goods. Vehicles can differ for fixed costs, variable cost and capacity. Accessibility constraints restrict the use of vehicles to specific customers and to specific paths while time constraints impose that a vehicle visit a customer not later than its due date. Time constraints can be multiple, that is, composed by several intervals. The number of vehicles to use is biased by a maximum make-span per vehicle which could be expressed in time or in number of customers to visit and by the objective function to minimize. The objective function is the weighted sum of fixed costs per vehicle, cost per vehicle according to its use and total traveled length. However, the definition of weights and fixed costs should be carefully analyzed because the property to determine the number of vehicles depends from their interaction.

Resuming, we can identify the following hard and soft constraints.

Hard Constraints:

- maximum make-span per vehicle
- accessibility
- Non homogeneous fleet (different fixed cost, different cost per kilometer, different capacity)
- multiple intervals time windows.

Soft Constraints:

- fixed cost per vehicle
- variable cost per vehicle

- expected length

The problem thus defined is at the core of several real cases. Stochastic demands could be included and the work done for the VRPSD simply reused to optimize each single route. Nevertheless, we conjectured that time windows, meant as hard constraints, would require to consider only worst cases behavior, that is, the maximum possible demand for each customers. Indeed, the mixed strategy does not guarantee against failures, and failures drastically modify timetable making it easily infeasible in the case of strict time windows rules.

Conclusions

The main goal of this DEA thesis is a fair and meaningful comparison of different metaheuristics applied to the Vehicle Routing Problem with Stochastic Demands, whose formulation was given in Section 1.2.

The author of the thesis participated to the research conducted at the IRIDIA node of the *Metaheuristics Network*, a Research Training Network presented in the Introduction. The experimental studies in the *Metaheuristics Network* are designed in the first place to compare the implementations of the metaheuristics contributed by the labs participating in the Network, in order to understand which metaheuristics perform well on which problems, and only as a secondary goal to compare the developed metaheuristics against externally available algorithms to show that they perform better. In order to avoid the pitfall we have illustrated in Section 3.1, the *Metaheuristics Network* has defined a strictly controlled, machine independent, experimental methodology which allows a fair and meaningful comparison of experimental results in order to compare different metaheuristics under the same experimental conditions.

In a first phase we compared, over four classes of instances, a basic Random Restart Local Search that was used to set the minimum performance level to be achieved by the other algorithms, Iterated Local Search, Simulated Annealing, Tabu Search, Ant Colony Optimization and Evolutionary Computation (the five basic implementations of the metaheuristics normally implemented in the *Metaheuristics Network*), and also an algorithm called Flim-Flam, whose purpose was to see how far we could go by using strategies for solving Vehicle Routing Problem with Stochastic Demands instances as Traveling Salesman Problem ones. These comparisons were illustrated in Section 3.3. Flim-Flam resulted the best performing algorithm, showing that, at least on the instances that we considered, a TSP approximation of the cost function is particularly effective. In a second phase we studied more in detail how to exploit the similarities of the Vehicle Routing Problem with Stochastic Demands with the Traveling Salesman Problem, therefore each of the five metaheuristics (ILS, SA, TS, ACO, EC) and a Randomized Farthest Insertion Random Restart Local Search were implemented using two different approximated objective functions (a VRPSD proxy objective function and a TSP objective function). Furthermore, state-of-the-art algorithms for the TSP and for the CVRP were used for comparison. The same four classes of instances used in the first phase were adopted. These comparisons were illustrated in Section 4.3. From this second phase we concluded that the TSP approximation of the cost function is indeed effective compared to the VRPSD proxy objective function. The general conclusions that we can draw after all the comparisons are:

1. The difficulty of the instances of the problem, with respect to the aggregate results of the metaheuristics, varies, sometimes even significantly, among instances of different classes, and, although in a minor measure, among instances that belong to the same class. This behavior was expected, and reflects the situation of the real world, where some specific problems (for example, a few clients located far away from all the others, or some clients having a very high variability in the demands) can be much more difficult to solve than the average ones.

2. The relative performance of the algorithms under analysis is similar across the different classes of instances: ILS is always the best performing metaheuristic followed by EC and then by ACO, TS and SA. The metaheuristics ILS, EC obtain significantly better results than the Randomized Farthest Insertion Random Restart Local Search, while ACO, TS and SA fail in achieving the minimum required performances.

3. At least on the instances considered in this research, a TSP approximation of the cost function is particularly effective. The performance of the metaheuristics that adopt the TSP objective function are significantly better than the metaheuristics that adopt the VRPSD proxy objective function.

It is interesting to note that the two state-of-the-art algorithm for TSP and CVRP obtained worse performances than some of the Metaheuristics Network's algorithms; this is a point which encourages the development of VRPSD-specific algorithms and suggests that trying to treat the VRPSD as a TSP is not a good idea, but that we have to make more efforts in trying to exploit more problem specific knowledge and the specific solution landscape structure.

For what concern the bad performance obtained by the ACO metaheuristics we want to point out the fact that the implemented ACS was quite a "basic" one. As future work, in addition to what presented in Section 4.4 it could be interesting to implement a new version of ACS using the Hyper-Cube Framework for ACO recently developed by Blum and Dorigo [15].

Bibliography

[1] E. H. L. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven. Simulated annealing. In *Local Search in Combinatorial Optimization*, pages 91–120. John Wiley & Sons, Chichester, UK, 1997.

[2] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, New York, NY, 1996.

[3] T. Bäck, D. Fogel, and M. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing, Bristol, UK, 1997.

[4] Y. Bar-Yam. *Dynamics of Complex Systems*. Westview Press, Boulder, CO, 1997.

[5] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

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

[7] D. J. Bertsimas. A vehicle routing problem with stochastic demand. *Operations Research*, 40(3):574–585, 1992.

[8] D. J. Bertsimas, P. Chervi, and M. Peterson. Computational approaches to stochastic vehicle routing problems. *Transportation Science*, 29(4):342–352, 1995.

[9] D. J. Bertsimas, P. Jaillet, and A. Odoni. A priori optimisation. *Operations Research*, 38:1019–1033, 1990.

[10] D. J. Bertsimas and D. Simchi-Levi. A new generation of vehicle routing research: robust algorithms, addressing uncertainty. *Operations Research*, 44(2):216–304, 1996.

[11] L. Bianchi, M. Birattari, M. Chiarandini, M. Manfrin, M. Mastrolilli, L. Paquete, O. Rossi-Doria, and T. Schiavinotto. Metaheuristics for the vehicle routing problem with stochastic demands. Technical Report IDSIA-06-04, IDSIA, Manno, Switzerland, 2004.

[12] L. Bianchi, L. M. Gambardella, and M. Dorigo. An ant colony optimization approach to the probabilistic traveling salesman problem. In *Parallel Problem Solving from Nature - PPSN VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 883–892, Berlin, Germany, 2002. Springer Verlag.

[13] L. Bianchi, L. M. Gambardella, and M. Dorigo. Solving the homogeneous probabilistic traveling salesman problem by the ACO metaheuristic. In *Proceedings of ANTS 2002 – Third International Workshop on Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 176–187, Berlin, Germany, 2002. Springer Verlag.

[14] L. Bianchi, M. Mastrolilli, M. Birattari, M. Manfrin, M. Chiarandini, L. Paquete, and O. Rossi-Doria. Report for task 5: Research on the vehicle routing problem with stocastic demand. Internal document of the Metaheuristics Network, 2004.

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

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

[17] J. Bramel and D. Simchi-Levi. *The Logic of Logistics*. Springer, Berlin, Germany, 1997.

[18] P. Calegari, G. Coray, A. Hertz, D. Kobler, and P. Kuonen. A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics*, 5:145–158, 1999.

[19] D. J. Cavicchio. *Adaptive search using simulated evolution*. PhD thesis, University of Michigan, Ann Arbor, MI, 1970.

[20] V. Cerný. A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[21] P. Chardaire, J. L. Lutton, and A. Sutter. Thermostatical persistency: a powerful improving concept for simulated annealing algorithms. *European Journal of Operational Research*, 86:565–579, 1995.

[22] W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, NY, 1999.

[23] T. G. Crainic and G. Laporte. Planning models for freight transportation. *European Journal of Operational Research*, 97:409–438, 1997.

[24] K. A. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, 1975.

[25] J.-L. Deneubourg, S. Aron, S. Goss, and J.-M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behaviour*, 3: 159–168, 1990.

[26] R. L. Devaney. *An introduction to chaotic dynamical systems*. Addison-Wesley, Reading, MA, second edition, 1989.

[27] M. Dorigo. *Ottimizzazione, apprendimento automatico, ed algoritmi basati su metafora naturale*. PhD thesis, Politecnico di Milano, Milan, Italy, 1992.

[28] M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, UK, 1999.

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

[30] M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.

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

[32] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications and advances. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 251–285. Kluwer Academic Publishers, Norwell, MA, 2002.

[33] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[34] A. E. Eiben and Z. Ruttkay. Constraint satisfaction problems. In T. Bäck, D. Fogel, and M. Michalewicz, editors, *Handbook of Evolutionary Computation*. Institute of Physics Publishing, Bristol, UK, 1997.

[35] W. Feller. *An introduction to Probability Theory and its Applications*. John Wiley & Sons, New York, NY, 1968.

[36] M. Fisher. Vehicle routing. In O. M. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Routing*, pages 1–33. Elsevier, Amsterdam, The Netherlands, 1995.

[37] M. Fleischer. Simulated annealing: past, present and future. In C. Alexopoulos, K. Kang, W. R. Lilegdon, and G. Goldsam, editors, *Proceedings of the 1995 Winter Simulation Conference*, pages 155–161, 1995.

[38] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, Jan 1994.

[39] L. J. Fogel. Toward inductive inference automata. In *Proceedings of the International Federation for Information Processing Congress*, pages 395–399, Munich, Germany, 1962.

[40] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, NY, 1966.

[41] M. R. Garey and D. S. Johnson. *Computers and Intractability / A Guide to the Theory of $\mathcal{NP}$-Completeness*. W.H. Freeman & Company, San Francisco, CA, 1979.

[42] M. Gendreau, G. Laporte, and R. Séguin. An exact algorithm for the vehicle routing problem with stochastic demands and customers. *Transportation Sciences*, 29(2):143–155, 1995.

[43] M. Gendreau, G. Laporte, and R. Séguin. A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. *Operations Research*, 44(3), 1996.

[44] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[45] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13:533–549, 1986.

[46] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, second edition, 1997.

[47] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In J. J. Grefenstette, editor, *Genetic Algorithms and their Applications*, pages 41–49. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[48] B. L. Golden and A. A. Assad, editors. *Vehicle Routing: Methods and Studies*. Elsevier, Amsterdam, The Netherlands, 1988.

[49] M. Haimovitch and A. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10:527–542, 1985.

[50] A. Hertz and D. Kobler. A framework for the description of evolutionary algorithms. *European Journal of Operational Research*, 126:1–12, 2000.

[51] J. H. Holland. *Adaptation in natural artificial systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[52] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[53] L. Ingber. Adaptive simulated annealing (asa): Lessons learned. *Control and Cybernetics - Special Issue on Simulated Annealing Applied to Combinatorial Optimization*, 25(1):33–54, 1996.

[54] P. Jaillet. *Probabilistic Traveling Salesman Problems*. PhD thesis, MIT, Cambridge, MA, 1985.

[55] P. Jaillet. A priori solution of a travelling salesman problem in which a random subset of the customers are visited. *Operations Research*, 36(6): 929–936, 1988.

[56] P. Jaillet and A. Odoni. The probabilistic vehicle routing problems. In B. L. Golden and A. A. Assad, editors, *Vehicle Routing: Methods and Studies*. Elsevier, Amsterdam, The Netherlands, 1988.

[57] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Operations Research*, 37(6):865–892, 1989.

[58] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, New York, NY, 1997.

[59] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[60] G. Laporte and F. Louveaux. The integer l-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, 33: 133–142, 1993.

[61] H. R. Lourenço, O. Martin, and T. Stützle. A beginner's introduction to iterated local search. In *Proceedings of MIC'2001 - Meta-heuristics International Conference*, volume 1, Porto, Portugal, 2001.

[62] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.

[63] S. W. Mahfoud. *Niching Methods for Genetic Algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1995.

[64] M. Manfrin, M. Birattari, C. Blum, M. Sampels, O. Rossi-Doria, M. Chiarandini, L. Paquete, T. Schiavinotto, L. Bianchi, M. Mastrolilli, T. Bousonville, H. Juillé, and D. Huber. Report for task 7: Experimental comparison. Internal document of the Metaheuristics Network, 2004.

[65] M. Manfrin, M. Birattari, C. Blum, M. Sampels, O. Rossi-Doria, M. Chiarandini, L. Pauqete, T. Schiavinotto, L. Bianchi, and M. Mastrolilli. Report for task 8: Final analysis. Internal document of the Metaheuristics Network, 2004.

[66] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, 1991.

[67] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[68] Z. Michlewicz and M. Michalewicz. Evolutionary computation techniques and their applications. In *Proceedings of the IEEE International Conference on Intelligent Processing Systems*, pages 14–24, Piscataway, NJ, 1997. IEEE Publications.

[69] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, CA, 1989.

[70] P. Moscato. Memetic algorithms: A short introduction. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 219–234. McGraw-Hill, London, UK, 1999.

[71] H. Mühlenbein and G. Paaß. From recombination of genes to estimation of distributions. In H.-M. Voight, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, volume 1411 of *Lecture Notes in Computer Science*, pages 178–187, Berlin, Germany, 1996. Springer Verlag.

[72] H. Mühlenbein and H.-M. Voigt. Gene pool recombination in genetic algorithms. In I. H. Osman and J. P. Kelly, editors, *Proceedings of the Metaheuristics Conference*, Norwell, MA, 1995. Kluwer Academic Publishers.

[73] G. I. Nemhauser and A. L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, NY, 1988.

[74] I. Or. *Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking*. PhD thesis, Nortwestern University, Evanston, IL, 1976.

[75] I. Osman and G. Laporte. Metaheuristics: A Bibliography. *Annals of Operations Research*, 63:513–623, 1996.

[76] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization - Algorithms and Complexity*. Dover Publications, Inc., New York, NY, 1982.

[77] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, second edition, 1993.

[78] N. J. Radcliffe. Forma analysis and random respectful recombination. In *Proceedings of the Fourth International Conference on Genetic Algorithms, ICGA 1991*, pages 222–229, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[79] I. Rechenberg. *Evolution strategy: Optimization of technical systems by means of biological evolution*. Fromman-Holzboog, Stuttgart, Germany, 1973.

[80] D. Roberts and E. Hadjiconstantinou. A computational approach to the vehicle routing problem with stochastic demands. In P. Borne, M. Ksouri, and A. El Kamel, editors, *Computational Engineering in Systems Applications*, pages 139–144. IEEE Publications, Piscataway, NJ, 1998.

[81] N. Secomandi. *Exact and heuristic dynamic programming algorithms for the vehicle routing problem with stochastic demands*. PhD thesis, University of Houston, Houston, TX, 1998.

[82] N. Secomandi. A rollout policy for the vehicle routing problem with stochastic demands. *Operations Research*, 49(5):796–802, 2001.

[83] W. M. Spears, K. A. De Jong, T. Bäck, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In P. B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning (ECML-93)*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 442–459, Berlin, Germany, 1993. Springer Verlag.

[84] P. F. Stadler. Towards a theory of landscapes. In R. Lopéz-Peña, R. Capovilla, and R. García-Pelayo, editors, *Complex Systems and Binary Networks*, volume 461 of *Lecture Notes in Physics*, pages 77–163. Springer Verlag, Berlin, Germany, 1995.

[85] P. F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20:1–45, 1996.

[86] T. Stützle. *Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms and New Applications*. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany, 1998. Published in 1999 - Infix, Sankt Augustin, Germany - volume 220 of DISKI.

[87] T. Stützle and M. den Besten. Guidelines for the production of software. Internal document of the Metaheuristics Network, 2000.

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

[89] T. Stützle and H. Hoos. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. *Future Generation Computer System*, 16(8):889–914, 2000.

[90] T. Stützle and H. Hoos. Analyzing the run-time behaviour of iterated local search for the tsp. In P. Hansen and C. Ribeiro, editors, *Essays and Surveys on Metaheuristics*, pages 589–612. Kluwer Academic Publishers, Boston, MA, 2002.

[91] G. Syswerda. Simulated crossover in genetic algorithms. In *Proceedings of the second workshop on Foundations of Genetic Algorithms*, pages 239–255, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[92] S. Voss, S. Martello, I. Osman, and C. Roucairol. *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Boston, MA, 1999.

[93] W. Yang, K. Mathur, and R. H. Ballou. Stochastic vehicle routing problem with restocking. *Transportation Science*, 34(1):99–112, 2000.

[94] M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research*, 131:373–395, 2004.