UNIVERSITE LIBRE DE BRUXELLES, UNIVERSITE D'EUROPE

ULB

Faculté des Sciences appliquées

Année académique 2005-2006

ANT COLONY OPTIMIZATION AND LOCAL SEARCH FOR THE PROBABILISTIC TRAVELING SALESMAN PROBLEM: A CASE STUDY IN STOCHASTIC COMBINATORIAL OPTIMIZATION

Directeur de Mémoire:
Prof. Marco Dorigo

Codirecteur de Mémoire:
Prof. Luca Maria Gambardella

Mémoire de fin d'études présenté par Leonora Bianchi en vue de l'obtention du titre de Docteur en Sciences Appliquées

Ant Colony Optimization and Local Search for the Probabilistic Traveling Salesman Problem:

A Case Study in Stochastic Combinatorial Optimization

Leonora Bianchi

Université Libre de Bruxelles

Summary

In this thesis we focus on Stochastic Combinatorial Optimization Problems (SCOPs), a wide class of combinatorial optimization problems under uncertainty, where part of the information about the problem data is unknown at the planning stage, but some knowledge about its probability distribution is assumed.

Optimization problems under uncertainty are complex and difficult, and often classical algorithmic approaches based on mathematical and dynamic programming are able to solve only very small problem instances. For this reason, in recent years metaheuristic algorithms such as Ant Colony Optimization, Evolutionary Computation, Simulated Annealing, Tabu Search and others, are emerging as successful alternatives to classical approaches.

In this thesis, metaheuristics that have been applied so far to SCOPs are introduced and the related literature is thoroughly reviewed. In particular, two properties of metaheuristics emerge from the survey: they are a valid alternative to exact classical methods for addressing real-sized SCOPs, and they are flexible, since they can be quite easily adapted to solve different SCOPs formulations, both static and dynamic. On the base of the current literature, we identify the following as the key open issues in solving SCOPs via metaheuristics: (1) the design and integration of ad hoc, fast and effective objective function approximations inside the optimization algorithm; (2) the estimation of the objective function by sampling when no closed-form expression for the objective function is available, and the study of methods to reduce the time complexity and noise inherent to this type of estimation; (3) the characterization of the efficiency of metaheuristic variants with respect to different levels of stochasticity in the problem instances.

We investigate the above issues by focusing in particular on a SCOP belonging to the class of vehicle routing problems: the Probabilistic Traveling Salesman Problem (PTSP). For the PTSP, we consider the Ant Colony Optimization metaheuristic and we design efficient local search algorithms that can enhance its performance. We obtain state-of-the-art algorithms, but we show that they are effective only for instances above a certain level of stochasticity, otherwise it is more convenient to solve the problem as if it were deterministic. The algorithmic variants based on an estimation of the objective function by sampling obtain worse results, but qualitatively have the same behavior of the algorithms based on the exact objective function, with respect to the level of stochasticity. Moreover, we show that the performance of algorithmic variants based on ad hoc approximations is strongly correlated with the absolute error of the approximation, and that the effect on local search of ad hoc approximations can be very degrading.

Finally, we briefly address another SCOP belonging to the class of vehicle routing problems: the Vehicle Routing Problem with Stochastic Demands (VRPSD). For this problem, we have implemented and tested several metaheuristics, and we have studied the impact of integrating in them different ad hoc approximations.

Acknowledgments

The research work of this thesis has been mainly done at IDSIA, the Dalle Molle Institute for Artificial Intelligence in Lugano, Switzerland. I express my sincere thanks to all people that have been at IDSIA since I arrived there in the year 2000, because of the friendly and special working environment they created.

An important part of this thesis is rooted in one month spent at IRIDIA, Université Libre de Bruxelles, Brussels, Belgium. I must thank all the people working there, and particularly Joshua Knowles, because, without already being involved in my subject of research, he was very open and we could have a profitable exchange of ideas, that resulted in my first journal paper. From IRIDIA I also thank the secretary, Muriel Decreton, for her help in the bureaucratic formalities that she carried out for me while I was in Switzerland.

I also thank all the people involved in the "Metaheuristics Network", particularly those with whom I worked for the research about the stochastic vehicle routing problem: Mauro Birattari, Marco Chiarandini, Max Manfrin, Monaldo Mastrolilli, my husband Fabrizio Oliverio, Luis Paquete, Olivia Rossi-Doria, and Tommaso Schiavinotto. From each of them I learnt something very useful for my research. I especially thank Mauro Birattari and Marco Chiarandini for their support with statistical analysis of results.

I acknowledge financial support by two sources: the Swiss National Science Foundation project titled "On-line fleet management", grant 16R10FM; and the "Metaheuristics Network", a Research and Training Network founded by the Improving Human Potential Programme of the Commission of the European Communities, grant HPRN-CT-1999-00106.

I am particularly grateful to Luca Maria Gambardella for having supported and guided my research since the beginning, and to Marco Dorigo for his careful supervision of my work.

Contents

List of Algorithms

Original contributions and outline

The contributions of this thesis stay at the intersection of two very active streams of research inside the field of combinatorial optimization: Stochastic Combinatorial Optimization Problems (SCOPs) on one side, and metaheuristics on the other side. The general logical structure of the thesis, which is schematized by Figure 1, is the following. Part I addresses the field of solving SCOPs via metaheuristics; Part II focuses on solving one particular SCOP, the Probabilistic Traveling Salesman Problem (PTSP), via one particular metaheuristic, Ant Colony Optimization (ACO); and Appendix B considers solving another particular SCOP, the Vehicle Routing Problem with Stochastic Demands (VRPSD), via several metaheuristics. For each of these conceptual blocks, we describe here the main original contributions, with pointers to the corresponding scientific publications where part of the ideas have appeared or will appear.

Clear definition of the scope of SCOPs There is an increasing interest of the operations research community in addressing optimization problems that include in their mathematical formulation uncertain, stochastic, and dynamic information. The multitude of model formulations in the literature makes it difficult to compare different solution approaches and identifying promising directions of research. Chapter 1 proposes a classification of the modeling approaches to uncertainty according to dynamicity and type of description of uncertain data, and precisely defines the scope of SCOPs. Chapter 2 recalls the main formal definitions of both static and dynamic SCOPs from the literature, by providing links to the corresponding algorithmic domains, with the aim of giving a clear view of the intersection between classical approaches and new ones based on metaheuristics.

Survey of metaheuristics applications to SCOPs Chapter 3 aims at putting under a unifying view the several applications of metaheuristics to SCOPs, by filling a gap in the literature, where a number of surveys and books about solving SCOPs via classical techniques exist, but none about using metaheuristics, despite the research literature is already quite rich. This chapter also selects and discusses some open issues that emerge from the survey. In particular, the issue of using approximations of the objective function inside optimization algorithms will be further investigated in Part II. The content of Part I (from Chapter 1 to Chapter

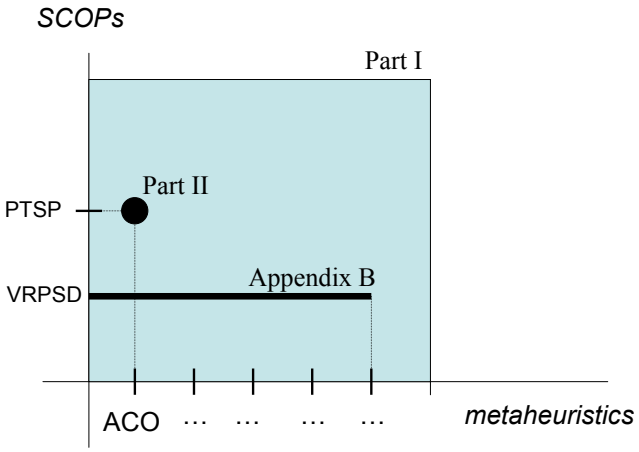


Figure 1: General outline of this thesis (acronyms are explained in the text).

3), is the source of an article ([37]) which is in preparation for submission to an international journal .

A benchmark for the PTSP The PTSP has many features that makes it useful as a test problem for algorithms addressing SCOPs, and it is likely it will be considered again in the literature. Thus, we think that the creation of a benchmark of instances for the PTSP will be a useful instrument for future research. Chapter 4, besides introducing the PTSP as a particular case study of the SCOPs class, describes a benchmark of 432 PTSP instances that we have set up for the experimental evaluation of algorithms for the PTSP. The benchmark has been carefully designed, in particular to allow the analysis of the behavior of optimization algorithms for different levels of stochasticity. In order to facilitate future comparisons with our results, we have also used a known lower bound from the literature to evaluate the lower bound of the optimal solution values for the instances of the PTSP benchmark.

ACO for the PTSP Chapter 5 develops several ACO algorithms for the PTSP, and focuses on two aspects. First, the dependence of ACO performance on PTSP instance characteristics. Second, the impact of different approximations of the objective function on the performance of ACO. Preliminary experiments about the first ACO algorithm of this chapter, pACS, have been published in the proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN-VII) [38] and in the proceedings of the 3rd International Workshop on Ant Algorithms (ANTS 2002) [39].

Powerful local search algorithms for the PTSP Chapter 6 explores, in the specific context of the PTSP, a range of possibilities among which to choose when designing a local search algorithm for a SCOP that has a computationally ex-

pensive objective function. The most promising alternatives are: using an approximated cost for the local search operators, and finding cost expressions that can be computed efficiently and exactly by exploiting some recursive mechanism. In Chapter 6 different cost approximations are proposed and analyzed, and, for the 2-p-opt and 1-shift local search operators, efficient and exact cost expressions are derived. The derivation and preliminary experimental results based on these efficient local search algorithms (described in Section 6.4) have been published in two articles in the European Journal of Operations Research [42, 35].

State-of-the-art algorithm combining ACO with local search In Chapter 7 we obtain a state-of-the-art algorithm for solving the PTSP, by combining one ACO algorithm proposed in Chapter 5 and the powerful 1-shift local search derived in Chapter 6. Chapter 7 also achieves a classification in terms of solution quality of the different local search variants based on different cost approximations for the local search proposed in Chapter 6. The contents and experimental results of Chapters 5 and 7 are being condensed in an article ([40]) in preparation for submission to an international journal.

Using objective function approximations in metaheuristics for the VRPSD Appendix B analyzes the performance of several metaheuristics on the VRPSD, which, like all SCOPs, has a computationally demanding objective function. Two types of approximations of the objective function are used inside the metaheuristics. One approximation based on the deterministic traveling salesman problem reveals to be particularly efficient when coupled to two metaheuristics (Iterated Local Search and Evolutionary Computation), and leads to state-of-the-art algorithms for the VRSPD. This work has been possible thanks to the collaboration of several people from different research labs, who have participated to the "Metaheuristics Network" an European project funded by the Marie Curie programme [143]. Preliminary results have been published in the proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII) [32], and the contents of Appendix B have been accepted for publication by the Journal of Mathematical Modelling and Algorithms [33].

Summarizing, the original contributions of this thesis can be quantified in terms of the following scientific publications:

- Articles in international journals: 3 articles published ([42], [35], [33]), and 2 articles in preparation ([37], [40]).

- Articles in international conference proceedings: 3 articles published ([38], [39], [32]).

It follows the complete list of mentioned scientific publications.

[42] L. Bianchi, J. Knowles, and N. Bowler. Local search for the probabilistic traveling salesman problem: correction to the 2-p-opt and 1-shift algorithms. *European Journal of Operational Research*, 162(1):206–219, 2005.

[35] L. Bianchi and A. M. Campbell. Extension of the 2-p-opt and 1-shift algorithms to the heterogeneous probabilistic traveling salesman problem. *European Journal of Operational Research*. To appear.

[33] L. Bianchi, M. Birattari, M. Manfrin, M. Mastrolilli, L. Paquete, O. Rossi-Doria, and T. Schiavinotto. Hybrid metaheuristics for the vehicle routing problem with stochastic demands. *Journal of Mathematical Modelling and Algorithms*. To appear.

[37] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr. Metaheuristics in stochastic combinatorial optimization: a survey. Technical Report IDSIA-08-06, IDSIA - Dalle Molle Institute for Artificial Intelligence, Manno, Switzerland, March 2006.

[40] L. Bianchi, L. M. Gambardella, and M. Dorigo. Ant colony optimization and local search for the probabilistic traveling salesman problem. Technical Report IDSIA-02-06, IDSIA - Dalle Molle Institute for Artificial Intelligence, Manno, Switzerland, 2006.

[38] L. Bianchi, L. M. Gambardella, and M. Dorigo. An ant colony optimization approach to the probabilistic traveling salesman problem. In J. J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacañas, and H.-P. Schwefel, editors, *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN VII)*, volume 2439 of *Lecture Notes in Computer Science*, pages 883–892. Springer, London, UK, 2002.

[39] L. Bianchi, L. M. Gambardella, and M. Dorigo. Solving the homogeneous probabilistic traveling salesman problem by the ACO metaheuristic. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Proceedings of the 3rd International Workshop on Ant Algorithms (ANTS 2002)*, volume 2463 of *Lecture Notes in Computer Science*, pages 176–187. Springer, London, UK, 2002.

[32] 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. In X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo Guervós, J. A. Bullinaria, J. Rowe, P. Tiňo, A. Kabán, and H.-P. Schwefel, editors, *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, volume 3242 of *Lecture Notes in Computer Science*, pages 450–460. Springer, Berlin, Germany, 2004.

Part I

Metaheuristics for Stochastic Combinatorial Optimization

Chapter 1

Introduction

The focus of the first part of this thesis (from Chapter 1 to Chapter 3) is on Stochastic Combinatorial Optimization Problems (SCOPs), a wide class of combinatorial optimization problems under uncertainty, where all or part of the information about the problem data is unknown, but some knowledge about its probability distribution is assumed. Our intention is to put under a unifying view the several applications of metaheuristics to SCOPs, by filling a gap in the literature, where a number of surveys and books about solving SCOPs via classical techniques exist, but none about using metaheuristics, despite the research literature is already quite rich.

The first part of the thesis is organized as follows. Chapter 1 motivates, in Section 1.1, the interest for studying SCOPs via metaheuristics, and in Section 1.2 proposes a classification of the modeling approaches to uncertainty according to dynamicity and type of description of uncertain data, and precisely defines the scope of SCOPs. Chapter 2 recalls the main formal definitions of both static and dynamic SCOPs from the literature, by providing links to the corresponding algorithmic domains, with the aim of giving a clear view of the intersection between classical approaches and new ones based on metaheuristics. Chapter 2 also introduces the issue of objective function computation in SCOPs, which may involve different types of objective function approximations. Chapter 3 reviews the main applications to SCOPs of metaheuristics for which a significant amount of interesting literature exists, namely Ant Colony Optimization (ACO), Evolutionary Computation (EC), Simulated Annealing (SA), Tabu Search (TS), Stochastic Partitioning Methods (SPM), Progressive Hedging (PH), and Rollout Algorithms (RO). Finally, it selects and discusses some open issues by taking a transversal view on the reviewed metaheuristics.

1.1 Motivation

There is an increasing interest of the operations research community in addressing optimization problems that include in their mathematical formulation uncertain, stochastic, and dynamic information. Problem solving under uncertainty has a very high impact on real world contexts, since optimization problems arising in practice are becoming

Acronym	Full SCOP name
PTSP	Probabilistic Traveling Salesman Problem
TSPTW	Traveling Salesman Problem with Stochastic Time Windows
VRPSD	Vehicle Routing Problem with Stochastic Demands
VRPSDC	Vehicle Routing Problem with Stochastic Demands and Customers
SCP	Set Covering Problem
SSP	Shop Scheduling Problem
SDTCP	Stochastic Discrete Time-Cost Problem
SOPTC	Sequential Ordering Problem with Time Constraint

Table 1.1: Explanation of acronyms used to refer to some relevant SCOPs.

increasingly complex and dynamic, also thanks to the fast development of telecommunications that makes not only the perception but also the changes of the world more rapid, stochastic and difficult to forecast. Therefore, the study of optimization algorithms for SCOPs is an aspect of operations research which is of increasing importance.

In recent years, metaheuristic algorithms such as Ant Colony Optimization (ACO), Evolutionary Computation (EC), Simulated Annealing (SA), Tabu Search (TS), Stochastic Partitioning Methods (SPM), and others, have emerged as successful alternatives to classical approaches based on mathematical and dynamic programming for solving SCOPs. In fact, due to the high complexity and difficulty of optimization problems under uncertainty, often classical approaches (that guarantee to find the optimal solution) are feasible only for small size instances, and they could require a lot of computational effort. In contrast, approaches based on metaheuristics are capable of finding good and sometimes optimal solutions to problem instances of realistic size, in a generally shorter computation time. Table 1.2 lists some papers in the literature providing evidence about the advantages in solving SCOPs via metaheuristics instead of using exact classical methods, and Table 1.1 explains the acronyms used to refer to the SCOPs involved in Table 1.2.

1.2 Modeling approaches to uncertainty

In defining the scope of SCOPs one should consider the many ways in which uncertainty may be formalized. Uncertainty is included in the formulation of optimization problems in order to go nearer to real world conditions, but models should also be a bit simplified, in order to be tractable analytically or numerically. The efforts done in reaching a good trade-off between usefulness of the model and tractability of the problem have produced a multitude of formalizations of uncertainty. This is even more evident for metaheuristics, because, due to their simplicity, they may be easily applied to complex formulations that would be considered intractable for many classical algorithmic approaches.

Reference(s)	SCOP	Metaheuristic(s)	Exact method	Advantage of the metaheuristic(s) over the exact method
Beraldi and Ruszczyński [16] (2005)	SCP	SPM (Beam Search)	Branch and Bound	Maximum deviation from optimal solution is 5%, and in some cases Beam Search finds the optimal solution. The running time reduction with respect to Branch and Bound is roughly between 20% and 90%
Bianchi et al. [32] [33] (2005)	VRPSD	ACO, SA, TS, ILS, EC	Integer L-shaped method by Gendreau et al. [86] solves instances with up to 70 customers	ACO, SA, TS, ILS, EC address instances with up to 200 customers (distance from the optimum unknown)
Gutjahr [100] (2004)	TSPTW	ACO	Complete Enumeration solves in about 4 hours instances with 10 customers	ACO and SA solve instances with up to 20 customers in a few seconds (distance from the optimum unknown)
Branke and Guntsch [51] (2003) [52] (2004), Bowler et al. [50] (2003), Bianchi [38] [39] (2002)	PTSP	ACO, SA	Branch and Cut by Laporte et al. [132] solves instances with up to 50 customers	In [38, 39], ACO addresses instances with up to 200 customers. In [51, 52] ACO addresses instances with up to 400 customers. In [50], SA addresses instances with up to 120 customers (distance from the optimum unknown)
Finke et al. [77] (2002)	SSP	TS	Mixed Integer Linear Programming solves instances with up to 10 jobs and 5 machines	29 out of 30 small instances solved to optimality. Instances with up to 20 jobs and 10 machines have been addressed (distance from the optimum unknown)
Gutjahr et al. [104] (2000)	SDTCP	SPM (Stochastic Branch and Bound)	Complete Enumeration approaches fail to generate results within a reasonable amount of time even for medium-size problems	Stochastic Branch and Bound outperforms classic techniques both in solution quality and in runtime
Costa and Silver [65] (1998)	SOPTC	TS	Branch and Bound solves instances with up to 14 causes, with a computation time from 0.1 to about 30000 seconds	TS is much faster (always 0.1 or 0.2 seconds for the small instances with up to 14 customers). Addressed also big instances with up to 500 customers (distance from the optimum unknown)
Gendreau et al. [88] (1996)	VRPSDC	TS	Integer L-shaped method by Gendreau et al. [86] solves instances with up to 70 customers	TS is faster (from Tables I and II of [88] the time gain of TS with respect to the exact method may be computed)

Table 1.2: Evidence about some of the advantages in solving SCOPs via metaheuristics instead of using exact classical methods. Metaheuristics are capable of finding good and sometimes optimal solutions to problem instances of realistic size, in a generally shorter computation time.

When considering models of optimization problems under uncertainty, there are mainly two aspects to define: first, the way uncertain information is formalized, and second, the dynamicity of the model, that is, the time uncertain information is revealed with respect to the time at which decisions must be taken. The several modeling approaches differ in the way the first and/or the second aspects are defined. Here, we propose a classification of models according to these two aspects, uncertainty and dynamicity, as schematized in Figure 1.1. This thesis (Part I) will then focus only on a subset of models, that correspond to our definition of SCOPs.

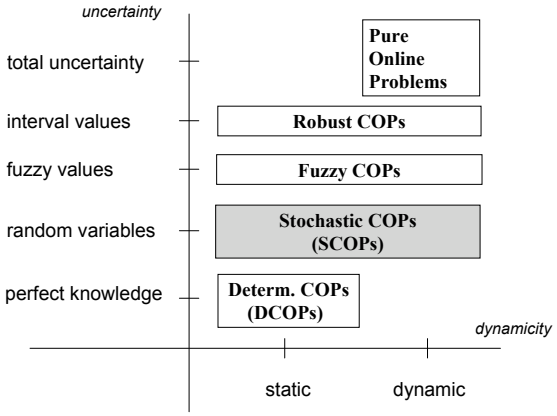


Figure 1.1: Scheme for the conceptual classification of Combinatorial Optimization Problems (COPs) under uncertainty. The first part of this thesis focuses on Stochastic COPs (SCOPs) and to solution methods based on metaheuristics.

Uncertain information may be formalized in several ways (vertical axis of Figure 1.1). The case of perfect knowledge about the data of the problem corresponds to the classical field of solving (Deterministic) Combinatorial Optimization Problems (DCOPs) (low left corner of Figure 1.1). Here, all information is available at the decision stage, and it is used by optimization algorithms to find a possibly optimal solution. The concrete application of a solution found would lead exactly to the cost of the solution as computed by the optimization algorithm, therefore DCOPs are also considered static problems, because from the point of view of the decision maker, there is nothing else to be decided after the optimization took place[1]. A typical example of DCOP is the well known Traveling Salesman Problem (TSP) [96], where, given a set of customers and the set of distance values among each couple of customers, one must find the Hamiltonian tour (that is, a tour visiting once each customer) of minimal length.

[1]Nevertheless, a solution algorithm may use a 'dynamic' or 'stochastic' mechanism also in these cases, as, for example, the dynamic programming algorithm applied to (deterministic, static) shortest path problems, or algorithms that involve some random choice such as virtually all metaheuristics and local search procedures.

Despite its simple formulation, the TSP is an NP-hard problem, like many DCOPs.

Let us now consider problem formulations involving uncertainty (upper levels of Figure 1.1). One possibility is to describe uncertain information by means of random variables of known probability distributions. This is what we assume in SCOPs (a more precise definition and examples of SCOPs will be given in Chapter 2). Under this assumption, the optimization problem is stochastic, and the objective function strongly depends on the probabilistic structure of the model. Typically, the objective function involves quantities such as an expected cost, the probability of violation of some constraints, variance measures, and so on. In SCOPs one can distinguish a time *before* the actual realization of the random variables, and a time *after* the random variables are revealed, because the associated random events happen. Static SCOPs are characterized by the fact that decisions, or, equivalently, the identification of a possibly optimal solution, is done *before* the actual realization of the random variables. This framework is applicable when a given solution may be applied with no modifications (or very small ones) once the actual realization of the random variables are known. The literature sometimes addresses this type of problems as 'a-priori' optimization. As an example of this class of problems, consider the probabilistic TSP (PTSP), that consists in finding a Hamiltonian tour visiting all customers (the 'a priori' tour) of minimum expected cost, given that each customer has a known probability of requiring a visit. Once the information of which customers actually require a visit on a certain day is known, the customers requiring a visit are visited in the order of the 'a priori' tour, simply skipping the customers not requiring a visit.

Dynamic SCOPs arise when it is not possible or not convenient to design a solution that is usable as it is for any realization of the random variables. In this case, decisions that need an optimization effort must be taken also *after* the random events have happened. This could also be done in stages, because it is often the case that the uncertain information is not revealed all at once, but in stages. As an example of dynamic SCOP, consider for instance a TSP where new customers of known positions appear with a certain probability while the salesman has already started to visit the customers known a priori. In this case an a priori tour must be modified dynamically in order to include the new customers in the visiting tour.

Another way of formalizing uncertainty is to identify the uncertain information with fuzzy quantities (vectors or numbers), and constraints with fuzzy sets. This approach has its roots in Bellman and Zadeh [15] and in Zimmermann [187], but currently occupies a minor portion of the optimization literature.

An approach which is receiving increasing attention in the last years is the one of robust optimization, which assumes that uncertain information is known in the form of interval values. For example, one could consider the robust TSP, where the cost of arcs between couples of customers is given by interval values. These costs could have the meaning of travel times, being small if there is no or little traffic, and being high in case of traffic congestion. The robustness approach consists in finding solutions that hedge against the worst contingency that may arise, given that no knowledge about the probability distribution of random data is known. One possible way of quantifying robustness is the *minmax* criterion, under which the robust decision is the one for which

the highest level of cost taken across all possible future input data scenarios is as low as possible. Both static and dynamic versions of robust optimization problems may be formulated. For a good introduction to robust optimization, see for instance the book by Kouvelis and Yu [130].

On the highest level of Figure 1.1 we placed problems that we call Pure Online, where the input is modeled as a sequence of data which are supplied to the algorithm incrementally, but without making any assumption that can help to make a prevision on the new data. An algorithm for a Pure Online Problem produces the output incrementally without knowing the complete input, and its performance is evaluated with respect to an abstract competitor, who knows all the complete (past and future) data, and that is able to solve the offline problem optimally. This way of evaluating algorithms is called in the literature *competitive analysis* [4, 49]. An example of Pure Online problem is the Dynamic Traveling Repair Problem [117], where a set of servers move from point to point in a metric space; the speed of each server is constant, so the time it takes to travel from one point to another is proportional to the distance between two points; time is continuous and at any moment a request for service can arrive at any point in the space; each job also specifies a deadline; if a job is serviced, a server must reach the point where the request originated by its deadline; the goal is to service as many incoming request as possible by their deadlines.

We should again remark that in this thesis we restrict to SCOPs (the shaded box in Figure 1.1). SCOPs are *combinatorial* optimization problems, having by definition a discrete decision space. By this choice we neglect the vast field of continuous optimization under uncertainty, although the scheme we have just proposed for classifying problems under uncertainty equally applies to continuous problems.

SCOPs are relevant in many practical contexts, such as vehicle routing problems, where stochasticity is due to variable customers demands, or variable travel times, routing on information networks, where stochasticity is due to the variability of traffic and the related speed of information packages, finance, scheduling, location problems and many other contexts. All these problem domains may be, and usually are, also modeled as DCOPs. The advantage of using SCOPs over DCOPs is that the solutions produced may be more easily and better adapted to practical situations where uncertainty cannot be neglected, such as thrash collection, cash collection from banks, location of emergency services, and so on. Of course, the use of SCOPs instead of DCOPs comes at a price: first, the objective function is typically much more computationally demanding in SCOPs than in DCOPs; second, for a practical application of SCOPs, there is the need to assess probability distributions from real data or subjectively, a task that is far from trivial. For a discussion about the issue of computational burden and complexity in certain SCOP formulations, see for instance Haneveld and van der Vlerk [109], and Dyer and Stougie [73]. The ways this issue is managed in metaheuristics applied to SCOPs will be described in detail in Chapter 3.

Chapter 2

Formal descriptions of SCOPs

The class of SCOPs is so important and has impact in so many domains that several research areas are dedicated to its study: Stochastic Integer Programming, Markov Decision Processes (which is part of Stochastic Dynamic Programming) and Simulation Optimization being the main ones. Each research area corresponds to a particular way of modeling, formulating and solving optimization problems under uncertainty, and it is often treated separately in the optimization literature. The application of metaheuristics to SCOPs is a quite recent and fast growing research area, and it is thus natural that many of the papers borrow from the classical SCOP literature the same problem formulations. In this chapter we first give, in Section 2.1, a definition of SCOPs which is very general, but is only tentative since it does not well specifies what are the quantities to be minimized or maximized (in a minimization, respectively maximization problem). The limits of this tentative SCOP definition justify the need to consider other formalizations, as we do in Section 2.2 and 2.3, where we recall the main formal definitions of, respectively, static and dynamic SCOPs from the literature. We also provide pointers to the research areas that originally proposed the various SCOP definitions. In Section 2.4, we introduce the important issue of objective function approximations in SCOPs, which will be the leitmotif of the rest of this thesis.

2.1 General but tentative SCOP definition

Let us now give a general definition of SCOP, as proposed by Kall and Wallace [127]. (Throughout the first part of the thesis we use the minimization form for optimization problems, the maximization form is equivalent and can be derived in a straightforward manner by substituting the word 'min' with the word 'max').

Definition 1 (SCOP - tentative)
Consider a probability space (Ω, Σ, P) *([95]), where* Ω *is the domain of random variables* ω *(typically a subset of* $\mathbb{R}^k$*),* Σ *is a family of "events", that is subsets of* Ω*, and* P *is a probability distribution on* Σ*, with* $P(\Omega) = 1$*. Consider also a finite set* S *of decision variables* x*.* S *is typically a subset of* $\mathbb{R}^n$*. The random variable* ω *could also depend on*

the decision variable x, in that case it is denoted by ω_x. Given a cost function G and constraint functions H_i, $i = 1, 2, \ldots, m$, mapping $(x, \omega) \in (S, \Omega)$ to $\mathbb{R}$, find

$$\begin{cases} \text{“}min\text{”}_{x \in S}\ G(x, \omega), \\ \text{subject to } H_i(x, \omega) \leq 0,\ i = 1, 2, \ldots, m. \end{cases} \tag{2.1}$$

Note, however, that according to the above definition, a SCOP is not well defined, since the meaning of “min” as well as of the constraints are not clear at all [127]. In fact, how could one take a decision on x before knowing the value of ω, and how could one verify if $H_i(x, \omega) \leq 0$, if ω is not yet known? Moreover, since ω is a random variable, also $G(x, \omega)$ and $H_i(x, \omega)$ are random variables. For these reasons, Definition 1 is not operational, and it must be refined. There are several possibilities to do this, giving rise to different SCOP variants, both static and dynamic. These are also called *deterministic equivalents* of Definition 1. Let us first focus on static SCOPs, and later on dynamic SCOPs.

2.2 Static SCOPs

Definition 2 (Stochastic Integer Program - SIP)
Given a probability space (Ω, Σ, P), a finite set S of feasible solutions x, a real valued cost function $G(x, \omega)$ of the two variables $x \in S$ and $\omega \in \Omega$, and denoting by $\mathbb{E}_P(G(x, \omega))$ the expected value of $G(x, \omega)$ over Ω according to P, find

$$\min_{x \in S} \big\{ g(x) := \mathbb{E}_P(G(x, \omega)) \big\}. \tag{2.2}$$

The above definition is maybe the simplest SCOP formulation, and it does not consider random constraints (observe, though, that deterministic constraints could be implicitly included in the definition of the domain S of decision variables).

In some cases the cost function G is deterministic, that is, G only depends on x and not on the random variable ω, but constraints do depend on the random variable ω. In such situation it might be impossible to enforce $H_i(x, \omega) \leq 0$ for all $\omega \in \Omega$. Thus, one could relax the notion of constraint satisfaction by allowing constraint violation, and by imposing that constraints are satisfied at least with some given probabilities. This leads to the following

Definition 3 (Chance Constrained Integer Program - CCIP)
Given a probability space (Ω, Σ, P), a finite set S of feasible solutions x, a real valued cost function $G(x)$, a set of real valued constraint functions $H_i(x, \omega)$, and a set of constraint violation probabilities α_i, with $0 \leq \alpha_i \leq 1$ and $i = 1, 2, \ldots, m$, find

$$\begin{cases} \min\limits_{x \in S}\ G(x), \\ \text{subject to } \mathrm{Prob}\{H_i(x, \omega) \leq 0\} \geq 1 - \alpha_i,\ i = 1, 2, \ldots, m. \end{cases} \tag{2.3}$$

Both the Stochastic and Chance Constrained Program formulations have been originally proposed in the context of Mathematical Programming applied to SCOPs, and this field is also called in the literature Stochastic Integer Programming (SIP), a subset of the broader field of Stochastic Programming [46]. The Stochastic Programming community has a very active website [168] where updated bibliographic references and papers are available. Recent surveys on SIP include [109] and [128] (the latter overviews SIP applications in the context of location routing problems). Let us now focus on some dynamic SCOP deterministic equivalents of Definition 1.

2.3 Dynamic SCOPs

Informally, a stochastic dynamic problem is a problem where decisions are taken at discrete times $t = 1, \ldots, T$, the horizon T being finite or infinite. Decisions taken at time t may influence the random events that happen in the environment after t. In dynamic SCOPs the concept of solution used in static SCOPs is no longer valid. For example, in the dynamic TSP that we described in Chapter 1.2, a tour among the set of customers known at the beginning of the day cannot be traveled as it is in practice, but it must be modified when new observations (new customers) are known. What the decision maker can do before the observation-decision process starts is to decide which *policy* (or *strategy*) to adopt, that is, to specify a set of rules that say what action will be taken for each possible random future event. For example, in the dynamic TSP, a possible policy consists in re-optimizing the portion of route among the not-yet-visited customers each time that a new customer appears. Another policy, which is less computationally expensive, but that possibly leads to a more costly tour, is to re-optimize at stages, only after a certain number of new customers has appeared. Note that in solving dynamic SCOPs one has to make a double effort: first, decide which policy to adopt, second, given the policy, solve the optimization sub-problems emerging dynamically. Both parts have influence on the final solution cost, but often the choice of the policy is due to factors that are outside the control of the decision maker. For instance, in the dynamic TSP one could be forced not to optimize every time a new customer arrives, in case customers want to know in advance the vehicle arrival time.

Among the dynamic formulations the most common ones are those belonging to the class of Stochastic Programming with Recourse (Two-stage and Multiple-stage Integer Stochastic Programs) and Markov Decision Processes.

Definition 4 (Two-stage Stochastic Integer Program - TSIP)
Given a probability space (Ω, Σ, P), a finite set S_1 of first-stage decisions x_1, a finite set S_2 of second-stage decisions x_2, and real valued cost functions f_1 and f_2, find

$$\min_{x_1 \in S_1} \big\{ g_1(x_1) := f_1(x_1) + \mathbb{E}_P\big(G(x_1, \omega)\big) \big\}, \tag{2.4}$$

where

$$G(x_1, \omega) := \min_{x_2 \in S_2(x_1, \omega)} f_2(x_1, x_2, \omega). \tag{2.5}$$

Given the above definition, solving a Two-stage Stochastic Integer Program consists in solving two problems: a DCOP for the second-stage (equation (2.5)), and a Stochastic Integer Program (Definition 2) for the first-stage (equation (2.4)). The meaning of the two-stage decision process is the following. The first-stage decision x_1 must be taken before knowing the actual value of the random variable ω. After the value of ω is observed, it may be convenient or necessary to take some other decision (the second-stage decision x_2) in order to better adapt to the new situation discovered. The second-stage decision is also called *recourse* action, because in some practical situations it has the effect of 'repairing' the consequences of an action (x_1) taken before knowing the value of the random variable. Informally, a Two-stage Stochastic Integer Program cosists in finding the best decision now, with the hypothesis that I will also take the best decision when I will know the value of the random quantities. A practical example of a Two-stage Integer Program is the Vehicle Routing Problem with Stochastic Demands (VRSPD), where a vehicle tour among a set of customers is decided, prior of knowing the actual demand of each customer. The vehicle travels along the tour, and the driver discovers the actual demand of a customer only when arriving at that customer. When a customer demand is known and the customer has been serviced, the next best decision may be to go back to the depot for replenishment, or to proceed to the next planned customer. The choice between these options is part of the second-stage optimization problem. In this context, the tour planned a priori may be interpreted as the first-stage decision x_1, while the set of return trips to the depot may be interpreted as the second-stage decision x_2.

The Two-stage Stochastic Integer Program may be easily extended to the general Multi-stage case.

Definition 5 (Multi-stage Stochastic Integer Program - MSIP)
Consider T decision stages $t = 1, 2, \ldots, T$, and correspondingly, T decision variables $x_t \in S_t$ (with S_t finite subsets depending on $(x_1, \ldots, x_{t-1}, \omega_1, \ldots \omega_{t-1})$), and T random variables ω_t belonging to probability spaces $(\Omega_t, \Sigma_t, P_t)$. The problem consists in finding

$$\min_{x_1 \in S_1} \left\{ g(x) := f_1(x_1) + \mathbb{E}_{P_1}(G_1(x_1, \omega_1)) \right\} \tag{2.6}$$

where, for $t = 1, 2, \ldots, T-2$,

$$\begin{aligned} G_t(x_1, \ldots, x_t, \omega_1, \ldots \omega_t) = \min_{x_{t+1} \in S_{t+1}} \big[& f_{t+1}(x_1, \ldots, x_{t+1}, \omega_1, \ldots, \omega_{t+1}) \\ & + \mathbb{E}_{P_{t+1}}(G_{t+1}(x_1, \ldots, x_{t+1}, \omega_1, \ldots, \omega_{t+1})) \big], \end{aligned} \tag{2.7}$$

and

$$G_{T-1}(x_1, \ldots, x_{T-1}, \omega_1, \ldots, \omega_{T-1}) = \min_{x_T \in S_T} f_T(x_1, \ldots, x_T, \omega_1, \ldots, \omega_T). \tag{2.8}$$

Observe that, from the above definition, solving a Multi-stage Stochastic Integer Program consists in solving one DCOP for the last stage (equation (2.8)), and $T-1$ Stochastic Integer Programs for the intermediate stages (equations (2.6) and (2.7)).

Let us now introduce the framework of Markov Decision Processes (MDP). The central concept in MDP is the *state*, which at each time step describes the knowledge about the problem (called here system). In MDP the Markov property is assumed, that is, future behavior of the system does only depend on past history through the current state and the current decision taken. Obviously, this also depends on the way a state is defined. Often, the state corresponds to the physical description of the system, but this is not always the case. We now briefly provide a standard formulation of MDP. For a complete discussion, see for instance [151].

Definition 6 (Markov Decision Process - MDP)
Consider a 4-tuple (X, A, P, C), where X is a finite state space, and A is a finite set of possible actions or decisions. At state x, we denote the set of admissible actions by $A(x)$. For each $x \in X$, $A(x)$ is a finite set. P is a state transition function, that describes the stochastic behavior of the system over time: at time t, if the current state is $x \in X$, and action $a \in A$ is chosen, then the next state is y with probability $P(y|x, a)$. P is thus a function from $X \times A(X) = \{(x, A(x)|x \in X)\}$ to a probability distribution over X. C specifies the costs of actions depending on the state in which they are performed. Taking an action a in state x has a cost $c(x, a)$. C is a function from $X \times A(X)$ to $\mathbb{R}$.

A policy π is defined as a mapping from X to $A(X)$ that associates to every x a feasible action $a \in A(x)$, and Π is the set of all possible policies. Informally, a policy tells what actions need to be taken at which state, and this is what the decision maker needs to know in order to take a decision at each discrete decision time, once the actual state of the system is known.

Let us now define the cost associated to a policy. Let $X_t, t = 0, 1, 2, \ldots, T$ be a random variable that denotes the state at time t. For a given policy $\pi \in \Pi$, and a given initial state $X_0 = x_0$, if the decision maker follows the policy π over time, a particular system path is given as a sequence of states and actions $(X_0 = x_0, a_0, X_1 = x_1, a_1, \ldots, X_t = x_t, a_t, X_{t+1} = x_{t+1}, a_{t+1}, \ldots, a_{T-1}, X_T = x_T)$, where a_t is the action taken at time t and $a_t = \pi(x_t)$, with $x_t \in X$. Over the system path, the system accumulates the discounted costs defined, for $T < \infty$, as

$$\sum_{t=0}^{T} \gamma^t C(x_t, \pi(x_t)), \text{ with } \gamma \in (0, 1]. \tag{2.9}$$

For $T = \infty$, γ (that is called discount factor) must be smaller than 1. Given a policy π, the accumulated discounted cost (Equation (2.9)) is a random quantity, due to the randomness of the system path. The expected discounted cost of a policy over all possible system paths may be computed as follows:

$$J(\pi) = \sum_{x_0, \ldots, x_T} P_\pi(x_0, x_1, \ldots, x_T) \left[\sum_{t=0}^{T} \gamma^t C(x_t, \pi(x_t)) \right], \tag{2.10}$$

with $\gamma \in (0, 1]$, and where P_π is the probability of a particular path:

$$P_\pi(x_0, x_1, \ldots, x_T) = \delta(x_0) \Pi_{t=0}^{T} P(x_{t+1}|x_t, \pi(x_t)), \tag{2.11}$$

$\delta(x_0)$ *being the probability that state* $X_0 = x_0$.

Given a 4-tuple (X,A,P,C) and the associated finite set of policies Π*, an MDP consists in finding*

$$\min_{\pi \in \Pi} J(\pi). \tag{2.12}$$

The model we have defined is known both as Markov Decision Process and Stochastic Dynamic Programming. The first term comes from the fact that for any fixed policy, the resulting model is a Markov chain. The second term comes from the family of solution methods based on Dynamic Programming [19], originally designed for solving optimal control problems. Dynamic Programming and MDP are also related to (and could be seen as part of) Reinforcement Learning [172] and Neuro-Dynamic Programming [22], which mainly focus on methods for approximating optimal solutions of MDP and for dealing with incomplete information about the state of the system. Stochastic Dynamic Programming is also related to Stochastic (Mathematical) Programming, and in particular to Multi-stage Integer Programs. For a clear exposition on the relation between these two domains, see for instance Kall and Wallace [127]).

Finding the optimal policy can be a prohibitive task unless the state and/or action space is very small, which is usually not the case. For example Value Iteration, a well known exact method for solving MDP [151], has a computational complexity polynomial in $|X|$, $|A|$, and $1/(1-\gamma)$, and one single iteration takes $O(|X|^2 \cdot |A|)$. There are several approximation algorithms (that is, algorithms that do not guarantee to find the optimal solution) that try to find good solutions in a feasible computation time via techniques such as structural analysis, aggregation, sampling feature extraction and so on. See, for example, [151, 172, 22]. Recently, also some metaheuristics have been applied to approximately solve MDP, and they are discussed in the remainder of the first part of this thesis.

2.4 Objective function computation

As we have seen, all the above SCOP formulations involve the computation of one or more expected values for evaluating the objective function. As a consequence, three different situations may arise when computing SCOP objective functions:

1. closed-form expressions for the expected values are available, and the objective function is *computed exactly* based on these objective values;

2. as in case 1, closed-form expressions for the expected values are available, but the computation of the objective function value is considered to be too time consuming to be done during optimization. Therefore, *ad hoc and fast approximations* of the objective function are designed and used during optimization (possibly alternating exact and approximated evaluations);

3. the problem is so complex in terms of decision variables and/or in terms of probabilistic dependencies, that no closed-form expression exists for the computation

SCOP formulation	Exact	Ad hoc approximation	Sampling approximation
SIP	[74], [175]	[38], [39], [51], [52], [32], [33], [76]	[99], [100], [45], [181], [185], [186], [121], [84], [102], [103], [156], [79], [7], [6], [115], [5], [56], [105], [157], [138], [77], [67], [65], [101], [104], [150]
CCIP		[16]	[9]
TSIP	[139]	[88]	
MSIP		[155], [136], [23]	
MDP	[60], [61]	[134], [135]	

Table 2.1: Papers on metaheuristic applications to SCOPs, classified according to SCOP formulation (rows) and type of objective function computation (columns).

of the expected value of the objective function. Therefore, the objective function value is estimated by simulation with the so-called *sampling approximation*.

All the three above situations have been addressed within the metaheuristics literature, as summarized in Table 2.1. Let us now give some introductory information on the use of ad hoc and sampling approximations in SCOPs.

2.4.1 Ad hoc approximations

The design of ad hoc approximations is strongly problem dependent, and no general rule exists for finding efficient approximations of the objective function. Examples of ad hoc approximations in the literature include: the use of the objective function of a DCOP similar in some respects to the SCOP considered; the use of truncated expressions for the expected values, by neglecting terms that are estimated to be small; the use of scenarios, instead of considering the true probabilistic model. Ad hoc approximations, if on one side accelerate the evaluation and comparison among solutions, on the other side introduce a systematic error in the computation of objective values. Usually, the systematic error cannot be reduced unless a different, more precise ad hoc approximation is designed, and it can only be evaluated by comparison with the exact objective value. Thus, metaheuristics typically alternate exact and approximated evaluations during the optimization process. More details about the way ad hoc approximations are used by metaheuristics are given in Chapter 3.

2.4.2 Sampling approximation

When a closed-form expression for the expected value(s) of the objective function is not available, one common choice is to estimate expectations by Monte Carlo-type

simulations. For example, in the case of the Stochastic Integer Program (Definition 2), the objective function $g(x)$ is typically approximated by the sample average

$$g_N(x) := \frac{1}{N} \sum_{j=1}^{N} G(x, \omega_j) \tag{2.13}$$

where $\omega_1, \omega_2, \ldots, \omega_N$ is a random sample of N independent, identically distributed (i.i.d.) realizations of the random vector ω. The sample average is also referred to as sample estimate, and the random realizations as random scenarios. In this thesis, we will use these terms interchangeably.

The main difference between SCOPs requiring simulation for estimating the objective function and DCOPs, or SCOPs with exactly computable objective function is that, in the first-mentioned case, it is not possible to decide with certainty whether a solution is better than another one. This can only be tested by statistical sampling, obtaining a correct comparison result only with a certain probability. Thus, the way sampling approximation is used in metaheuristics largely depends on the way solutions are compared and the best solution among a set of other solutions is selected ('selection-of-the-best' method).

A huge research area devoted to solving problems with simulated objective function is Simulation Optimization. Following the definition given by Fu [81], Simulation Optimization means "searching for the settings of controllable decision variables that yield the maximum or minimum expected performance of a stochastic system that is presented by a simulation model." A compact picture of the field is given by the reviews of the Winter Simulation Conference [8, 146]. The latest one by Ólafsson and Kim [146] emphasizes discrete problems and practical approaches, including some references to metaheuristics. Until a few years ago, the literature on Simulation Optimization was especially focussed on theoretical results of convergence of mathematically elegant algorithms. Interestingly, as noted by Fu [80], the many new commercial software packages for simulation do not take advantage of the theoretical results of the literature. On the contrary, most of them rely on metaheuristics such as Genetic Algorithms and Neural Networks, that are more easily adaptable to complex real-world simulations, but often their integration into commercial packages lacks rigor and is not provably convergent. Fu speculates that an interesting direction of research would be the development of algorithms that take advantage of the theoretical results of the literature, but are still flexible and applicable to real-world situations, so to fill the gap between theory and practice. Indeed, recent developments in Simulation Optimization, especially relying on metaheuristics, go in this direction.

Chapter 3

Metaheuristics for SCOPs

A metaheuristic is a set of concepts that can be used to define heuristic methods capable of being applied to a wide range of different problems. 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 that have been applied to SCOPs include: ACO, EC, SA, and TS (the meanings of these acronyms are listed in Table 3.1). For a review and comparison among these and other metaheuristics in the context of DCOPs, see for instance the paper by Blum and Roli [48], and the publications pointed to by [143]. Besides these metaheuristics, there are some algorithms such as Stochastic Partitioning Methods, Progressive Hedging, and Rollout Algorithms (see Table 3.1) that could be called metaheuristics, even if they are not commonly known as such, or that make use of metaheuristics as part of the algorithmic procedure.

The following sections focus on the cited metaheuristics. The format of Section 3.1, 3.2, 3.3, 3.4, and 3.5 describing respectively ACO, EC, SA, TS and SPM is the same: first some background information on the principles of the metaheuristic is given, then the results and issues of applying the metaheuristic to SCOPs are reviewed. For each metaheuristic, the reviewed papers have been grouped in different parts, respectively focussing on:

- SCOPs with exactly computed objective or ad hoc approximations;
- SCOPs with sampling approximation;
- Markov Decision Processes.

MDPs have been treated separately because they require a particular modeling effort for each metaheuristic, which is quite different from the modeling of static SCOPs and of TSIPs and MSIPs. Section 3.6 briefly gives references to the SCOP literature involving metaheuristics that are still at their early stage in the SCOP domain (PH and RO). Finally, Section 3.7 takes a transversal view on the reviewed metaheuristics, and selects some open issues for discussion.

Acronym	Full name of the metaheuristic
ACO	Ant Colony Optimization
EC = (EP+ES+GA)	Evolutionary Computation = (Evol. Programming + Evol. Strategies + Genetic Algorithms)
SA	Simulated Annealing
TS	Tabu Search
SMP = (BS+SBB+NP)	Stochastic Partitioning Methods =(Beam Search + Stoch. Branch and Bound + Nested Partitions)
PH	Progressive Hedging
RO	Rollout Algorithms

Table 3.1: Acronyms used for the metaheuristics described in this thesis.

3.1 Ant Colony Optimization

3.1.1 Introduction to ACO

The first algorithms based on the ant colony analogy appeared at the beginning of the nineties in a paper by Dorigo et al. [70] later published as [71]. ACO is now a widely studied metaheuristic for combinatorial optimization problems, as the recent book by Dorigo and Stützle [72] testifies.

The inspiring concept that links optimization with biological ants is based on the observation of their foraging behavior: when walking on routes from the nest to a source of food, ants seem to find not simply a random route, but a quite 'good' one, in terms of shortness, or, equivalently, in terms of time of travel; thus, their behavior allows them to solve an optimization problem. This kind of success of biological ants is entirely explained by their type of communication and by their way of deciding where to go: While walking, ants deposit a chemical called *pheromone* on the ground, and they tend to choose routes marked by strong pheromone concentrations. Given two initially unexplored routes, a short and a long one, between the nest and the source of food, ants choose at first randomly which one to walk. Ants that have chosen, at first by chance, the shorter route are the first to reach the food and to start their return to the nest. Therefore, pheromone starts to accumulate faster on the shorter route than on the longer one. Subsequent ants tend to follow the shorter route because it has more pheromone, thus reinforcing it more and more, and further attracting other ants on the good route.

Combinatorial problems addressed by ACO are usually encoded by a *construction graph* $G = (V, A)$, a completely connected graph whose nodes V are components of solutions, and arcs A are connections between components. Finding a solution means constructing a feasible walk in G. For example, in the TSP nodes correspond to customers, arcs correspond to streets connecting customers, and a feasible solution is a Hamiltonian path on the graph. The construction graph encoding is also used in current ACO applications to SCOPs and to dynamic optimization problems. Some examples are also described in [72].

The ACO algorithm is essentially the interplay of three procedures [68]: ConstructAntsSolutions, UpdatePheromones, and DeamonActions, as represented by Algorithm 1.

ConstructAntsSolutions is the process by which artificial ants construct walks on the construction graph incrementally and stochastically. For a given ant, the probability p_{kl} to go from a node k to a feasible successor node l is an increasing function of τ_{kl} and $\eta_{kl}(u)$, where τ_{kl} is the pheromone on arc (k, l), and $\eta_{kl}(u)$ is the *heuristic* value of arc (k, l), which should be a reasonable guess of how good arc (k, l) is (for example, in the TSP η_{kl} is the reciprocal of the distance between customer k and customer l). The heuristic value may depend on the partial walk u.

UpdatePheromones is the process by which pheromone is modified on arcs. Pheromone may be both increased and decreased. Pheromone is modified (decreased) by each ant on each arc as soon as it is added to a partial walk on the construction graph, this operation is called *local update*. Moreover, pheromone is further modified (increased) on selected good solutions to more strongly bias the search in future iterations, and this operation is called *global update*. Decreasing pheromone on selected arcs is important, in order to avoid too rapid convergence of the algorithm to suboptimal solutions. Interestingly, pheromone decreases also in the biological environment, due to evaporation.

DeamonActions are centralized operations, such as comparing solution values among ants in order to find the best solution, or running a local search procedure.

Algorithm 1 Ant Colony Optimization (ACO)

```
while termination condition not met do
  ScheduleActivities
    ConstructAntsSolutions
    UpdatePheromone
    DeamonActions
  end ScheduleActivities
end while
```

Several convergence proofs have been provided for the ACO metaheuristic. Convergence in value, which, informally, means that the algorithm will find at least once the optimal solution, has been given by Gutjahr [97] and by Stützle and Dorigo [169]. Convergence in solution, which, informally, means that the algorithm will generate over and over the optimal solution, has been given by Gutjahr [98]. For details and for a comprehensive discussion, see Dorigo and Stützle [72].

3.1.2 ACO for SCOPs

The investigation of ACO applications to SCOPs is at its early stages, the first works being appeared at conferences after year 2000. Nevertheless, the ACO literature contains both theoretical and experimental works that cover both static and dynamic SCOPs.

3.1.2.1 Exact objective and ad hoc approximation

The first SCOPs that have been addressed by ACO are the probabilistic TSP (PTSP), in Bianchi et al. [38, 39] and Branke and Guntsch [51, 52], and the vehicle routing with stochastic demands (VRPSD) in Bianchi et al. [32, 33]. These problems are Stochastic Integer Programs with closed form expression for the objective function, that is, the objective function is computable a priori, independently of particular random realizations of the stochastic variables.

The PTSP and the VRPSD have in common the fact that their solution structure (and the corresponding construction graph) is very similar to their deterministic counterpart (the TSP, respectively capacitated VRP). The main difference with the respective deterministic counterpart problem is the much higher computational complexity of the objective function in the stochastic version of the problem. In the PTSP, the objective function is computable in $O(n^2)$ time, n being the number of customers, while in the TSP it only requires $O(n)$ time. In the VRPSD, the objective requires $O(nKQ)$, where n is the number of customers, K is the number of possible demand values of each customer, and Q is the vehicle capacity, while the capacitated VRP objective only requires $O(n)$ time. The fact that the difference between the stochastic and deterministic versions of these problems mainly lies in the objective function makes them particularly appropriate for studying a first application of ACO to SCOPs. In fact, in this case it is possible to apply to the stochastic problem an ACO algorithm originally designed for the deterministic problem with almost no modifications.

In [38, 39], the authors experimentally investigate on the PTSP two versions of ACO: ACS and pACS. ACS, that was originally designed for the TSP by Gambardella and Dorigo [82] and by Dorigo and Gambardella [69], solves the PTSP using the objective function of the TSP (the length of a Hamiltonian path) as a rough but fast approximation of the PTSP objective function. The second version of ACO considered in [38, 39], pACS, is identical to ACS except for the fact that it uses the PTSP objective function (the expected length). Such difference implies that pACS considers as good solutions different solutions with respect to ACS, and so pACS reinforces pheromone (during global updating) on different solutions with respect to ACS, with the consequence that ants in pACS converge on different solutions. Note, however, that ACS and pACS use the same, TSP specific, heuristic information (the reciprocal of the distance between two customers). Experimental results on PTSP instances with homogeneous customers probabilities have shown that pACS is better than ACS, except for the case when the customers probabilities are close to 1, in which case ACS is more efficient than pACS. This means that the overhead of the time consuming PTSP objective function is not justified in those cases where the approximate objective function, which can be computed much faster, is close enough to the exact one. The idea to employ faster approximations of the exact objective function has been further developed in [51, 52]. The authors propose an ad hoc approximation of the expected cost that neglects the least probable customers configurations. This approximation is shown experimentally to accelerate convergence without significantly worsening the solution quality. Another issue addressed by [51, 52] is the design of PTSP-specific heuristics to guide the ants

construction process. The authors experimentally analyze different heuristics, and show that one of them indeed improves the quality of solution constructed by ants, but at the cost of a higher computational time.

An important aspect in designing ACO for SCOPs (but also for classical DCOPs), is the application of a local search procedure to improve solutions found by ants (the local search is part of the DeamonActions of Algorithm 1). In order to be competitive with state-of-the-art algorithms, it has been *necessary* for ACO algorithms to use a local search both in the PTSP [52] and the VRPSD [32]. Unfortunately, designing an effective local search for a stochastic problem with a computationally expensive objective function may be a quite challenging task. The reason is that in local search it is very important to compute efficiently the change, or 'delta', of the objective function between two neighboring solutions. When the objective function is complex like in most SCOPs, it is difficult to find a delta expression which is both exact and fast to be computed. For the PTSP it has been possible to derive for two local search operators, the 1-shift and the 2-p-opt, recursive, fast and exact expressions for the objective function delta [42, 35]. The 1-shift and 2-p-opt are very efficient, since they can explore the whole neighborhood of a solution in $O(n^2)$ time, the same time it would take for the same operators in the TSP. For the VRPSD, a local search operator is available, the OrOpt, with an efficient delta expression which is not exact, but approximated, and has been introduced by Yang et al. in [183] (we will call this 'Yang approximation'). In Bianchi et al. [32, 33], besides the Yang approximation, one based on computing the length difference between two neighboring solutions has been considered. This last approximation is equivalent to treat a VRPSD solution (which is a Hamiltonian path) like a solution for the TSP, and it is faster but less accurate than the Yang approximation. In [32, 33], the impact of using the two above types of delta approximation has been tested on several metaheuristics, namely ACO, EC, SA, TS, and Iterated Local Search. In ACO, the use of the rough but efficient TSP approximation lead to better results than the Yang approximation (even though ACO was not able to reach the quality of the best performing metaheuristics, that were Iterated Local Search and EC).

3.1.2.2 Sampling approximation

When ACO is applied to this type of problems, the DeamonActions procedure (Algorithm 1) must implement ways of performing statistical tests for comparing the sample average values of the solutions generated by the ants, in order to select the best solution (or a set of best solutions). Sampling could also be used in ConstructAntsSolutions, in order to estimate heuristic values $\eta_{k,l}(u)$, when the chosen heuristic depends on random variables.

The first sampling-based ACO, called S-ACO, has been proposed and analyzed by Gutjahr in two papers [99, 100]. The first paper [99] theoretically analyzes S-ACO, by proving convergence to the optimal solution with probability one. The second paper [100] experimentally studies S-ACO on two stochastic routing problems, the PTSP, and the TSP with time windows and stochastic service times (TSPTW). S-ACO has been applied in a third paper by Rauner et al. [152] to a policy optimization problem in

healthcare management. Algorithm 2 summarizes the functioning of S-ACO, showing in particular how sampling is used; for details about procedures ConstructAntsSolutions and UpdatePheromone, see [99, 100]. In every iteration, after ants have constructed their

Algorithm 2 S-ACO

1: **for** iteration $k = 1, 2, \ldots$ **do**
2: ConstructAntsSolutions [s ants construct their walk x_σ, $\sigma = 1, 2, \ldots, s$ on the graph G]
3: from $\{x_1, \ldots, x_s\}$ select a walk x;
4: **if** $k = 1$ **then**
5: set $x^* = x$ [x^* is the current approximation of the optimal solution]
6: **else**
7: based on N_k independent random scenarios ω_ν, compute a sample estimate $g_k(x) = 1/N_k \sum_{\nu=1}^{N_k} G(x, \omega_\nu)$ of x;
8: based on N_k independent random scenarios ω'_ν, compute a sample estimate $g_k(x^*) = 1/N_k \sum_{\nu=1}^{N_k} G(x^*, \omega'_\nu)$ of x^*;
9: **if** $g_k(x) < g_k(x^*)$ **then**
10: set $x^* = x$;
11: **end if**
12: **end if**
13: UpdatePheromone
14: **end for**

solutions x_σ, only one ant solution x is selected for being further compared with the current best solution (step 3 of Algorithm 2). Interestingly, for the sake of convergence, it does not matter how the ant solution is selected [99]. A possible way to do it, which has been chosen in [100], is to evaluate each x_σ on a same random scenario drawn specifically for a given iteration, and to take x as the solution with the best value. In the case of the more complex problem treated in [152], selecting x_σ based on a single random scenario turned out as suboptimal; better results were obtained by choosing several (but not too many) scenarios. After x has been selected, it is then again evaluated, together with the current best solution x^*, in order to decide whether it is better than x^*. This is done by estimating x by sampling over N_k scenarios ω_ν and x^* over N_k scenarios ω'_ν. In the convergence proof of [99], it has been necessary to impose that ω_ν and ω'_ν are independent, but in practice [100], if $\omega_\nu = \omega'_\nu$ S-ACO also performs well. The number of sample scenarios N_k is a critical parameter of S-ACO: if too small, the estimate and comparison of solutions will be often faulty, but if N_k is too big, the computational time required for one solution evaluation could become a problem. As shown in [99], for proving convergence it is sufficient that N_k increases linearly with the iteration number k. This result is interesting especially if compared with the faster than quadratic increase recognized as necessary for the corresponding SA approach in [102, 115]. In practical implementations of S-ACO, it may be more convenient to choose the sample size N_k adaptively, based on some statistical test. In [100], one version of S-ACO establishes the sample size by means of a parametric statistical

test: N_k is gradually increased till when the difference between the sample estimation for the two solutions being compared is larger than 3 times their estimated standard deviation. This kind of sample schedule, also known as variable-sample strategy, has been theoretically analyzed in the context of random search algorithms by Homem-de-Mello [116].

More recently, the ACO/F-Race algorithm has been proposed by Birattari et al. [45], where at each iteration the selection of the new best solution (steps 3 to 12 of Algorithm 2) is done with a procedure called F-Race, which is more sophisticated than the simple parametric test of S-ACO. As explained in [45], F-Race consists in a series of steps at each of which a new scenario ω is sampled and is used for evaluating the solutions that are still in the race (at the beginning, all solutions generated by ants in a given iteration, together with the current best solution, are in the race). At each step, a Friedman test is performed and solutions that are statistically dominated by at least another one are discarded from the race. The solution that wins the race is stored as the new current best solution. Preliminary experiments on homogeneous instances of the PTSP problem have shown that ACO/F-Race improves over the parametric procedure adopted by S-ACO.

3.1.2.3 Markov Decision Processes

To our knowledge, there are only two papers that use ACO to solve MDP, the first one by Chang et al. [60] and the second one by Chang [59]. Both papers design ACO algorithms to solve MDP, and theoretically analyze their properties by providing convergence proofs.

Chang et al. [60] focus on MDP with infinite horizon (that is, $T = \infty$). This simplifies a little the problem, because in this case the optimal policy is stationary, that is, it does not depend on time. The construction graph on which ACO algorithms proposed in [60] are based is represented in Figure 3.1. The states in X are arranged

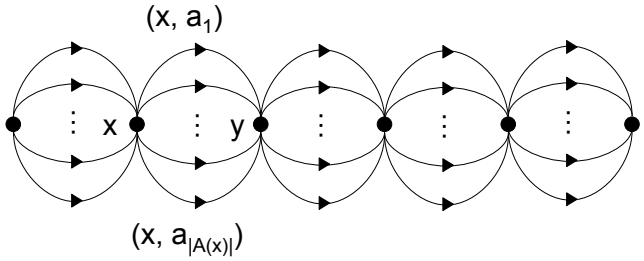


Figure 3.1: Construction graph of the ACO algorithm proposed in [60] for solving Markov Decision Processes with stationary policies.

in an arbitrary order $\mathcal{O}$; each state $x \in X$ corresponds to a vertex of the construction graph, and each arc of the construction graph corresponds to a pair (x, a) of state $x \in X$ and an admissible action $a \in A(x)$. The direction of the arc goes from the

current state x to the next state with respect to the order $\mathcal{O}$. A particular ant traverses all the states in the construction graph following the order $\mathcal{O}$ from the first state of $\mathcal{O}$. When it moves from a state x to a state y, it traverses randomly the arc (x, a) with a transition probability that depends on the pheromone on that arc, and on some heuristic appropriately defined (see [60] for details). Once an ant finishes the tour, it has generated a stationary policy, which, in the context of MDP, is a candidate solution to the problem. In [60], two ACO versions are proposed, ANT-PI and ANT-TD. The first one is inspired by a well-known exact method for MDP, policy iteration (PI) [151], and assumes that the state transition probabilities P of the MDP system are known. The second one considers the case of unknown P, and uses one the well-known reinforcement learning algorithms called temporal difference learning (TD) [172] for evaluating the average value of a given policy. It is proven that both ANT-PI and ANT-TD probabilistically converge to the optimal solution. For practical implementations, due to the high computational complexity inherent to the problem, the authors suggest that parallel implementations of the proposed ACO algorithms are used.

In Chang [59], the focus is on MDP with completely unknown system dynamics, that is, unknown state transition probability P *and* unknown costs C. A finite horizon T is considered and, for simplicity, it is assumed that every action is admissible at every state. An ACO algorithm called ANT-EE is proposed, which is based on a TD algorithm for evaluating a policy, similarly to the ANT-TD algorithm of [60]. Theorems are provided that show that ANT-EE has the same convergence properties as Q-learning [177], one of the best known basic techniques in reinforcement learning.

3.2 Evolutionary Computation

3.2.1 Introduction to EC

EC [31] is a collective term for all variants of optimization algorithms that are inspired by Darwinian evolution. In this context, a solution to a given optimization problem is called an *individual*, and a set of solutions is called a *population*. The basic structure of an EC algorithm is given by Algorithm 3. Every iteration of the algorithm corresponds to a *generation*, where certain operators are applied to some individuals of the current population to generate the individuals of the population of the next generation. Typical operators are those of *recombination*, that recombine two or more individuals to produce new individuals, and *mutation*, that modify single individuals to obtain self-adaptation. At each generation, only some individuals are selected for being elaborated by recombination and mutation operators, or for being just repeated in the next generation without any change, on the base of their fitness measure (this can be the objective function value, or some other kind of quality measure). Individuals with higher fitness have a higher probability to be selected.

In the literature there are mainly three different categories of EC that have been developed independently from each other: Evolutionary Programming (EP), proposed by Fogel et al. in 1966 [78], Evolutionary Strategies (ES) proposed by Rechenberg in 1973 [153], and Genetic Algorithms proposed by Holland in 1975 [114]. Presently, algorithms

that fall in the EP and ES category mostly apply to continuous optimization problems, while GA are more specific for discrete and combinatorial optimization problems. Recent overviews about EC include Hertz and Kobler [112], and Calégari et al. [57]. For the convergence properties of EC and GA, see for instance Rudolph [161], Vose [180], and Reeves and Rowe [154].

Algorithm 3 Evolutionary Computation (EC)

P = GenerateInitialPopulation()
while termination condition not met **do**
  P' = Recombine(P)
  P'' = Mutate(P')
  Evaluate(P'')
  P = Select($P'' \cup P$)
end while

3.2.2 EC for SCOPs

There is a very large amount of literature on applying EC to optimization problems 'under uncertainty', such as problems with noisy fitness, with time varying and dynamic fitness, and with approximated fitness. For a recent survey on how the EC literature addresses different types of uncertainty, see Jin and Branke [123]. Other reviews include a book by Arnold [10], and a paper by Beyer [30]. Although SCOPs are a particular case of optimization under uncertainty, the translation to the SCOP domain of the results from the EC literature on optimization under uncertainty is not easy. The main difficulty is that most papers focus on continuous optimization, and they often restrict their attention to the optimization problem characterized by ad hoc test functions, such as the 'spherical' objective function $f(\mathbf{x}) = \mathbf{x}^T\mathbf{x}$, $\mathbf{x} \in \mathbb{R}^N$. Also when discrete optimization problems are considered, experiments are often restricted to the 'onemax bit-counting' function, which can be regarded as the counterpart of the spherical objective function in binary search spaces.

In the following, we outline the contributions of EC in the SCOP domain, and we also highlight the main ideas and methods proposed for problems under uncertainty that may be relevant for SCOPs, even if they have not been directly tested on specific problems from this domain.

3.2.2.1 Exact objective and ad hoc approximation

The EC literature addressing this kind of problems may be roughly divided in two groups. In the first group ([74, 139]) EC algorithms use the exactly computable objective function as it is, even if computationally expensive, while in the second group ([32, 33], and references cited in [122]) EC exploits also computationally more efficient objective function (fitness) approximations. Let us briefly analyze these two groups of papers.

In [74] Easton and Mansour apply a distributed GA to three different labor scheduling problems, one of which is formulated as a stochastic goal programming problem. Their algorithm operates in parallel on a network of three workstations. Separate sub-populations evolve independently on each processor, but occasionally the fittest solutions migrate over the network to join the other sub-populations. Also infeasible solutions are accepted (with a fitness penalty) in order to encourage the exploration of promising regions of the search space. The proposed GA is compared experimentally to a SA and a TS metaheuristic previously developed by other authors (respectively in [54, 55] and in [75]), and it is shown to outperform both of them.

In [139] Guo and Mak consider a vehicle routing problem with stochastic demand and soft time windows, which is formulated as a Two-stage Stochastic Integer Program (Definition 4). The authors propose an EC algorithm called Age-GA where, instead of being replaced by their offspring after each iteration, individuals may grow up and generate new offspring continuously before death, and the population comprises individuals from various age-groups. With the same amount of computational effort, it is possible to use a larger population size in Age-GA than in a canonical GA. The paper shows that, on a set of eighteen randomly generated instances, Age-GA outperforms a canonical GA without the aging mechanism.

To the group of papers using in SCOPs efficient approximations of the fitness function belong [32, 33] by Bianchi et al. (also cited in Section 3.1), that compare a simple EC with other metaheuristics (ACO, SA, TS, and Iterated Local Search) for the VRPSD. Similarly to the other metaheuristics, EC is integrated with the OrOpt local search operator, where two approximations for the objective value difference between neighboring solutions have been tested, the Yang and the TSP approximation. The exact VRPSD objective function is used for accepting a new solution in the local search, and for the selection of a new population. EC, like ACO and Iterated Local Search, performs better with the TSP approximation. Interestingly, EC is improved even more when in [33], instead of OrOpt, a more TSP-specific local search (3-opt) is used. EC, together with Iterated Local Search, is shown to be the best performing among the tested metaheuristics.

Here, it is useful to note that there is a thread in the EC literature that focuses on the use of computationally efficient approximations of the original fitness in continuous optimization problems. Some aspects of this issue that are developed in the context of continuous optimization may be relevant to SCOPs as well. Fitness approximations are also known as approximate models, meta-models or surrogates. A comprehensive survey on fitness approximation in EC has been written by Jin [122]. This growing research area is particularly oriented to continuous optimization problems with extremely time consuming objective function computations, such as, for instance, structural design optimization [12], where one single fitness evaluation may take over ten hours on a high-performance computer. The issue of how the approximate model can be incorporated in the EC algorithm, which has been widely addressed by the EC literature on fitness approximation, is quite independent from the continuous or discrete nature of the optimization problem. Nevertheless, most of the ideas still haven't been applied to SCOPs. For a review and pointers to existing literature, see Section 4 of [122].

3.2.2.2 Sampling approximation

The EC literature about optimization with noisy fitness function is also relevant for SCOPs with sampling estimated objective function. In fact, noise is mostly assumed to be additive with zero-mean, which is the case when Monte Carlo sampling is used to estimate the objective function. Section II of Jin and Branke [123] is a good overview about the methodological approaches used in EC to deal with noisy fitness functions. The authors identify three main strategies for dealing with noise: explicit averaging, implicit averaging, and modifying the selection process.

Explicit averaging corresponds to the computation of sample averages of the fitness function performing repeated measures of the noisy fitness and computing their average. This is very similar to the Simulation Optimization technique we have illustrated in Section 2.4.2. Aizawa and Wah [3] propose either to increase the number of samples with the generation counter, or to higher the number of samples for individuals that have a high estimated variance. Stagge [167] proposes to adjust the number of samples according to the probability of an individual of being among the μ best ones. Branke and Schmidt [53] also propose an adaptive sampling method that takes additional samples of two individuals participating in a tournament until the normalized fitness difference between the two individuals falls below some threshold. The normalized fitness difference is obtained dividing the difference of the observed fitnesses by the standard deviation of the difference: $(g_N(x) - g_N(y))/\sigma_d$. Cantú-Paz [58] uses an adaptive sampling method that consists in taking the smallest number of samples necessary to make a decision between competing individuals during the selection process. Their approach is very similar to Branke and Schmidt's, but differs in that they take samples one at a time from the individual with the highest observed variance, and they use standard statistical tests to select the winner of the tournament with a certain probability. Similar to the approach of Cantú-Paz, Teller and Andre [174] propose a method that allocates varying numbers of samples to evaluate individuals. Individuals are initially evaluated with a small number of samples, and are further evaluated only if there is some chance that the outcome of the tournaments they participate in can change. A similar technique has been developed by Giacobini et al. [89].

The second type of strategy for dealing with noise in EC is implicit averaging. Its aim is to reduce the influence of noise by using a large population size, instead of performing more fitness measures of a single individual. The intuition behind implicit averaging is the following ([123], p.305): Because promising areas of the search space are sampled repeatedly by the EC algorithm, and there are usually many similar solutions in the population, when the population is large, the influence of noise in evaluating an individual is very likely to be compensated by that of a similar individual. This can be regarded as an implicit averaging effect. In the literature, conflicting conclusions have been reported on whether it is more effective the use of implicit or explicit averaging, given a fixed fitness evaluation number per generation is allowed. For a summary about the history of implicit averaging we direct the reader to [123] and to the references cited therein.

The third strategy used in EC to reduce the influence of noise is modifying the se-

lection process of individuals. One example is to accept an offspring individual only if its fitness is better that that of its parents by at least a predefined threshold, as done by Markon et al. [140]. Beielstein and Markon [14] study the relationship between threshold selection and hypothesis testing techniques. Another way of modifying the selection process with respect to standard EC is to eliminate random choices during selection, in order to exploit the uncertainty due to the noisy fitness as a sort of randomization effect. This method has been proposed by Branke and Schmidt [53].

Convergence properties of EC and the dynamics of the fitness function when noise is present have been analyzed in several papers, for example by Miller and Goldberg [142] and by Beyer [30].

In all the above cited papers aiming at reducing the influence of noise via implicit and explicit averaging, or via modifications of the selection process, the computational experience is unfortunately limited to ad hoc continuous or discrete test functions. It appears that an experimental validation of the various techniques in the SCOP domain is still missing in the literature. In fact, the few papers applying EC to SCOPs that we are going to outline below, either use very simple techniques for dealing with noise or rely on methods that are unrelated to the main EC literature on noisy fitness functions.

Watson et al. [181] address a stochastic warehouse scheduling problem where the objective function must be estimated by simulation. The authors consider a GA, a solution construction heuristic specific for that problem, two local search and a random search algorithm. Two versions of the GA and the local search algorithms are considered where the (set of) starting solution(s) is randomly generated in one case, and provided by the constructive heuristic in the other case. In order to keep the run time of the algorithms feasible, the simulator is used in a fast but inaccurate mode. Only final solutions are eventually evaluated with a more accurate - two order of magnitude slower - simulator mode. The constructive heuristic exploits specific knowledge about the internal states of the simulator in order to construct a solution. Instead, in the GA and local search algorithms the simulator is used as a black box, that, provided a solution, returns a real value indicating the solution quality. Experimental results show that GA initialized with the domain-specific construction heuristic outperforms all the other algorithms. Moreover, all algorithms perform worse when initialized by random solutions. The results also highlight an interesting phenomenon related to the use of a black box, fast but inaccurate simulator for the evaluation of solutions during the execution of the GA. As better and better solutions according to this simulator are found, it is observed that the correlation with solution values given by the slow-accurate simulator (evaluated a posteriori) decreases. This implies that the final solution returned by the GA as best solution may be quite bad with respect to the nearly-exact objective value. It is reasonable to think that this is a general phenomenon that can happen in any metaheuristic exploiting a non-exact or noisy objective function evaluation, particularly when estimating the objective function by sampling and with a fixed (low) number of samples. One possibility to overcome this problem is to keep in memory a set of promising solutions encountered during the execution of the algorithm, and to evaluate them a posteriori with the accurate simulator, or to apply more sophisticated adaptive sampling techniques. Yoshitomi [185] and Yoshitomi and Yamaguchi [186] use

GA for solving the stochastic job-shop scheduling problem. In both papers, the best solution is extracted among the set of solutions that have been more frequently present through the generations of the GA. In [186], Monte Carlo sampling is used to select among the set of most frequent solutions the best final solution. Other applications of GA based on Monte Carlo sampling are the ones by Sudhir Ryan Daniel and Rajendran [171] applying GA to the inventory optimization problem in a serial supply chain, and by Jellouli and Châtelet [121] using GA for addressing a supply-chain management problem in a stochastic environment.

3.2.2.3 Markov Decision Processes

Chang et al. [61] propose an EC algorithm called Evolutionary Policy Iteration (EPI) for solving inifinite horizon discounted reward MDPs. EPI is particularly suited for problems where the state space is small but the action space is very large. This situation makes the use of the well-known policy iteration (PI) algorithm [151] for solving MDPs impractical, since PI must perform a maximization over the entire action space. EPI eliminates the need of maximizing over the entire action space by directly manipulating policies via a method called policy switching that generates an improved policy from a set of given policies. The computation time for running the policy switching is on the order of the state space size. The algorithmic structure of EPI is that of standard GA, with appropriate modifications and extensions required for the MDP setting. EPI iteratively generates a population or a set of policies such that the performance of the best policy for a population monotonically improves with respect to a defined fitness function. In [61], it is proved that EPI converges with probability one to a population whose best policy is an optimal policy.

Lin et al. [134, 135] focus on finite horizon partially observed Markov decision processes (POMDPs), an extension of MDPs that consider situations such as: State observations are costly; sensors that measure the current state value are noise-corrupted; at least part of the current state value is inaccessible. In [134, 135], a GA is developed for finding a good approximation of the value function. In [135], the GA is combined with a mixed integer programming algorithm, and this combination is capable of determining the value function and of finding the optimal policy in less computation time than other exact methods.

Yokoama and Lewis III [184], address a stochastic dynamic production cycling problem whose original formulation is that of an MDP. In order to reduce the search space dimensionality, the problem is first re-formulated in a two-level problem. The first level consists in a DCOP that, in order to be solved, needs that a series of MDP sub-problems (constituting the second level) are solved. The first level DCOP is addressed by a GA, which calls dynamic programming algorithms as sub-routines to solve the second level MDPs.

3.3 Simulated Annealing

3.3.1 Introduction to SA

The SA algorithm has been introduced in the area of combinatorial optimization by Kirkpatrick et al. [129]. It relies on a model developed by Metropolis et al. [141] for simulating the physical annealing process, where particles of a solid arrange themselves into a thermal equilibrium. An introduction to SA can be found in van Laarhoven and Aarts [179] or Aarts and Korst [1].

The standard type of applications concerns combinatorial optimization problems of the form

$$min_{x \in S}\ g(x),$$

where S is a finite set of feasible solutions. The algorithm uses a pre-defined neighborhood structure on S. A control parameter which is called "temperature" in analogy to the physical annealing process governs the search behavior. In each iteration, a neighbor solution y to the current solution x is computed. If y has a better objective function value than x, the solution y is "accepted", that is, the current solution x is replaced by y. If, on the other hand, y has a worse objective function value than x, the solution y is only accepted with a certain probability depending on (i) the difference of the objective function values in x and y, and (ii) the temperature parameter.

In pseudocode, the SA algorithm can be represented as follows (cf. [1], p. 16):

Algorithm 4 Simulated Annealing (SA)

```
Initialize state x and temperature parameter T_1;
for iteration k = 1, 2, ... do
  select y randomly from S(x);
  if g(y) ≤ g(x) then
    set x = y;
  else if exp((g(x)−g(y))/T_k) ≤ uniform[0,1] then
    set x = y;
  end if
  update T_k to T_{k+1};
end for
```

Therein,

- x and y are feasible solutions from S;
- T_1,T_2,... is a (usually decreasing) sequence of values for the temperature parameter; the update of the values T_k is done according to a so-called *cooling schedule*;
- the sets $S(x)$ form the pre-defined *neighborhood structure*: to each feasible solution $x \in S$, a set $S(x) \subseteq S \setminus \{x\}$ of "neighbor solutions" is assigned;
- uniform$[\alpha, \beta]$ is a procedure selecting a uniformly distributed (pseudo-)random number from the interval $[\alpha, \beta]$.

Several results showing convergence of SA to the set of optimal solutions under suitable cooling schedules have been obtained by diverse authors, for example Geman and Geman [85], Gelfand and Mitter [83], or Hajek [107]. Essentially, convergence can be assured by a cooling schedule where T_k is decreased as $\Gamma / \log k$, with sufficiently large Γ. In practice, cooling is usually done faster for computation time reasons. (For more details, see [1].)

3.3.2 SA for SCOPs

In the literature, several extensions of the SA algorithm above have been suggested for treating Stochastic Integer Programs (Definition 2), both in the case of exact objective and ad hoc approximation, and sampling approximation.

3.3.2.1 Exact objective and ad hoc approximation

One early application of SA in the context of SCOPs is due to Teodorović and Pavković [175]. The authors address a VRPSD with multiple vehicles, and use SA in two stages, first for partitioning the customers among the different vehicles, and second to improve the single vehicle routes. In this preliminary work, computational results are reported only for one instance of 50 customers.

More recently, the already cited papers [32, 33] by Bianchi et al. (see Section 3.1 and 3.2) have applied to the VRPSD a simple SA algorithm, together with other metaheuristics (ACO, EC, TS, and Iterated Local Search). Similarly to the other metaheuristics, two approximations for the objective value difference between neighboring solutions generated according to the OrOpt scheme have been tested, the Yang and the TSP approximation. Differently from what happens for ACO, SA performs better when using the more accurate but more computationally expensive Yang approximation. On average, SA does not perform significantly different from ACO, and it is not able to reach the quality of the best performing metaheuristics, that are EC and Iterated Local Search.

3.3.2.2 Sampling approximation

Algorithm 5 shows a typical basic structure of an SA modification to the solution of Stochastic Integer Programs (Definition 2) with sampling estimated objective function. The approaches from the literature outlined below follow this general scheme. Differences stay particularly in the way step 5 (estimation of the objective value), step 11 (choice of a new approximation of the optimal solution), and step 12 (temperature level) are implemented in Algorithm 5.

Gelfand and Mitter [84] investigate the case where the observation of the objective function $g(x)$ is disturbed by random noise W_k in iteration k of the SA process, such that instead of $g(x)$, the estimate $g_k(x) = g(x) + W_k$ is observed. They show that if W_k is normally distributed with mean zero and variance σ_k^2, if certain conditions on the values σ_k and on acceptance probabilities are satisfied, and if a suitable cooling

Algorithm 5 Stochastic Simulated Annealing (SSA)

```
Initialize state x, temperature parameter T_1 and sample size N_1;
Set x* = x [x* is the current approximation of the optimal solution];
for iteration k = 1, 2, ... do
    select y randomly from S(x);
    compute sample average estimates g_k(x) and g_k(y) for the costs in x resp. y;
    if g_k(y) ≤ g_k(x) then
        set x = y;
    else if exp((g_k(x) − g_k(y)) / T_k) ≤ uniform[0,1] then
        set x = y;
    end if
    compute a new current approximation x* of the optimal solution;
    update T_k to T_{k+1};
    update N_k to N_{k+1};
end for
```

schedule (ensuring convergence of ordinary SA) is used, then the convergence property of ordinary SA remains valid.

Gutjahr and Pflug [102] follow a similar approach by showing that under suitable conditions on the "peakedness" (Birnbaum [47]) of the noise distribution, convergence of the current solutions to the set of optimal solutions can be guaranteed. To be more specific, let us call a symmetric distribution μ_1 more peaked around zero than a symmetric distribution μ_2, if for all $t > 0$, the probability mass on the interval $[-t, t]$ is larger or equal under μ_1 than under μ_2. Then, if the distribution of the noise W_k is more peaked around zero than a normal distribution $N(0, \sigma_k^2)$, where $\sigma_k = O(k^{-\gamma})$ with a constant $\gamma > 1$, the distribution of the solution in iteration k converges as $k \to \infty$ to the uniform distribution on the set of global optimizers, provided that a suitable cooling schedule (ensuring convergence of ordinary SA) is used. Decreasing σ_k with the required rate can be achieved by increasing the sample size N_k more than quadratically in k, that is, by imposing that $N_k = O(k^{\mu})$ with $\mu > 2$. An application of the technique of [102] to a discrete time/cost tradeoff problem in activity planning has been reported in [103].

Other approaches have been presented by Roenko [156], who proposes to store the feasible solutions produced during the execution of the algorithm and to compare them with the solution generated in each current iteration, and by Fox and Heine [79], who derive a convergence result based on the assumption that with probability one, the objective function estimates $g_k(x)$ coincide after some finite time with the true objective function values $g(x)$, as it can be achieved by consistent estimators in the case of only finitely many *possible* objective function values. The last assumption can also be relaxed, if some more complicated condition can be verified, but Fox and Heine argue that in each computer representation, objective function values are taken from some finite domain (given by the machine number precision) anyway. The algorithm indicated by Fox and Heine does not use independent sampling from scratch in each

iteration, as it is done in [84] and [102], but cumulates the sampling results, which is of course advantageous from a computation time viewpoint.

Alrefaei and Andradóttir [7] pursue a different idea by keeping the temperature parameter T_k *constant* during the process instead of decreasing it toward zero (as usual in ordinary SA). To obtain convergence, two alternative techniques are suggested. The first, let us call it A1, consists in the following procedure: In each iteration k, for the current solution x chosen in this iteration, a counter $V_k(x)$ is increased by one in order to register how often x has been visited since the start of the algorithm. The current number $V_k(x)$ of visits is divided by the number $D(x)$ of neighbors of x. The estimated optimal solution x^* in iteration k is then defined as that solution $x^* = x$ for which $V_k(x)/D(x)$ is maximal, among all the solutions x that have been encountered so far. The second technique, let us call it A2, is to estimate the objective function value of solutions x and y (step 5 of Algorithm 5), by cumulating previous estimates of x and y (if any), and then, choose as new approximation x^* of the optimal solution at iteration k the solution with the smaller estimated objective value, among all solutions evaluated so far. Both A1 and A2 compute sample averages with an increasing number of samples at each iteration k.

Alrefaei and Andradóttir show that both alternatives guarantee, under mild conditions, convergence with probability 1 to the set of optimal solutions. Their article also reports on experimental comparisons showing a superiority of the introduced new algorithms over the previous approaches in [84], [102] and [79]; among the two new algorithms, A2 turns out to yield better results than A1. The experiments are restricted to a test instance with only 50 feasible solutions, therefore it is not clear whether the results can be generalized to larger search spaces; nevertheless, the empirical findings give some evidence that using the solution with best objective function estimate so far as the proposed solution may be a very good choice. Interestingly, for the considered test instance, a random-search-like neighborhood structure including all elements of S (different from x) into the neighborhood $S(x)$ of x produces, for all tested algorithms, better results than a more restricted neighborhood. This seems to indicate that in the stochastic case, the hill-climbing feature of SA gains importance only for larger solution spaces S.

A further important contribution of [7] is that the article discusses optimization both in a transient and in a steady-state simulation context. It is shown that if $g(x)$ is given as the expectation of a functional $G(x, \omega)$ of a stochastic process in either a transient or a steady-state situation, then the theoretical result derived for the simple static SCOP case (corresponding to our Definition 2) still remains valid.

One practical limitation of approaches such as the two just described by Alrefaei and Andradóttir [7] and the one by Roenko [156] is that they require the storage of information about all or most of the solutions encountered by the algorithm, and this is an infeasible task for problems that have a combinatorial nature.

Alkhamis et al. [6] use again a decreasing cooling schedule for the parameters T_k. They propose to decide on acceptance or rejection of a neighbor solution y by means of a *statistical significance test*: A confidence interval for the difference between the true objective function values in x resp. y is computed; depending on the position of

the value zero in relation to this confidence interval, the neighbor solution is judged as equal, better or worse than the current solution x. After that, the usual acceptance rule of SA is applied. The authors are able to show that on certain conditions on sample size and cooling schedule, the classical SA convergence property is still satisfied.

Homem-de-Mello [115], [116] presents a comprehensive framework for describing and analyzing variants of SA for SCOPs. The framework enables a thorough theoretical analysis and opens a broader range of flexibility in the choice of sampling distributions. Using ergodicity theory, Homem-de-Mello proves in [115] a rather general convergence theorem for a variable-sample modification of SA. The theorem includes the result in [102] as a special case, but does not make use of any normality assumptions related to noise distributions anymore. In [116], this approach is further generalized beyond the area of SA, although the described analytical techniques and algorithmic ideas remain applicable in a SA context, as well as in the context of other metaheuristics dealing with SCOPs with objective function estimated by sampling. In particular, the author presents the interesting idea of adaptively modifying the sample size N_k during the iterations of the algorithm, in such a way that N_k is usually only increased if the result of a t-test indicates that higher accuracy of the objective function estimates is required. To preserve the convergence property, the sample size is increased at some specific points in time regardless of the t-test.

In Alkhamis and Ahmed [5], the acceptance rule based on confidence intervals developed in [6] is modified by applying the constant-temperature schedule of [7] instead of the classical decreasing temperature schedule. As the current estimated solution, the authors take the solution with the maximum (normalized) number of visits so far. Again, a convergence result is given.

There are also some purely experimental papers involving SA and SCOPs with sampling estimated objective function. The earliest is a paper by Bulgak and Sanders [56] addressing a buffer allocation problem in the context of a complex manufacturing system. The objective function to be maximized (the efficiency of the system) is estimated by means of a discrete event simulator. Similarly to [116], an adaptive sampling procedure is used, where the number of samples is gradually increased for testing whether a candidate solution is statistically better than the current best solution.

Haddock and Mittenthal [105] investigate the feasibility of using an SA algorithm in conjunction with a simulation model to find the optimal parameter levels at which to operate a system. The authors modify Kirkpatrick et al. [129] by substituting an estimate of the expected value of the system response (the objective function) in all places requiring a deterministic objective function value.

Rosen et al. [157] propose a combined procedure, called RS team method, that improves the SA of Haddock and Mittenthal [105] by initially searching for good solutions to be then employed as starting solutions by SA. The initial search for good starting solutions is done by the use of first-order linear approximations of the model, adapting the technique of response surface methodology to the case of a discrete decision space. The RS team method is tested on a simulation model of a semi-conductor manufacturing process consisting of over 40 workstations, and it is experimentally compared with the SA algorithm of Haddock and Mittenthal [105].

Bowler et al. [50] use a stochastic SA algorithm to experimentally analyze the asymptotic behavior of (sub)optimal homogeneous PTSP solutions, in the limit of pn (customers probability times number of customers) going to infinity. The PTSP objective function is estimated by sampling, and the sampling estimation error is used instead of the annealing temperature. Temperature decrease during the execution of the SA algorithm is mimicked by an increase in the accuracy of the objective function estimation, which, in turn, is obtained by increasing the number of samples.

Finally, two papers [100] and [150] focus on different metaheuristics, but involve SA in experimental comparison. The paper by Gutjahr [100] that we also cited in Section 3.1 focuses on S-ACO, and reports experimental comparisons between S-ACO and the SSA algorithm of Gutjahr and Pflug [102]. Pichitlamken and Nelson [150], while focusing on a Stochastic Partitioning Method that will be described in Section 3.5, use the SA algorithm of Alrefaei and Andradóttir [7] as a term of comparison in the experimental analysis of their algorithm.

Although variants of SA for SCOPs have received a great deal of attention in the last decade, such that, for example, the question under which conditions convergence to the optimal solution is ensured can now be considered as relatively well understood, there is a comparably smaller body of comprehensive *experimental* results aiming at interesting questions such as: Which properties of the problem instance make which algorithmic variant well-suited? In particular, there seems still to be little empirical knowledge about the influence of the search space size on the performance of the single variants.

3.4 Tabu Search

3.4.1 Introduction to TS

The main ideas characterizing the TS metaheuristic were independently proposed in the eighties by Glover [90] and Hansen [110], and since then TS has been widely applied to combinatorial optimization problems. A comprehensive introduction to TS can be found in the book by Glover and Laguna [94], or in Hertz, Taillard and de Werra [113].

TS is essentially a sophisticated and improved type of *local search*, an algorithm that in its simplest form, also known as Hill Climbing, works as follows. Consider a starting current solution, evaluate its neighboring solutions (according to a given neighborhood structure), and set the best or the first found neighbor which is better than the current solution as new current solution. Iterate this process until an improving solution is found in the neighborhood of a current solution. The local search stops when the current solution is better than all its neighbors, that is, when the current solution is a *local optimum*.

Such a simple and very general local search behaves quite poorly in practice, particularly because when a local optimum is found, the algorithm stops improving, and combinatorial problems often have local optima whose objective values are much worse than that of the global optimum. The strength of the TS metaheuristic with respect to simple local search is that, by employing three TS-specific concepts, it avoids to get pre-

maturely stuck in a local optimum. These TS-specific concepts are: *best improvement*, *tabu lists*, and *aspiration criteria*.

Best improvement means that each current solution is always replaced by its best neighbor, even if the best neighbor is worse than the current solution. This is clearly a way not to get stuck in local optima. Using best improvement poses the problem of possible cycling among already visited solutions, because it is possible, for example, that the best neighbor of a solution is indeed the last visited current solution. In order to avoid cycling, choosing recently visited solutions is forbidden, by storing some attributes of these solutions in the so-called *tabu lists*. Whole solutions are not stored in a tabu list, because this would require too much memory for most combinatorial optimization problems. The choice of attributes is a delicate point. Typically, tabu lists store the 'moves' that should be performed in order to go from one solution to another, or the differences between solutions. In this way the memory requirement of tabu lists is feasible, but another problem arises: forbidding all solutions corresponding to a tabu attribute may forbid also solutions that have not yet been visited, and possibly also very good or optimal solutions. TS employs *aspiration criteria* for solving this problem. An aspiration criterion is a condition that, if satisfied, allows to set as new current solution a solution obtained by performing a tabu move. A typical example of aspiration criterion is requiring that a solution is better than the best solution found from the beginning of the algorithm.

In pseudocode, the TS metaheuristic may be represented as in Algorithm 6, where

Algorithm 6 Tabu Search (TS)

Generate a starting current solution x
Initialize the tabu lists
for iteration $k = 1, 2, \ldots$ **do**
  Set $A(x,k) = \{y \in S(x) \backslash T(x,k) \cup \tilde{T}(x,k)\}$
  Set $x = \arg\min_{y \in A(x,k)} g(y)$
  Update the tabu lists and the aspiration criteria
end for

x, y are feasible solutions of the combinatorial optimization problem, $A(x,k)$ is the set of solutions among which the new current solution is chosen at iteration k, $S(x)$ is the set of neighbors of x, $T(x,k)$ is the set of tabu moves at iteration k, and $\tilde{T}(x,k)$ is the set of tabu moves satisfying at least one aspiration criterion. In TS, typical stopping criteria may be a maximum CPU time, a maximum number of consecutive iteration not producing an improving solution, or the emptiness of the set $A(x,k)$.

Theoretical properties of convergence of TS to the optimal solutions has been analyzed only quite recently by Hanafi [108] and by Glover and Hanafi [93]. Both papers derive convergence results for a version of TS where the choice of a given neighborhood and a decision criterion for selecting moves force some solutions to be revisited before exploring other new ones.

3.4.2 TS for SCOPs

In comparison to the wide literature about TS for DCOPs, there are still very few papers applying TS to SCOPs. These works are all experimental, and address static SCOPs both in the case of exact objective and ad hoc approximation, and sampling approximation.

3.4.2.1 Exact objective and ad hoc approximation

As we have already pointed out, one of the major difficulties when solving SCOPs is that the objective function, even if explicitly computable, is computationally expensive. In local search algorithms, TS included, it is crucial to be able to evaluate the neighborhood of a solution efficiently. Therefore, one of the main issues of applying TS to SCOPs is indeed to find efficient approximations of the objective value difference between couples of neighboring solutions.

Gendreau et al. [88] propose a TS algorithm for solving the vehicle routing problem with stochastic demands and customers. One of the major contribution of their paper is indeed the development of an easily computed approximation for the objective function, used for the evaluation of potential moves. The proposed TS was quite successful in experiments: for instances up to about 50 customes, it was able to find optimal solutions in about 90% of cases, with an average deviation of 0.38% from optimality.

Other papers applying TS to SCOPs are the already cited [32, 33] by Bianchi et al. (see Section 3.1, 3.2, and 3.3), where a simple TS algorithm has been compared with other metaheuristics (ACO, EC, SA, and Iterated Local Search). Similarly to the other metaheuristics, two approximations for the objective value difference between neighboring solutions generated according to the OrOpt scheme have been tested, the Yang and the TSP approximation. Even if the two approximations have different characteristics (the first one is more accurate but more computationally expensive than the second), the quality of results produced by the two versions of TS seemed to be quite insensitive to the type of approximation. In [33], TS obtained results better than ACO and SA, but worse than EC.

3.4.2.2 Sampling approximation

In the literature two types of contributions may be distinguished: papers that inside TS use simulation as a black box for the evaluation of the objective value of solutions, and papers that adapt the simulation procedure to the different components of TS, such as neighborhood exploration, setting of tabu moves, verification of aspiration criteria, in order to speed up the computation.

To the first group belong the papers by Lutz et al. [138], Finke et al. [77], Dengiz and Alabas [67]. These papers apply quite standard TS techniques, and are usually very time consuming, since the evaluation of solutions by simulation is a time consuming process often relying on extern or commercial simulation packages. The advantage of using simulation is that in this way the real objective function is considered, in

problems where a rigorous mathematical programming formulation would impose severe unrealistic restrictions.

Among the second group of papers (adapting simulation to the different components of TS), we describe in some detail the works by Costa and Silver [65] and by Aringhieri [9]. Costa and Silver [65] describe a TS algorithm for a problem in the context of cause-effect analysis, where the true cause of an undesirable effect must be recognized and eliminated. Given that the time to investigate a cause is a random variable with known probability distribution, the goal is to establish a fixed sequence of n causes so as to maximize the expected reward associated with discovering the true cause within a specified time horizon. This problem is also called stochastic ordering problem with time constraint (SOPTC).

The TS developed in this context, called NTS (Noisy TS), is based on sampling and statistical tests, and is suited for all optimization problems where the evaluation of the objective function is computationally expensive due to the presence of noise in the problem definition. In the following we only describe the characteristics of NTS that are directly related to the stochastic nature of the problem. What we do not describe, is part of standard TS techniques for permutation problems. The objective value of a new current solution is computed by a sample average of the type of Equation (2.13). N samples are generated according to the so-called descriptive sampling technique as described in [126], in order to obtain substantial variance reduction with respect to other sampling methods. Descriptive sampling has been adopted by Costa and Silver also because in this way the quality of estimation of the exact objective value does not depend on the quality of the pseudo-random generator used. The estimation of the objective function takes $O(Nn)$ time, and if N is large enough to guarantee a good estimation quality, this computation may be quite time consuming. For this reason, the evaluation of the (possible many) neighbors of a solution is done with the following method relying on a smaller number of samples. A statistical test is used to decide whether a considered neighbor $y_c \in A(x,k)$ is better than the best neighbor $y_b \in A(x,k)$ examined so far in the current iteration k. The decision is done in two phases. First, a small number $N_c < N$ of samples is randomly generated for estimating the expected value $g_{N_c}(y_c)$ of y_c. The decision as to wether the true objective value of y_c, is higher than that of y_b is done by hypothesis testing. Second, if the test 'has decided' that y_c is better than y_b, this is further ascertained, by using all the N samples. If it results that $g_N(y_c) > g_N(y_b)$, than y_b is replaced by y_c. Since N is finite, notwithstanding the use of this double-check procedure, there is a certain probability that y_b is not truly the best feasible neighbor, and that the best solution so far is updated with not the truly best solution so far. In order to lesser the risk of missing a very good solution due to the bad quality of sampling, NTS keeps track of the ns best solutions encountered so far. At the end of the run all solutions in this list are re-evaluated with a number of samples $\kappa > N$, and the best solution according to this new estimation is the solution returned by NTS.

In [65], the influence on the performance NTS of several factors has been experimentally analyzed: the hypothesis testing technique (the t-test, the Wilcoxon test, and the Median test have been compared), and the number of samples N, N_c and κ

to be used in the different phases of NTS. NTS has been compared also with a TS that is similar in everything to NTS, except for the fact that the objective function is computed exactly on the base of a closed form expression available for SOPTC, and no hypothesis test is performed. TS outperforms NTS both in computation time and solution quality, but the solution quality is only slightly better than NTS. This is a result that encourages the use of NTS for problems with very complex, or impossible to compute, objective functions. Note, however, that when a closed form expression for the objective function is available, even if it is quite computationally expensive like in SOPTC, it may still be more efficient to use the classical TS algorithm, instead of NTS.

An application where sampling is employed to save time with respect to using the exact objective function is the one by Aringhieri [9], that applies TS to a Chance Constrained Program (Definition 3). Constraints are supposed to be linear functions, that is, in Definition 3 we pose $H_i(x,\omega) = \sum_j a_{ij}x_j - b_i(\omega)$, with $j = 1, 2 \ldots, n$ and $x \in S \subset \mathbb{R}^n$. Note that in this problem, only the vector b is assumed to be random. In the proposed TS, sampling is used to estimate the probability $p_i(x,k)$ that at iteration k solution x violates constraint b_i. Given a set of Nm random samples $b_{i,r}$, $i = 1, 2, \ldots, m$, $r = 1, 2, \ldots N$, the probabilities are estimated as follows

$$p_i(x,k) = \frac{\sum_{r=1}^{N} \delta_{i,r}}{N}, \quad \text{where } \delta_{i,r} = \begin{cases} 1 & \text{if } \sum_j a_{ij}x_j - b_{i,r} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Probabilities $p_i(x,k)$ are used to define the concept of *probably tabu* moves that in practice extends the set of tabu moves. A move is *probably tabu* at iteration k, if it leads to a solution x for which $p_i(x,k) > \alpha_i$, $i = 1, 2, \ldots, m$ (compare this with Equation (2.3)). Given the set $P(x,k)$ of probably tabu neighbors of x, the new TS, called SIMTS-CCP (simulation TS for Chance Constrained Programs), can be obtained from algorithm 6 by modifying the computation of $A(x,k)$ as

$$A(x,k) = \{y \in S(x) \backslash T(x,k) \backslash P(x,k) \cup \tilde{T}(x,k)\}. \tag{3.2}$$

In [9] the SIMTS-CCP algorithm has been applied to two NP-hard optimization problems arising in the design of telecommunication networks. Preliminary computational results show that solution quality is comparable to that obtained by a TS algorithm that addresses the problem as deterministic, and the increase in computation time is acceptable.

3.5 Stochastic Partitioning Methods

3.5.1 Stochastic Partitioning Methods for SCOP's

We have grouped under the name of SPM the Beam Search heuristic applied to SCOPs [16, 76], the Stochastic Branch and Bound [144, 145], and the combined procedure inspired by Nested Partitions [150]. These methods, explicitly designed for SCOPs,

follow in different ways the same search strategy: the search space is recursively partitioned in sub-spaces, and the computation effort is concentrated on the sub-spaces that are estimated to be the most promising ones. SPM are not usually considered as belonging to the class of metaheuristics, but they could, since inside the general search strategy, several heuristics may be employed for the evaluation of search sub-spaces, for the improvement of solutions, and for the estimation and comparison among solutions. In the following, we introduce the different SPM methods in the context of the type of SCOP that each method mainly focus on.

3.5.1.1 Exact objective and ad hoc approximation

The Beam Search (BS) heuristic is a heuristic strategy closely related to Branch and Bound, where the search space is recursively partitioned in sub-spaces, for which upper and lower bounds for the objective function are computed, in order to guide the search in the more promising partitions. Unlike Branch and Bound, BS reduces the width of the search moving downward in the search tree only from a limited number of best promising nodes. The success of BS depends on the evaluation function that is used to select the nodes that will be further explored. Typically, in BS different evaluation functions are used. First, a simple but imprecise evaluation function is used to discard some nodes (this phase is called *filtering*); second, nodes that survive filtering are subject to a more precise and time consuming evaluation. Thus, the main principles behind BS (partitioning the search space and dosing the computation effort in specific partitions) are similar to those of SBB and NP. The BS has been only recently applied to SCOPs. Beraldi and Ruszczyński [16] consider Chance Constrained problems like the ones we described in section 2.2. They apply BS to a set covering problem with probabilistic constraints, and show experimentally that BS allows a considerable time saving with respect to an exact Branch and Bound algorithm, and the solution quality of BS goes from optimal to 5% worse than optimal. Erel, et al. [76] present a BS-based method for the stochastic assembly line balancing problem in U-lines. Computational experiments indicate that the average performance of the proposed method is better than the best-known heuristic in the literature for the traditional straight-line problem.

3.5.1.2 Sampling approximation

Stochastic Branch and Bound (SBB) has been first proposed by Norkin, Ermoliev, and Ruszczyński [144], as a method for solving problems where the objective function must be estimated by sampling as described in Section 2.4.2. This algorithm extends to SCOPs the main principle of the classical Branch and Bound, that is, the computation of upper and lower bounds for the objective function of portions of the search space, in order to guide the search. The main difference with respect to classical Branch and Bound is that here, due to the stochastic and non-exact estimation of the objective function (and thus of the upper and lower bounds), sub-spaces cannot in general be cut during the search, but a sort of backtracking into previously evaluated sub-spaces may be necessary.

The SBB algorithm proposed in [144] is represented by the pseudocode of Algorithm 7, and works as follows. Given the search space S, the algorithm constructs increasingly finer partitions of S, denoted by $\mathcal{P} = \{S^1, S^2, \ldots\}$. The original problem of finding $g^*(S) := min_{x \in S}\{g(x)\}$ (see Equation (2.2)), is divided into the sub-problems of finding $g^*(S^r) := min_{x \in S^r}\{g(x)\}$, with $r = 1, 2, \ldots$, and $g^*(S) = min_{S^r \in \mathcal{P}}\{g^*(S^r)\}$. Assume that there exist functions L and U from $\mathcal{P}$ to $\mathbb{R}$ such that, for each $S^r \in \mathcal{P}$, $L(S^r) \leq g^*(S^r) \leq U(S^r)$, and $U(S^r) = g(\bar{x})$ for some $\bar{x} \in S^r$, and if S^r is a singleton set, then $L(S^r) = g^*(S^r) = U(S^r)$. Suppose that the lower and upper bounds $L(S^r)$ and $U(S^r)$ cannot be exactly computed, but instead estimates $\lambda^l(S^r)$ and $\nu^m(S^r)$ are used, respectively, assuming that almost surely $\lim_{l \to \infty} \lambda^l(S^r) = L(S^r)$, and $\lim_{m \to \infty} \nu^m(S^r) = U(S^r)$.

Algorithm 7 Stochastic Branch and Bound (SBB)

Set $\mathcal{P}_0 = S$, $\lambda_0(S) = \lambda^{l_0}(S)$, $\nu_0(S) = \nu^{m_0}(S)$;
for iteration $k = 0, 1, 2, \ldots$ **do**
  Select the lowest-bound subset $\bar{S}_k \in \operatorname{argmin}_{S^r \in \mathcal{P}_k}\{\lambda_k(S^r)\}$ and a current solution $x^k \in \operatorname{argmin}_{S^r \in \mathcal{P}_k}\{\nu_k(S^r)\}$;
  if the lowest-bound subset $\bar{S}_k$ is a singleton **then**
    $\mathcal{P}_{k+1} = \mathcal{P}_k$;
  else
    Construct a partition of the lowest-bound subset $\mathcal{P}'_k(\bar{S}^k) = \{\bar{S}^k_1, \bar{S}^k_2, \ldots, \bar{S}^k_{n_k}\}$;
    Construct a new full partition $\mathcal{P}_{k+1} = \mathcal{P}_k \backslash \{\bar{S}_k\} \cup \mathcal{P}'_k(\bar{S}^k)$;
  end if
  for all subsets $S^r \in \mathcal{P}_k$ **do**
    Update the estimates of lower and upper bounds $\lambda_k(S^r) = \lambda^{l_k}(S^r)$, $\nu_k(S^r) = \nu^{m_k}(S^r)$;
  end for
end for

Norkin et al. [144] proved the following convergence result: Suppose the indices l_k and m_k are chosen in such a way that whenever a subset S^r is an element of $\mathcal{P}_k$ for infinitely many k, then $\lim_{k \to \infty} l_k = \infty$ and $\lim_{k \to \infty} m_k = \infty$. Then with probability one there exists an iteration k_0 such that for all $k \geq k_0$, the lowest-bound subsets $\bar{S}_k$ are singletons and contain optimal solutions only. As suggested in [144], the estimation of a lower bound $L(S^r)$ for $g^*(S^r)$, may be done by exchanging the minimization and the expectation operator, since $g^*(S^r) = \min_{x \in S^r} g(x) = \min_{x \in S^r} \mathbb{E}_P(G(x, \omega)) \geq \mathbb{E}_P\big(\min_{x \in S^r} G(x, \omega)\big)$. Thus, one may chose $L(S^r) = \mathbb{E}_P\big(\min_{x \in S^r} G(x, \omega)\big)$, and the estimation of the lower bound $L(S^r)$ may be computed by the sample average

$$\lambda^N(S^r) = \frac{1}{N} \sum_{j=1}^{N} \min_{x \in S^r} G(x, \omega_j), \tag{3.3}$$

where $\omega_1, \omega_2, \ldots, \omega_N$ is an independent, identically distributed (i.i.d.) random sample of N realizations of the random vector ω.

In general, the practical application of SBB implies one major difficulty: computing an estimation of the lower bound by Equation (3.3) requires solving a possibly NP-hard deterministic combinatorial optimization problem, $\min_{x \in S^r} G(x, \omega_j)$, for every sample scenario ω_j, and this is unfeasible in a reasonable amount of computation time, unless very small problem instances are addressed.

Gutjahr et al. [101] use SBB to solve small instances of the single-machine-tardiness scheduling problem. They consider different sampling techniques for estimating lower bounds, and report computational experiments.

As a way to make SBB more efficient, Gutjahr et al. [104] propose to use heuristics or metaheuristics to approximately solve the deterministic subproblems for the lower bound estimation of Equation (3.3), as schematized by Figure 3.2. The authors focus on the problem of Activity Crashing in Project Management, and show experimentally that the replacement of an exact solution to deterministic subproblems by a heuristic one (in this case a local search algorithm) is very advantageous. The authors also say that it is possible to extend the convergence results of [144] to cases in which the deterministic subproblems are approximately solved by a search heuristic with a random starting point, keeping track of the best solution found so far. Another practical enhancement of the SBB proposed in [104] is the use of Importance Sampling as a technique to reduce the variance of the sample average estimates. Without the use of a variance-reduction technique, the number of Monte Carlo samples (and thus of computation time) required to obtain a sample average with the same variance would be much greater.

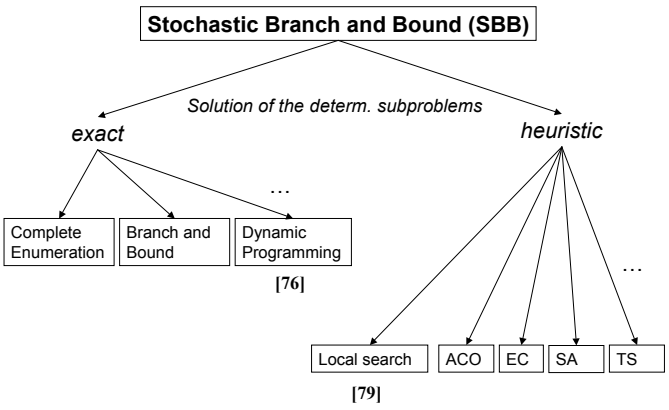


Figure 3.2: Possible ways of solving the deterministic subproblems for the computation of the lower bound (Equation (3.3)) in SBB.

Pichitlamchen and Nelson [150] propose a combined procedure extending the Nested Partitions (NP) method by Shi and Ólafsson [166] to SCOPs where the objective is estimated by sampling. NP is based on identifying a sequence of 'most promising' subsets of the search space S, and concentrating the search of good solutions there. At each iteration, the most promising subset of S is partitioned into M subsets, and the entire surrounding region is aggregated into one single subset of S. Thus, at each

iteration NP looks at a partition of $M+1$ subsets of the search space S. From each of these $M+1$ subsets, a random solution is chosen using some random sampling scheme, and the objective value of each solution is evaluated, in order to decide which is the most promising subset of the next iteration. With respect to NP, the combined procedure of Pichitlamchen and Nelson applied to SCOPs includes a number of enhancements. First, in each of the current $M+1$ subsets, more than one solution is randomly chosen for evaluating the most promising subset. Second, solutions here are evaluated by the sample average estimation the objective value (see Equation (2.13)). Moreover, in order to select the best solution of each subset, and the best solution of all subsets, a statistical procedure called Sequential Selection with Memory (SMM) is used. SMM guarantees to select the best or near-best alternative among a set of solutions with a user-specified probability. It also exploits memorized information (samples and sample averages) on previously encountered solutions. The spirit of SMM is similar to the F-Race procedure proposed in the context of ACO [45] (see Section 3.1), since it consists in a series of steps in which a set of competing solutions is evaluated and the worst of them are eliminated. For details about SMM, see also [148, 149]. The combined procedure based on NP also applies a Hill Climbing local search (HC) to the best solution of each iteration. In this way, the computational effort is concentrated on the most promising subset of the search space. Another specific characteristic of the combined procedure of Pichitlamchen and Nelson is that at the end of the algorithm, the solution having the smallest sample average accumulated over all visits to that solution is returned as final solution. Pichilamchen and Nelson call their combined procedure NP+SMM+HC, a name that underlines its main building blocks just described. In [150], they provide a proof that, with probability one, NP+SMM+HC finds one of the optimal solutions as the number of iterations goes to infinity. Moreover, numerical experiments applying the algorithm to an (s,S) Inventory Problem and to a Three-Stage Buffer allocation problem show that NP+SMM+HC has a good performance in comparison to a pure random search and to a SA algorithm. While the convergence guarantee of NP+SMM+HC is due to the global guidance system provided by NP, the practical performance is enhanced by the use of SMM selection-of-the-best method and HC local search.

3.6 Other algorithmic approaches to SCOPs

In this section we briefly give some references to other approaches for solving SCOPs, such as Progressive Hedging and Rollout Algorithms, that we have not reviewed in the previous sections because they are still not very frequent in the literature. Nevertheless, we include this section because the referenced approaches may be either classified as metaheuristics, or they may involve metaheuristics as part of their solution strategy.

Progressive Hedging (PH) is an algorithm proposed by Rockafellar and Wets [155] for solving multistage stochastic programs. It is based on considering a set of few representative scenarios that capture the uncertain future; for each of these scenarios, a deterministic optimization subproblem is solved; in this way one ends up with more solutions, neither of which is in general feasible for the original problem. Therefore,

a sort of averaging procedure among the solutions of the subproblems is performed, in order to obtain a 'blended' solution that hedges against future uncertainty. Some extensions of PH involve the use of heuristics or metaheuristics to solve the deterministic subproblems. For example, Løkketangen and Woodruff [136] integrate TS in the PH framework and apply this combined procedure to mixed integer (0,1) multistage stochastic programming. Haugen et al. [111] propose an extension of PH that is explicitly proposed as a metaheuristic: rather that using a heuristic algorithm to solve deterministic subproblems, it uses an algorithm for subproblems that is exact in its usual context, but severs as a heuristic for the proposed PH metaheuristic.

Rollout algorithms (RO) are an emerging class of methods for solving combinatorial optimization problems, that are capable of improving the performance of a given heuristic through sequential applications. Originally, the ideas underlying RO have been proposed by Tesauro and Galperin in 1997 [176] for developing a simulation-based algorithm to play blackgammon. In the same year, Bertsekas et al. [23] formalized RO for combinatorial optimization problems, by applying them to a machine maintainance and repair problem. RO are based on the Policy Iteration algorithm, which is part of the Dynamic Programming framework for solving MDP [19] (see the paragraph about Markov Decision Processes of Section 2.3). Some authors (Bertsekas et al. in Section 2 of [23], and Bertsekas on page 528 of [20]), emphasize that RO also share some ideas with Tabu Search, particularly with the sequential fan candidate list strategy (Glover and Laguna [94]) and its extended variant, the fan and filter method (Glover [92]). Among the papers that apply RO to SCOPs, we cite the works of Secomandi on the vehicle vouting problem with stochastic demands [162, 163] and on the TSP with stochastic travel times [164], and the paper by Bertsekas and Castañon [21] on stochastic scheduling.

3.7 Discussion and open issues

3.7.1 Using the Sampling approximation

We have seen that the selection-of-the-best method that a metaheuristic uses for performing sample averages and for comparing solutions can have a great impact on the effectiveness of the algorithm, but it is still hard to say which method is the most effective in relation to the metaheuristic where it is employed, and this is an interesting open issue.

Table 3.2 reports some successful selection-of-the-best methods described in the previous sections in the context of the metaheuristic where they have been used. In some cases ([99, 102, 7, 6, 115]), the use of a particular method has been justified mainly by the need to derive rigorous properties of convergence, and the application to other metaheuristics is not very meaningful. But in more experimental oriented papers, a method which is particularly efficient in one metaheuristic, could be advantageous also in others. This is the case, for instance, of F-Race [45], SMM [150], and the adaptive sampling procedures used in [100], [116], and [65]. In looking for efficient selection-of-the-best methods to be applied in metaheuristics, the literature about statistical

procedures of ranking, selection, and multiple comparisons (see, for example, [80, 173] and the references cited therein) could be a good source of inspiration. Moreover, for speeding up the sample average computations, it could be useful the application of variance-reduction techniques, such as, for example, those belonging to the field of Rare Event Simulation [160].

Given the above observations, a selection-of-the-best method working as a black box simulation that does not allow to specify how samples are chosen is not advisable. Another requirement that seems necessary is the possibility to increase the accuracy of objective function estimates, particularly when the algorithm has identified good or near optimal solutions. The intuitive reason is that often in SCOPs there are many local optima, whose values may be also quite near, and in order to discriminate between local optima one needs that the estimation error is small with respect to the difference between the exact value of local optima. We have seen a practical confirmation of this in several experimental papers, for instance [181] and [65]. A more rigorous argument in favor of this requirement is that all metaheuristics with provable convergence properties need to use a number of samples increasing with the iteration counter.

It has been recognized that completely different discrete stochastic optimization algorithm may be needed for small and for large search spaces, respectively (cf. [80]). Also the "degree of randomness", that is, the size of noise compared to the undisturbed objective function values, is an important factor. It cannot be expected that a metaheuristic variant working well for solution spaces with a small amount of noise will also perform optimally for solution spaces with a large amount of noise, and vice versa. It appears that a characterization of metaheuristic variants for SCOPs with respect to their appropriate domains of problem instance types still waits for being elaborated. In the second part of this thesis, we will address this particular aspect in the context of the PTSP, by considering a benchmark of PTSP instances with varying levels of stochasticity, and by analyzing the behavior of our developed metaheuristics for different type of instances.

3.7.2 Experimental comparisons among different metaheuristics

At the moment, most of the papers in the SCOP literature focus on one single metaheuristic, which is compared either to variants of the same metaheuristic, or to simple heuristics such as random search, or to exact methods when these are available. Only a very small number of papers perform comparisons among different metaheuristics, as reported in Table 3.3. Thus, it is still impossible to give guidelines on which metaheuristic is better in which situation.

One important aspect that experimentation with different metaheuristics could reveal is whether the effectiveness of a metaheuristic is due to the particular adjustments to speed up the computation (like approximating the objective function or using carefully designed sampling and statistical comparison strategies), or to the intrinsic search trajectory of the metaheuristic. This issue is addressed in Appendix B in the context of the Vehicle Routing Problem with Stochastic Demands (VRPSD).

Reference(s)	Selection-of-the-best method		Metaheuristic(s) where the method is used
	Way of evaluating a solution	Solutions compared	
Gutjahr [99]	Sample average with number of samples increasing linearly with the iteration counter	Current solution with current estimation of optimal solution	ACO
Gutjahr [100]	Sample average with number of samples decided adaptively on the base of a statistical test	Current solution with current estimation of optimal solution	ACO, SA
Birattari et al. [45]	Sample averages integrated with the F-Race statistical procedure	All solutions belonging to the (dynamic) set of solutions in the race	ACO
Gutjahr and Pflug [102]	Sample average with number of samples increasing more than quadratically with the iteration counter	Current solution with current estimation of optimal solution	SA
Alrefaei and Andradóttir [7]	Sample average and normalized number of visits	All solutions visited so far	SA
Alkhamis et al. [6]	Sample average with number of samples increasing with iterations, comparison with a statistical significance test	Current solution with current estimation of optimal solution	SA
Homem-de-Mello [115, 116]	Sample average with number of samples decided adaptively on the base of a t-test	Current solution with current estimation of optimal solution	SA
Costa and Silver [65]	Descriptive sampling, statistical test in two stages (using a higher number of samples only if first stage of the test is positive)	Current solution with current estimation of optimal solution, keeping in memory a given number of good solutions for final, more accurate comparison	TS
Gutjahr et al. [101]	Sample average using the Importance Sampling variance-reduction technique, and number of samples increasing with the iteration counter	Lower and upper bounds of all subsets in which the search space has been partitioned	SBB (SPM)
Pichitlamchen and Nelson [150]	Sample averages integrated with the SMM statistical procedure	Random selected solutions in each subset in which the search space has been partitioned, and random solutions selected from the neighborhood of the current solution during local search	NP+SMM+HC (SPM)

Table 3.2: Main selection-of-the-best methods used in the metaheuristics literature for SCOPs where the objective function must be estimated by sampling.

Reference	SCOP	Metaheuristics compared	"Winner"
Bianchi et al. [32, 33]	VRPSD	ACO, EC, SA, TS, ILS	EC, TS
Gutjahr [100]	TSPTW	ACO, SA	ACO
Pichitlamken and Nelson [150]	Inventory Problem and Three-stage Buffer Allocation Problem	SPM, SA	SPM
Easton and Mansour [74]	Stochastic Goal Programming	EC, SA, TS	EC

Table 3.3: Papers with comparisons among different metaheuristics.

3.7.3 Theoretical convergence properties

Papers analyzing theoretical convergence properties of metaheuristics applied to SCOPs are summarized in Table 3.4. Note that in the table, SCOPs with exactly computable objective function are missing. In fact, when a metaheuristic always uses an exact expression for the objective value of a solution, its convergence behavior is equivalent, from a theoretical point of view, to that of applying the metaheuristic to a DCOP (pointers to theoretical analyses of metaheuristics for DCOPs have been provided in the previous sections while introducing each metaheuristic). On the contrary, when ad hoc approximations of the objective function are used, the fact that the error is systematic makes a theoretical analysis very difficult.

Theoretical convergence analyses do exist for static SCOPs with sampling estimated objective function. The most studied metaheuristic from this point of view is certainly SA, followed by ACO and SPM. TS and EC still miss this kind of analysis. Interstingly, Homem-de-Mello [116] suggests that the results he derives for a simple variable-sample random search algorithm can be readily adapted to show the convergence of variable-sample versions of more sophisticated methods, in particular, those methods for which the proof of convergence in the DCOP domain relies on the convergence of pure random search. The author indicates explicitly EC (GA) as one of these, by referring to the work of Rudolph [161].

Finally, metaheuristics with provable convergence properties (ACO and EC) have been designed to solve MDPs. Actually, at the moment MDPs have been addressed *only* theoretically by metaheuristics. Therefore, there is the need to validate their effectiveness also by experimental investigations.

Metaheuristic	SCOP category	Referece(s)
ACO	sampling estimated objective	Gutjahr [99]
SA	“ ”	Alrefaei and Andradóttir [7]
SA	“ ”	Alkhamis et al. [6]
SA	“ ”	Homem-de-Mello [115]
SA	“ ”	Alkhamis and Ahmed [5]
SBB (SPM)	sampling estimated objective	Norkin et al. [144]
SA	objective function subject to normally distributed noise	Gelfand and Mitter [84]
SA	objective function subject to noise reducing to zero after a certain number of iterations	Fox and Heine [79]
SA	objective function subject to noise distributed according to a sufficiently ‘peaked’ distribution	Gutjahr and Pflug [102]
ACO	infinite horizon MDP	Chang et al. [60] and Chang [59]
EC	infinite horizon MDP	Chang et al. [61]
EC	finite horizon partially observed MDP	Lin et al. [135]

Table 3.4: Papers with theoretical convergence proofs.

Part II

Ant Colony Optimization and local search for the Probabilistic Traveling Salesman Problem

Chapter 4

The Probabilistic Traveling Salesman Problem

The first part of the thesis (up to Chapter 3) had a broad scope, being dedicated to the review of the application of several metaheuristics to the wide class of SCOPs. This second part of the thesis (from the present chapter to Chapter 7) focuses in particular on one metaheuristic, ACO (Ant Colony Optimization), and on one SCOP, the Probabilistic Traveling Salesman Problem (PTSP). Concentrating on this case study gives us the occasion to explore in a more detailed way many issues that have been introduced in the first part of the thesis, such as the use and effectiveness of objective function approximations inside ACO (in Chapter 5), the design of efficient local search algorithms also based on move cost approximations (in Chapter 6), and the effect of combining a metaheuristic with local search for solving the SCOP under study (in Chapter 7).

The remainder of this chapter is organized as follows. Section 4.1 formally defines the PTSP, by introducing the notation used throughout the second part of the thesis, and by defining and explaining the PTSP objective function. Section 4.2 reports on the literature involving the PTSP. Some papers have been already cited in the first part of the thesis, but they are again considered here for the sake of completeness. Section 4.3 describes the benchmark of PTSP instances that we have set up for our experimental investigations of the following chapters. The benchmark has been carefully designed, in particular to allow the analysis of the behavior of optimization algorithms for different levels of stochasticity. Section 4.4 and 4.5 describe, respectively, the computation of a lower bound of the optimal solution for the PTSP, and a set of simple heuristics available in the literature to solve the PTSP. These elements will be used for producing repeatable comparisons with our metaheuristics.

4.1 Statement of the problem

For many delivery companies, only a subset of the customers require a pickup or delivery each day. Information may be not available far enough in advance to create optimal schedules each day for those customers that do require a visit or the cost to acquire

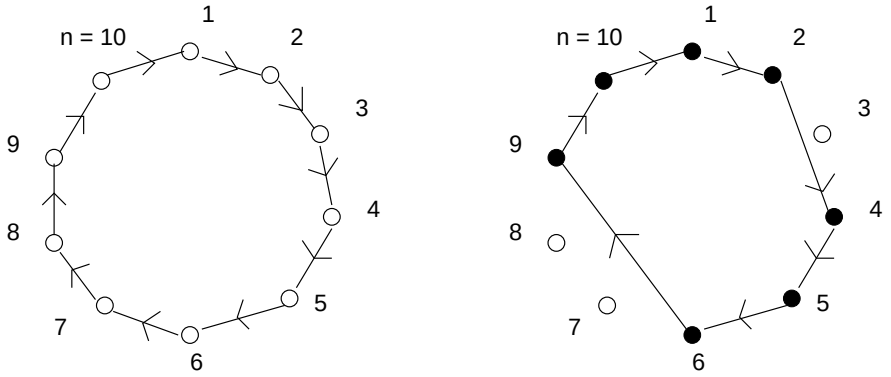


Figure 4.1: An a priori tour (left), and a possible a posteriori tour (right) obtained from the a priori tour by visiting only the customers requiring a visit on a given random realization of customers, and by keeping the same order of visit as in the a priori tour. In this example, a random realization where only customers number 1, 2, 4, 5, 6, 9, 10 require a visits is shown.

sufficient computational power to find such solutions may be prohibitive. For these reasons, it is not unusual to design a tour containing all customers (called *a priori* tour), and each day to follow the ordering of this a priori tour to visit only the customers requiring a visit that day (see Figure 4.1). The tour actually traveled each day when the customers requiring a visit are revealed is called *a posteriori* tour. The problem of finding an a priori tour of minimum expected cost, given a set of customers each with a given probability of requiring a visit, defines the PTSP. In the remainder of the thesis, the terms a priori tour tour, and solution, will be used interchangeably.

More formally, let $N = \{i \mid i = 1, 2, \ldots, n\}$ be a set of n customers. For each pair of customers $i, j \in N$, $d(i, j)$ represents the distance between i and j. Here, we assume that the distances are symmetric, that is, $d(i, j) = d(j, i)$. In the remainder of the thesis, distances will also be referred to as costs. An a priori tour $\lambda = (\lambda(1), \lambda(2), \ldots, \lambda(n))$ is a permutation over N, that is, a tour visiting all customers exactly once. Given the independent probability p_i that customer i requires a visit, $q_i = 1 - p_i$ is the probability that i does not require a visit. The general case where customers probabilities p_i may be different, is referred to as *heterogeneous* PTSP, while if probabilities are all equal ($p_i = p$ for every customer i), the problem is called *homogeneous* PTSP. We will use the following convention for any customer index i:

$$i := \begin{cases} i(\mathrm{mod} n) & \text{iff } i \neq 0 \text{ and } i \neq n \\ n & \text{otherwise,} \end{cases} \tag{4.1}$$

where $i(\mathrm{mod}\, n)$ is the remainder of the division of i by n. The reason for defining the above convention is that we want to use as customer indices numbers from 1 to n (and not from 0 to $n-1$), and we need to make computations with the 'modulus' operator. The expected length of an a priori tour λ can be computed in $O(n^2)$ time with the

following expression derived by Jaillet [118]

$$E[L(\lambda)] = \sum_{i=1}^{n} \sum_{r=1}^{n-1} d(\lambda(i), \lambda(i+r)) p_{\lambda(i)} p_{\lambda(i+r)} \prod_{i+1}^{i+r-1} q_{\lambda}. \tag{4.2}$$

We use the following notation for any $i, j \in \{1, 2, \ldots, n\}$

$$\prod_{i}^{j} q_{\lambda} := \begin{cases} \prod_{t=i}^{j} q_{\lambda(t)} & \text{if } 0 \leq j - i < n - 1 \\ \prod_{t=i}^{n} q_{\lambda(t)} \prod_{u=1}^{j} q_{\lambda(u)} & \text{if } i - j > 1 \\ 1 & \text{otherwise.} \end{cases} \tag{4.3}$$

The expression for the objective function (Equation (4.2)) has the following intuitive explanation: each term in the summation represents the distance between the i^{th} customer and the $(i+r)^{\text{th}}$ customer weighted by the probability that the two customers require a visit $(p_{\lambda(i)} p_{\lambda(i+r)})$ while the $r-1$ customers between them do not require a visit $(\prod_{i+1}^{i+r-1} q_{\lambda})$.

In the homogeneous PTSP, where $p_i = p$ and $q_i = q$ for every customer i, the expression for the objective function still requires $O(n^2)$ computation time, but it is a bit simpler than Equation (4.2):

$$E[L(\lambda)] = \sum_{i=1}^{n} \sum_{r=1}^{n-1} p^2 q^{r-1} d(\lambda(i), \lambda(i+r)). \tag{4.4}$$

Note that the PTSP falls in the category of SIPs (Stochastic Integer Programs) as introduced by Definition 2 in Chapter 2.

4.2 Literature review

The PTSP was introduced in 1985 by Jaillet in his PhD thesis [118], where he derives several theoretical properties of the problem and proposes different heuristics as well as an integer nonlinear programming formulation and an exact branch-and-bound algorithm for the homogeneous PTSP (the exact algorithms remain of an introductory level, since they are not tested experimentally). Some theoretical properties of the PTSP derived in [118] have been later published in Jaillet [119], and focus mainly on two issues: the derivation of closed form expressions for the efficient computation of the objective function under several customers probability configurations, and the analysis of properties of optimal PTSP solutions, especially in relation to optimal solutions of (deterministic) TSP. The analysis presented in [119] implies that under specific conditions (the set of customers on the Euclidean space is also the convex hull of this set of customers) the optimal PTSP and TSP solutions coincide. In general though, bounds have been derived that show how the optimal TSP solution may be arbitrarily bad with respect to the optimal PTSP solution. This observation justified and triggered subsequent research on developing specific algorithms for the PTSP that take into account its probabilistic nature.

Bertsimas et al. [28] showed that the PTSP is an NP-hard problem. As such, the PTSP is very difficult to be solved to optimality, and its literature is more rich in papers on heuristics rather than on exact methods.

Rossi and Gavioli [159] adapt to the PTSP two well known TSP tour construction heuristics, the Nearest Neighbor and the Clarke-Wright algorithms, by explicitly including the customers probability in the evaluation and selection of new portions of the tour to construct. Rossi and Gavioli experimentally evaluate the two heuristics on homogeneous PTSP instances with up to 100 uniformly distributed customers, and compare their performance to that of the classical deterministic Nearest Neighbor and Clarke-Wright heuristics that solve a TSP. Their conclusion is that for the tested instance it is important to use the techniques specifically designed for the PTSP only for instances with more than 50 customers, and customers probability less than 0.6.

The PhD thesis of Bertsimas [24] and a related article by Bertsimas and Howell [27] extend the results of Jaillet sharpening the bounds that link the PTSP with the TSP, and analyze the relation of the PTSP with another probabilistic combinatorial optimization problem, the probabilistic minimum spanning tree. In [27], the authors also investigate the worst-case and average performance of some TSP heuristics in a PTSP context. The analysis reveals that the Nearest Neighbour is very poor on average for the PTSP, while the Space Filling Curve is asymptotically very close to the re-optimization strategy, if nodes are uniformly distributed on the unit square and an Euclidean metric is used for customer distances. The Space Filling Curve is analyzed also experimentally in [27]. Two local search procedures, the 2-p-opt and the 1-shift, are applied to the solution produced by the Space Filling curve, to achieve better solutions for the PTSP. For the two local search procedures, recursive equations are proposed, that efficiently compute the change of the homogeneous PTSP expected cost of two neighboring solutions. We have verified that the recursive equations proposed in [27] are not correct, and we derive in this thesis correct expressions both for the homogeneous and for the heterogeneous case (see Section 6.4, that covers the contents of Bianchi et al. [42] and Bianchi and Campbell [36]).

The results and analyses of [24] are summarized and further extended in Bertsimas et al. [28], where also the probabilistic vehicle routing problem and the probabilistic traveling salesman facility location problem are investigated with the paradigm of a priori optimization. In [28], only problems with homogeneous probabilities are considered.

In [26], Bertsimas et al. investigate experimentally the PTSP and the Probabilistic Vehicle Routing Problem, by comparing the solution quality obtained by solving the problem a priori and a posteriori (that is, by sampling the stochastic variables and by approximately solving the deterministic subproblems obtained by sampling). For the PTSP, the authors consider both homogeneous and heterogeneous instances. Experimental results on small instances with up to 50 customers uniformly distributed on the unit square show that a priori solutions are on average only 1.22% worse than a posteriori ones obtained via re-optimization of the sampled subproblems. These experiments confirm the theoretical results of Bertsimas and Howell [27] on the similarity between a priori optimization and re-optimization when solving uniformly distributed

PTSP instances.

Exact algorithms for solving also heterogeneous PTSPs to optimality are addressed in two papers: Berman and Simchi-Levi [17], and Laporte et al. [132]. In [17], the considered problem is a PTSP on a network, that is, the underlying graph is not assumed to be complete. The key result is the derivation of a good lower bound for the optimal PTSP value, that is based on solving an appropriate transportation problem with $2n$ constraints. It is suggested that the lower bound can be used in a Branch and Bound algorithm to find the optimal solution. The authors of [17] refer to a working paper [18] for a first attempt to implement such Branch and Bound algorithm. Another result of [17] is the derivation of an algorithm that, given an a priori tour, finds in $O(n^3)$ time the optimal location of a hypothetical depot from where the traveling salesman departs, before traveling the a priori tour.

The paper by Laporte et al. [132] proposes a Branch and Cut algorithm based on an integer two-stage stochastic programming formulation of the PTSP (for the definition of this type of problems, see Section 2.2). The authors report on computational experiments on both homogeneous and heterogeneous PTSP instances, and show that the proposed algorithm finds the optimal solution for instances with up to 50 customers. An interesting point that emerges from the experiments is that the lower the customers probabilities, the more difficult the PTSP instance. In other words, "more random" problems are more difficult to solve by the proposed Branch and Cut algorithm. Unfortunately, [132] does not report the optimal values found for the tested instances, and their results cannot be used to compare the performance of heuristic approaches.

Recent approaches to the PTSP mainly involve the application of metaheuristics. These include Simulated Annealing (SA) [50], Evolutionary Computation [158], and Ant Colony Optimization (ACO) [51, 52, 100, 45, 38, 39].

Bowler et al. [50] analyze experimentally by a stochastic SA algorithm the asymptotic behavior of (sub) optimal homogeneous PTSP solutions, in the limit of pn (customers probability times number of customers) going to infinity. The PTSP objective function is estimated by sampling, and the sampling estimation error is used as a sort of temperature regulated during the annealing process.

The first preliminary results about solving the PTSP with ACO have been published by Bianchi et al. [38, 39], and are the base of the results described also in this thesis. The main issue addressed in [38, 39] is the comparison of a TSP specific version of ACO with respect to a version that explicitly considers the PTSP objective function, similarly to what has been done by Rossi and Gavioli [159] on the Nearest Neighbor and Clarke-Wright heuristics.

Branke and Guntsch [51, 52] explore the idea to employ faster approximations of the exact PTSP objective function. The authors propose an ad-hoc approximation of the expected cost that neglects the least probable customers configurations. This approximation is shown experimentally to accelerate convergence without significantly worsening the solution quality. Another issue addressed by [51, 52] is the design of PTSP-specific heuristics to guide the ants construction process. The authors experimentally analyze different heuristics, and show that one of them indeed improves the quality of solution constructed by ants, but at the cost of a higher computational time.

name	n
kroA100	100
eil101	101
ch150	150
d198	198
lin318	318
att532	532
rat783	783
dsj1000	1000

Table 4.1: TSPLIB instances providing customers coordinates, from which Euclidean distances have been computed. Each of these TSP instances has been combined with 54 different customer probability configurations characterized by average probability and variance as reported in Table 4.2.

Gutjahr [100] briefly addresses the PTSP as a pre-test for analyzing the performance of S-ACO, an ACO algorithm that is suited for stochastic problems where the objective function is not known exactly, but can only be estimated by sampling. After the pre-test on the PTSP, S-ACO is applied to a TSP with time windows and with stochastic travel times.

Recently (Birattari et al. [45]), the PTSP has been again used as a sort of test problem for ACO/F-Race, another ACO algorithm suited for stochastic problems where the objective function must be estimated by sampling.

More details on how S-ACO [100] and ACO/F-Race [45] work can be found in Section 3.1.

4.3 Benchmark of PTSP instances

A PTSP instance is characterized by two types of information: distances between each couple of customers, and probability for each customer of requiring a visit. Since customer distances are what characterize instances of the TSP, we have used instances from the well known TSPLIB benchmark [178] for this type of information. For each TSP instance, we have then considered different customer probability configurations. Homogeneous PTSP instances have been generated by simply associating a TSP instance one single probability value, corresponding to all customers probabilities. For heterogeneous PTSP instances different sets of customer probability values have been randomly generated according to a probability distribution.

We have considered 8 TSP instances, with n ranging from 100 to 1000, as reported in Table 4.1. For these instances we have computed Euclidean distances between customers, on the base of the customers coordinates provided by the TSPLIB.

Customers probabilities have been generated as follows. For homogeneous PTSP instances the customer probability has been varied from 0.1 to 0.9 with an increment of 0.1. For heterogeneous PTSP instances, each customer probability p_i, $i = 1, 2, \ldots, n$

has been generated randomly, according to the 'beta probability distribution'. We have chosen this probability distribution because it is defined on the finite interval $[0, 1]$, and it can easily model different average and variance of customers probability values by choosing the appropriate parameters. The 'beta probability distribution' $\beta_{a,b}(p_i)$ is defined as follows

$$\beta_{a,b}(p_i) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} p_i^{a-1}(1-p_i)^{b-1}, \tag{4.5}$$

where $p_i \in [0, 1]$, and $a, b > 0$ are parameters. The a and b parameters of the 'beta probability distribution' determine the average customer probability p and the variance σ^2 around the average value:

$$p = \frac{a}{a+b} \tag{4.6}$$

$$\sigma^2 = \frac{ab}{(a+b)^2(a+b+1)}. \tag{4.7}$$

In our benchmark we have generated different sets of customer probabilities by varying the average customer probability p and the variance σ^2. In order to have direct control on these parameters, rather than on a and b, we have used the inverse of Equation (4.6) and (4.7):

$$a = \frac{p\,[p(1-p) - \sigma^2]}{\sigma^2}, \tag{4.8}$$

$$b = \frac{(1-p)\,[p(1-p) - \sigma^2]}{\sigma^2}. \tag{4.9}$$

Note that the variance is limited by the relation $\sigma^2 < p(1-p) \leq 0.25$. Similarly to the homogeneous case, we have considered customer probability sets by varying p between 0.1 and 0.9, with and increment of 0.1. For every value of p, we have considered five different values of variance, corresponding respectively to 1/6, 2/6, 3/6, 4/6, and 5/6 of the maximum variance which is $p(1-p)$. More precisely, we have set $\sigma^2 \in \{p(1-p)/6, 2p(1-p)/6, 3p(1-p)/6, 4p(1-p)/6, 5p(1-p)/6\}$. For each probability, the different σ^2 values are also referred to as the 16%, 33%, 50%, 66%, and 83% of the maximum variance. In the plots that will be presented in the remainder of this thesis, the variance values will be indicated by percentage. The shape of the corresponding 'beta probability distributions' are shown in Figure 4.2.

Summarizing, for each of the 8 TSP instances, we have considered 54 customer probability configurations, which in total means 432 PTSP instances. Table 4.2 reports all the considered values of p and σ^2, including the ones corresponding to homogeneous instances (for which $\sigma^2 = 0$). The naming convention that we use throughout this thesis is illustrated in Table 4.3.

4.4 Lower bound of the optimal solution value

In evaluating the solution quality of algorithms for the PTSP, one faces the difficulty that the optimal solution of problems is not know, in general. In fact, unlike the

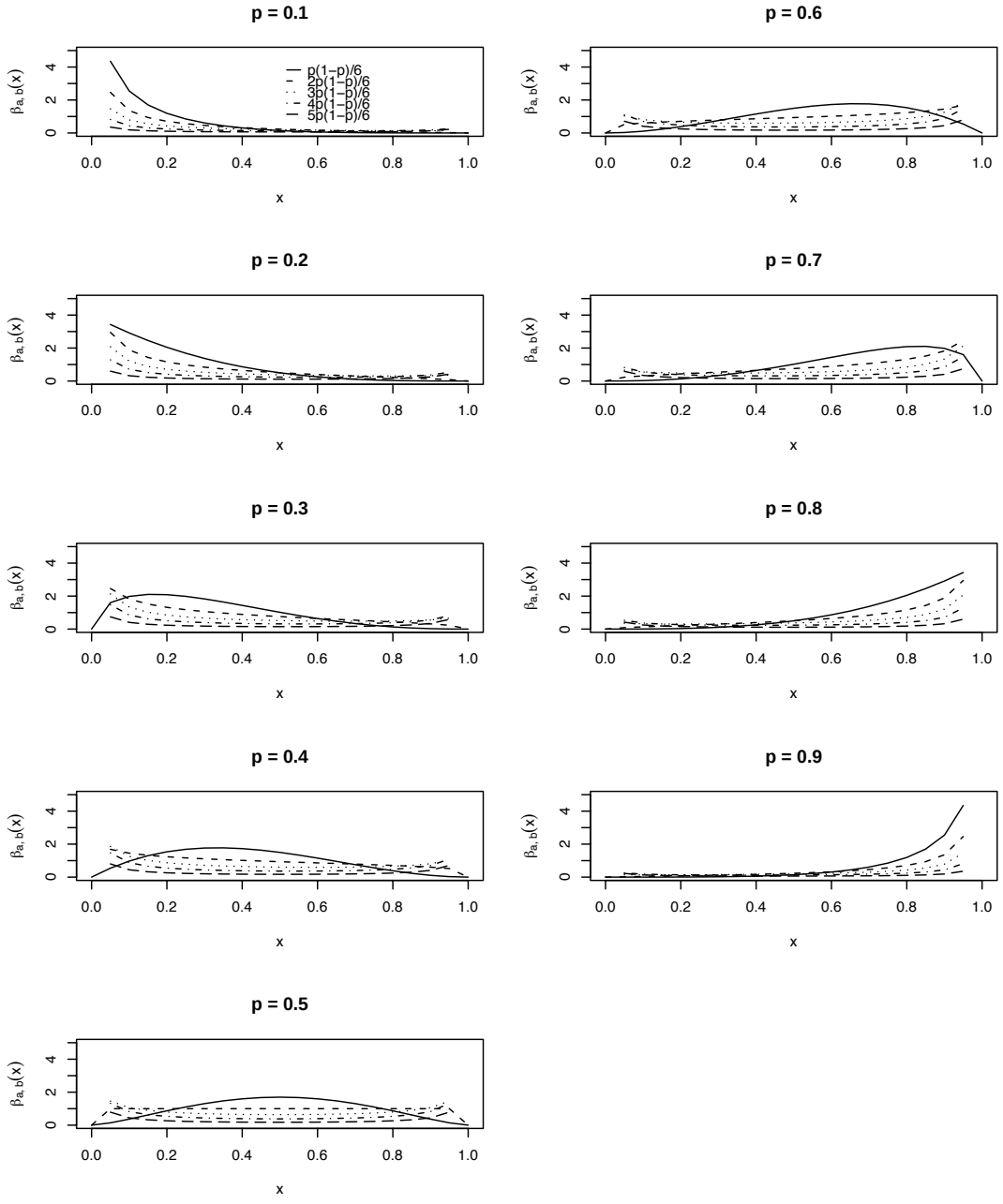


Figure 4.2: Shape of the 'beta probability distributions' used for generating customers probabilities in heterogeneous instances of our PTSP benchmark. The legend in the first plot specifies the type of line used for different values of σ^2, and applies to all plots of the figure.

p	σ^2					
0.1	0	0.015	0.030	0.045	0.060	0.075
0.2	0	0.027	0.053	0.080	0.107	0.133
0.3	0	0.035	0.070	0.105	0.140	0.175
0.4	0	0.040	0.080	0.120	0.160	0.200
0.5	0	0.042	0.083	0.125	0.167	0.208
0.6	0	0.040	0.080	0.120	0.160	0.200
0.7	0	0.035	0.070	0.105	0.140	0.175
0.8	0	0.027	0.053	0.080	0.107	0.133
0.9	0	0.015	0.030	0.045	0.060	0.075

Table 4.2: Average customer probability p and variance σ^2 characterize the 54 customer probability configurations generated for our PTSP benchmark. Homogeneous PTSP instances correspond to the column with $\sigma^2 = 0$. Heterogeneous probability configurations have been obtained for each TSP instance of Table 4.1 by using the above values of p and $\sigma^2(\neq 0)$ for computing a and b parameters (Equation (4.8)-(4.9)), and by generating sets of random customer probabilities according the 'beta probability distribution' (Equation (4.5)).

PTSP instance name	TSP instance name	n	p	σ^2
kroA100.p10.v16	kroA100	100	0.1	$p(1-p)/6 = 0.015$
kroA100.p20.v33	kroA100	100	0.2	$2p(1-p)/6 = 0.053$
kroA100.p10	kroA100	100	0.1	0

Table 4.3: Three examples illustrating the naming convention we use for instances belonging to our PTSP benchmark, where n is the number of customers, p is the (average) customers probability, and σ^2 is the variance of customers probability.

TSP, no benchmark of problems exists for which optimal solutions are known. When possible, a good alternative is to compare the solution quality of an algorithm with a known lower bound of the optimal solution, which yields to an upper bound of the distance from the optimal solution. More precisely, suppose that, for a given instance of an optimization problem, the optimal solution value is E^*, and a lower bound LB is available, such that, by definition

$$LB \leq E^*. \tag{4.10}$$

If the solution obtained by an algorithm that we want to analyze is E, we can bound the relative error of the algorithm by the following inequality

$$\frac{E - E^*}{E^*} \leq \frac{E - LB}{LB}. \tag{4.11}$$

For the PTSP, we rely on the following lower bound of the optimal tour value $E[L(\lambda_{PTSP})]$, proposed by Bertsimas, Howell [27]

$$z^* \leq E[L(\lambda_{PTSP})] \tag{4.12}$$

where z^* is the optimal solution to the (linear) transportation problem

$$\begin{aligned} z^* = \min & \sum_{i,j \in V, i \neq j} x_{i,j} d(i,j), \\ \text{s.t.} & \sum_{i \in V, i \neq j} x_{i,j} = p_j \left(1 - \prod_{k \neq j} q_k\right), \\ & \sum_{j \in V, j \neq i} x_{i,j} = p_i \left(1 - \prod_{k \neq i} q_k\right), \\ & x_{i,j} \geq 0. \end{aligned} \tag{4.13}$$

The values of the lower bound computed by solving this transportation problem for all the instances of our PTSP benchmark are shown in Table 4.5.

4.5 Simple constructive heuristics

Given that the solution structure of the PTSP is equal to that of the TSP (a Hamiltonian tour), all constructive heuristics for the TSP can also be used to construct solutions for the PTSP. Of course, we do not expect that such heuristics find near optimal solutions for the PTSP, mainly because they do not take into account customers probabilities in their construction mechanism. Nevertheless, there are at least two reasons why considering simple TSP constructive heuristics is useful:

- in developing sophisticated algorithms based on metaheuristics and local search for solving the PTSP, the solution quality provided by simple TSP constructive heuristics is a sort of minimal requirement for the quality of the more sophisticated algorithms;

- when a local search algorithm is available for a problem, it might be the case that even when applied to quite poor starting solutions such those produced by simple TSP constructive heuristics evaluated with the PTSP objective function, the final solution found is very good.

In this thesis, we consider metaheuristics (ACO) and sophisticated local search algorithms for solving the PTSP, therefore the above motivations are very relevant to us. The algorithms described and analyzed in Chapter 5, 6 and 7 will be often compared with selected TSP constructive heuristics, namely the Space Filling Curve, the Radial Sort, the Farthest Insertion, the Nearest Neighbor, and the Random Search heuristic.

The Space Filling Curve heuristic The Space Filling Curve heuristic (SFC) for the TSP was introduced by Bartholdi and Platzman [13], and the main idea behind it is the following. Let $C = \{\theta | 0 \leq \theta < 1\}$ denote the unit circle, so that $\theta \in C$ represents a point on the circle from a fixed reference point. Also let $S = \{(x, y) | 0 \leq x \leq 1, 0 \leq x \leq 1\}$ denote the unit square. A continuous mapping ψ from C onto S is known as 'spacefilling curve' [2]. Suppose that ψ is such that $\lim_{\theta \to 1} \psi(\theta) = \psi(0)$, so that, as θ ranges from 0 to 1, $\psi(\theta)$ traces out a tour of all the points in S. Given n points in S to be visited (like in the TSP and PTSP on the unit square), the idea is to visit them in the same order as they appear along the spacefilling curve, or, more mathematically, to sort the customers according to their inverse image under ψ.

In order to be a good heuristic for the TSP, the spacefilling curve ψ must be chosen in such a way that its inverse function is fast to be computed, and it satisfies some geometrical properties. Details about SFC are described in [13], where also a BASIC code of the algorithm is reported. SFC constructs a solution in $O(n \log n)$ computation time.

SFC has been analyzed in [27] in the context of the PTSP, where it is shown that it is asymptotically very close to the re-optimization strategy, if nodes are uniformly distributed on the unit square and an Euclidean metric is used for customer distances. In [27], SFC has been also used as starting solution of two local search algorithms, that will be compared to our algorithms and described in Chapter 6.

The Radial Sort heuristic Radial Sort (RAD) builds a tour by sorting customers by angle with respect to the 'center of mass' of the customer spatial distribution. More precisely, the center of mass is a point whose coordinates $(x_{\text{CM}}, y_{\text{CM}})$ are computed by averaging over the customers coordinates:

$$x_{\text{CM}} = \frac{1}{n} \sum_{i=1}^{n} x_i, \quad y_{\text{CM}} = \frac{1}{n} \sum_{i=1}^{n} y_i. \tag{4.14}$$

The angular position θ_i of customer i with respect to the center of mass is computed as follows

$$\theta_i = \arctan\left(\frac{x_i - x_{\text{CM}}}{y_i - y_{\text{CM}}}\right). \tag{4.15}$$

The tour returned by Radial Sort visits customers in order of increasing angle θ_i.

The Farthest Insertion heuristic Farthest Insertion (FI) [125] builds a tour by starting with a subtour on a small number of nodes, and by extending the subtour by inserting in the cheapest position the remaining nodes one after the other, choosing each time the node which is farthest from the subtour.

In our implementation, the starting subtour is made by a single node (the one labeled by 0), and the node to be selected for insertion at each iteration is the node which is farthest from the last inserted one.

The Nearest Neighbor heuristic Nearest Neighbor (NN) constructs a tour by choosing as next customer the nearest, not yet visited customer. Our implementation of NN constructs n tours starting each with a different customer, and returns the tour with the smallest expected length, evaluated with the PTSP objective function.

An adapted version of this heuristic which takes into account customers probabilities has been analyzed in [159]. In [27], it has been shown that NN is very poor on average for the PTSP with customers uniformly distributed on the unit square and an Euclidean metric is used for customer distances.

The Random Search heuristic Random Search (RS) is maybe the simplest one. It builds a solution starting from a random customer, and choosing iteratively as next customer one at random among the not yet visited ones. In comparing this heuristic with more sophisticated algorithms, a meaningful choice is to make RS run for the same computation time as the comparing algorithms.

4.5.1 Experimental analysis

We present here results obtained on the PTSP benchmark by the simple TSP constructive heuristics described above. These results will be also used in the following Sections of this thesis, as a term of comparisons with our ACO metaheuristics. Experiments have been run on a machine with two processors Intel(R) Xeon(TM) CPU 1.70 GHz, running the GNU/Linux Debian 2.4.27 operating system. All algorithms have been coded in C++ under the same development framework. Each heuristic was allowed a runtime equal to $n^2/100$ (with n being the number of customers of a PTSP instance).

Table 4.4 reports average results over all the PTSP benchmark. The solution quality achieved by a heuristic is expressed in terms of percentage of the best value found above the value of the lower bound (second column of the Table). More precisely, for each heuristic *heur* and for each PTSP instance *inst*, the percentage above the lower bound has been computed as:

$$\%\text{ above } LB(inst) = \frac{E[L(\lambda_{heur,inst})] - LB(inst)}{LB(inst)}, \tag{4.16}$$

where $E[L(\lambda_{heur,inst})]$ is the expected cost (that is, the PTSP objective value) of the best solution found by heuristic *heur* on the PTSP instance *inst*. The second column of Table 4.4 shows the average percentage above the lower bound over all PTSP instances,

heuristic	% above LB	time (seconds)
FI	90.2%	0.1
SFC	111.1%	0.1
NN	111.6%	35.5
RAD	388.8%	0.1
RS	1228.6%	2712.0

Table 4.4: Average results of the heuristics for the TSP applied to the PTSP benchmark.

while the third column reports the average time required by each heuristic to find the best solution. Note that comparisons with respect to the lower bound are useful for having an objective and repeatible way of evaluating an algorithm for the PTSP, but they do not indicate the amount by which an algorithm really deviates from optimal solutions. The reason is that we do not know how near is the lower bound to optimal solutions.

As expected, RS is the worst one, both in terms of solution quality and time. All the other heuristics were quite fast, with NN being the slowest. The reason why NN is so slow with respect to other heuristics (except RS), is that in our implementation of NN we have generated n different solutions considering each time a different starting customer, and all these n solutions have been evaluated with the PTSP objective function. On the contrary, in the other simple heuristics (except RS), the algorithm generated just one single solution.

Instance name	LB value	Instance name	LB value	Instance name	LB value	Instance name	LB value	Instance name	LB value	Instance name	LB value
kroA100.p10.v16	3046.08	d198.p40.v66	7351.09	dsj1000.p80.v33	12785590.00	ch150.p10.v16	880.64	att532.p40.v66	43216.89	eil101.p80.v33	474.31
kroA100.p10.v33	3749.19	d198.p40.v83	6176.07	dsj1000.p80.v50	12903120.00	ch150.p10.v33	975.37	att532.p40.v83	42008.26	eil101.p80.v50	489.49
kroA100.p10.v50	3602.53	d198.p50.v16	6151.70	dsj1000.p80.v66	13104130.00	ch150.p10.v50	1439.81	att532.p50.v16	41615.19	eil101.p80.v66	505.13
kroA100.p10.v66	5936.83	d198.p50.v33	5919.38	dsj1000.p80.v83	13244230.00	ch150.p10.v66	1710.72	att532.p50.v33	40915.18	eil101.p80.v83	512.06
kroA100.p10.v83	6476.77	d198.p50.v50	5475.46	dsj1000.p90.v16	13609660.00	ch150.p10.v83	1625.37	att532.p50.v50	44604.81	eil101.p90.v16	522.45
kroA100.p20.v16	4100.97	d198.p50.v66	5828.26	dsj1000.p90.v33	13744190.00	ch150.p20.v16	1513.44	att532.p50.v66	46501.26	eil101.p90.v33	528.18
kroA100.p20.v33	6470.23	d198.p50.v83	7830.80	dsj1000.p90.v50	13890010.00	ch150.p20.v33	1612.35	att532.p50.v83	50847.65	eil101.p90.v50	535.83
kroA100.p20.v50	6026.88	d198.p60.v16	6508.36	dsj1000.p90.v66	13891700.00	ch150.p20.v50	2105.61	att532.p60.v16	45435.27	eil101.p90.v66	534.22
kroA100.p20.v66	5124.01	d198.p60.v33	7300.36	dsj1000.p90.v83	14245880.00	ch150.p20.v66	2173.89	att532.p60.v33	46825.00	eil101.p90.v83	537.92
kroA100.p20.v83	9420.66	d198.p60.v50	7928.06	dsj1000.p10	1480976.00	ch150.p20.v83	2620.86	att532.p60.v50	49963.04	eil101.p10	58.11
kroA100.p30.v16	6301.18	d198.p60.v66	7764.63	dsj1000.p20	2961951.00	ch150.p30.v16	1920.28	att532.p60.v66	50437.37	eil101.p20	116.23
kroA100.p30.v33	6794.94	d198.p60.v83	8273.28	dsj1000.p30	4442927.00	ch150.p30.v33	2263.51	att532.p60.v83	54291.68	eil101.p30	174.35
kroA100.p30.v50	8693.83	d198.p70.v16	7408.01	dsj1000.p40	5923903.00	ch150.p30.v50	2507.08	att532.p70.v16	52322.56	eil101.p40	232.46
kroA100.p30.v66	8497.56	d198.p70.v33	8466.25	dsj1000.p50	7404878.00	ch150.p30.v66	2591.19	att532.p70.v33	55238.08	eil101.p50	290.58
kroA100.p30.v83	10210.88	d198.p70.v50	8786.42	dsj1000.p60	8885854.00	ch150.p30.v83	3059.95	att532.p70.v50	56307.43	eil101.p60	348.69
kroA100.p40.v16	8429.70	d198.p70.v66	9152.07	dsj1000.p70	10366830.00	ch150.p40.v16	2513.76	att532.p70.v66	59166.41	eil101.p70	406.81
kroA100.p40.v33	8791.15	d198.p70.v83	9065.00	dsj1000.p80	11847810.00	ch150.p40.v33	2809.98	att532.p70.v83	58578.00	eil101.p80	464.93
kroA100.p40.v50	10315.35	d198.p80.v16	8948.07	dsj1000.p90	13328780.00	ch150.p40.v50	2915.05	att532.p80.v16	58741.72	eil101.p90	523.04
kroA100.p40.v66	10534.56	d198.p80.v33	8038.98	rat783.p10.v16	1185.97	ch150.p40.v66	3321.94	att532.p80.v33	60574.31	lin318.p10.v16	6041.33
kroA100.p40.v83	10244.26	d198.p80.v50	9310.76	rat783.p10.v33	1558.38	ch150.p40.v83	3325.32	att532.p80.v50	64628.56	lin318.p10.v33	7957.71
kroA100.p50.v16	9754.43	d198.p80.v66	8839.78	rat783.p10.v50	1511.61	ch150.p50.v16	3004.14	att532.p80.v66	62711.97	lin318.p10.v50	7874.08
kroA100.p50.v33	10218.20	d198.p80.v83	9693.90	rat783.p10.v66	1915.01	ch150.p50.v33	3136.32	att532.p80.v83	63619.20	lin318.p10.v66	11944.43
kroA100.p50.v50	10102.49	d198.p90.v16	9535.53	rat783.p10.v83	1872.50	ch150.p50.v50	3198.81	att532.p90.v16	65035.06	lin318.p10.v83	12287.68
kroA100.p50.v66	9861.52	d198.p90.v33	10009.58	rat783.p20.v16	2034.14	ch150.p50.v66	3665.11	att532.p90.v33	64913.32	lin318.p20.v16	9132.71
kroA100.p50.v83	12483.14	d198.p90.v50	10276.10	rat783.p20.v33	2245.76	ch150.p50.v83	3319.95	att532.p90.v50	66682.62	lin318.p20.v33	10408.27
kroA100.p60.v16	10284.78	d198.p90.v66	10018.95	rat783.p20.v50	2601.12	ch150.p60.v16	3622.58	att532.p90.v66	67241.73	lin318.p20.v50	13731.64
kroA100.p60.v33	11442.14	d198.p90.v83	10081.61	rat783.p20.v66	2563.15	ch150.p60.v33	3659.43	att532.p90.v83	69304.29	lin318.p20.v66	13718.98
kroA100.p60.v50	11176.49	d198.p10	1064.02	rat783.p20.v83	3198.61	ch150.p60.v50	3809.46	att532.p10	7122.53	lin318.p20.v83	15270.89
kroA100.p60.v66	13435.22	d198.p20	2128.03	rat783.p30.v16	2656.69	ch150.p60.v66	4007.59	att532.p20	14245.05	lin318.p30.v16	11208.81
kroA100.p60.v83	13438.37	d198.p30	3192.04	rat783.p30.v33	2981.47	ch150.p60.v83	4175.02	att532.p30	21367.58	lin318.p30.v33	14177.21
kroA100.p70.v16	12326.80	d198.p40	4256.06	rat783.p30.v50	3391.91	ch150.p70.v16	4198.73	att532.p40	28490.10	lin318.p30.v50	15683.86
kroA100.p70.v33	13040.19	d198.p50	5320.07	rat783.p30.v66	3619.27	ch150.p70.v33	4247.70	att532.p50	35612.63	lin318.p30.v66	17407.57
kroA100.p70.v50	13087.34	d198.p60	6384.09	rat783.p30.v83	3875.69	ch150.p70.v50	4171.51	att532.p60	42735.15	lin318.p30.v83	17945.20
kroA100.p70.v66	14119.99	d198.p70	7448.10	rat783.p40.v16	3337.11	ch150.p70.v66	4509.64	att532.p70	49857.68	lin318.p40.v16	13919.39
kroA100.p70.v83	14678.74	d198.p80	8512.12	rat783.p40.v33	3839.62	ch150.p70.v83	4612.88	att532.p80	56980.21	lin318.p40.v33	16229.05
kroA100.p80.v16	14419.79	d198.p90	9576.13	rat783.p40.v50	4046.29	ch150.p80.v16	4592.32	att532.p90	64102.73	lin318.p40.v50	19063.22
kroA100.p80.v33	14398.59	dsj1000.p10.v16	2413777.00	rat783.p40.v66	4210.95	ch150.p80.v33	4706.49	eil101.p10.v16	73.71	lin318.p40.v66	19559.90
kroA100.p80.v50	14431.74	dsj1000.p10.v33	2741702.00	rat783.p40.v83	4187.43	ch150.p80.v50	4743.34	eil101.p10.v33	93.00	lin318.p40.v83	21746.95
kroA100.p80.v66	16623.24	dsj1000.p10.v50	3375467.00	rat783.p50.v16	4044.51	ch150.p80.v66	4955.88	eil101.p10.v50	93.04	lin318.p50.v16	16332.09
kroA100.p80.v83	14786.36	dsj1000.p10.v66	4028898.00	rat783.p50.v33	4314.19	ch150.p80.v83	4862.40	eil101.p10.v66	133.67	lin318.p50.v33	17159.94
kroA100.p90.v16	16050.75	dsj1000.p10.v83	4327180.00	rat783.p50.v50	4860.15	ch150.p90.v16	5064.83	eil101.p10.v83	160.42	lin318.p50.v50	19623.45
kroA100.p90.v33	15994.13	dsj1000.p20.v16	3782654.00	rat783.p50.v66	5043.91	ch150.p90.v33	5209.75	eil101.p20.v16	139.24	lin318.p50.v66	22333.14
kroA100.p90.v50	16382.99	dsj1000.p20.v33	4523072.00	rat783.p50.v83	5203.94	ch150.p90.v50	5154.09	eil101.p20.v33	171.47	lin318.p50.v83	25434.79
kroA100.p90.v66	15523.36	dsj1000.p20.v50	5127926.00	rat783.p60.v16	4797.07	ch150.p90.v66	5249.80	eil101.p20.v50	175.11	lin318.p60.v16	19972.60
kroA100.p90.v83	15323.54	dsj1000.p20.v66	5778787.00	rat783.p60.v33	5170.12	ch150.p90.v83	5083.59	eil101.p20.v66	190.24	lin318.p60.v33	21346.71
kroA100.p10	1709.10	dsj1000.p20.v83	6450438.00	rat783.p60.v50	5266.92	ch150.p10	556.14	eil101.p20.v83	198.62	lin318.p60.v50	23115.39
kroA100.p20	3418.30	dsj1000.p30.v16	5447787.00	rat783.p60.v66	5674.25	ch150.p20	1112.28	eil101.p30.v16	197.10	lin318.p60.v66	23603.75
kroA100.p30	5127.45	dsj1000.p30.v33	5975096.00	rat783.p60.v83	5398.97	ch150.p30	1668.42	eil101.p30.v33	220.63	lin318.p60.v83	25870.44
kroA100.p40	6836.60	dsj1000.p30.v50	6389477.00	rat783.p70.v16	5459.45	ch150.p40	2224.56	eil101.p30.v50	217.30	lin318.p70.v16	21908.55
kroA100.p50	8545.75	dsj1000.p30.v66	7436660.00	rat783.p70.v33	5707.45	ch150.p50	2780.70	eil101.p30.v66	250.27	lin318.p70.v33	23232.24
kroA100.p60	10254.91	dsj1000.p30.v83	7400875.00	rat783.p70.v50	5847.90	ch150.p60	3336.84	eil101.p30.v83	282.37	lin318.p70.v50	23834.18
kroA100.p70	11964.06	dsj1000.p40.v16	6978520.00	rat783.p70.v66	5918.57	ch150.p70	3892.98	eil101.p40.v16	245.70	lin318.p70.v66	24154.62
kroA100.p80	13673.21	dsj1000.p40.v33	7315055.00	rat783.p70.v83	5996.61	ch150.p80	4449.12	eil101.p40.v33	301.76	lin318.p70.v83	25054.59
kroA100.p90	15382.36	dsj1000.p40.v50	8011461.00	rat783.p80.v16	6180.58	ch150.p90	5005.26	eil101.p40.v50	286.42	lin318.p80.v16	24802.40
d198.p10.v16	2141.26	dsj1000.p40.v66	8898664.00	rat783.p80.v33	6259.45	att532.p10.v16	11676.86	eil101.p40.v66	329.05	lin318.p80.v33	24251.09
d198.p10.v33	2278.54	dsj1000.p40.v83	8798043.00	rat783.p80.v50	6500.02	att532.p10.v33	14685.17	eil101.p40.v83	290.38	lin318.p80.v50	25839.17
d198.p10.v50	2449.09	dsj1000.p50.v16	8113792.00	rat783.p80.v66	6551.43	att532.p10.v50	17183.52	eil101.p50.v16	309.33	lin318.p80.v66	26770.53
d198.p10.v66	2323.21	dsj1000.p50.v33	8625677.00	rat783.p80.v83	6566.91	att532.p10.v66	20708.56	eil101.p50.v33	338.21	lin318.p80.v83	27827.87
d198.p10.v83	4857.64	dsj1000.p50.v50	9405494.00	rat783.p90.v16	6895.74	att532.p10.v83	21741.35	eil101.p50.v50	352.59	lin318.p90.v16	26030.73
d198.p20.v16	2388.82	dsj1000.p50.v66	9744898.00	rat783.p90.v33	6967.85	att532.p20.v16	18985.72	eil101.p50.v66	331.27	lin318.p90.v33	26597.94
d198.p20.v33	2964.04	dsj1000.p50.v83	9878380.00	rat783.p90.v50	6996.76	att532.p20.v33	22303.80	eil101.p50.v83	336.15	lin318.p90.v50	27371.21
d198.p20.v50	2755.41	dsj1000.p60.v16	9653119.00	rat783.p90.v66	7108.17	att532.p20.v50	25554.69	eil101.p60.v16	377.13	lin318.p90.v66	27692.71
d198.p20.v66	3385.44	dsj1000.p60.v33	9820027.00	rat783.p90.v83	6991.72	att532.p20.v66	28117.06	eil101.p60.v33	345.66	lin318.p90.v83	27449.76
d198.p20.v83	4391.14	dsj1000.p60.v50	10557770.00	rat783.p10	747.59	att532.p20.v83	30282.42	eil101.p60.v50	401.28	lin318.p10	2730.06
d198.p30.v16	3871.18	dsj1000.p60.v66	10992180.00	rat783.p20	1495.19	att532.p30.v16	26330.13	eil101.p60.v66	366.17	lin318.p20	5460.11
d198.p30.v33	3415.73	dsj1000.p60.v83	11566720.00	rat783.p30	2242.78	att532.p30.v33	28601.60	eil101.p60.v83	438.00	lin318.p30	8190.17
d198.p30.v50	4457.03	dsj1000.p70.v16	10971300.00	rat783.p40	2990.38	att532.p30.v50	30680.62	eil101.p70.v16	416.54	lin318.p40	10920.22
d198.p30.v66	7155.86	dsj1000.p70.v33	11217690.00	rat783.p50	3737.97	att532.p30.v66	34881.71	eil101.p70.v33	417.53	lin318.p50	13650.27
d198.p30.v83	5013.44	dsj1000.p70.v50	11729110.00	rat783.p60	4485.56	att532.p30.v83	37100.20	eil101.p70.v50	439.06	lin318.p60	16380.33
d198.p40.v16	4896.63	dsj1000.p70.v66	12062110.00	rat783.p70	5233.16	att532.p40.v16	32495.83	eil101.p70.v66	434.64	lin318.p70	19110.38
d198.p40.v33	6255.30	dsj1000.p70.v83	12047830.00	rat783.p80	5980.75	att532.p40.v33	36310.92	eil101.p70.v83	413.83	lin318.p80	21840.44
d198.p40.v50	4607.99	dsj1000.p80.v16	12246860.00	rat783.p90	6728.35	att532.p40.v50	38035.95	eil101.p80.v16	477.86	lin318.p90	24570.49

Table 4.5: Lower bound of the optimal PTSP solutions for our PTSP benchmark.

Chapter 5

Ant Colony Optimization

In our case study on metaheuristics for SCOPs we have chosen to focus on ACO for at least two reasons. First of all, ACO has been shown to be successful in several difficult combinatorial optimization problems, and particularly in many routing problems, to which the PTSP belongs. Second, at the time the research for this thesis started, there were still no applications of ACO algorithms to SCOPs, and the investigation of ACO potentiality in solving problems from the SCOP domain was a completely open issue. In this chapter we propose several ACO algorithms, beginning, in Section 5.1, with one which is a straightforward extension to the PTSP of an ACO algorithm originally developed for the TSP. We will show experimentally that this first ACO version exploiting the exact PTSP objective function already obtains quite good results, outperforming all the simple heuristics. We will also show that the performance of ACO strongly depends on the PTSP instance characteristics, and that if the 'stochasticity' of an instance is under a certain level, it is better treating it like a (deterministic) TSP. In Section 5.2 we investigate some issues related to the use of objective function approximations inside ACO. As we have seen in the first part of this thesis (Section 2.4), there are two types of possible approximations: ad hoc and sampling approximations. Here, we will consider both types of approximations, and analyze for each type different aspects. For ad hoc approximations, we will see how different precision measures are correlated with the solution quality of an ACO algorithm, when ACO uses ad hoc approximations. Moreover, we will take advantage of the fact that the PTSP objective function is known exactly, to measure the effectiveness of using the sampling approximation inside ACO. A critical overview of the obtained results is done in Section 5.3.

5.1 A straightforward implementation

Because of the structural similarity between the PTSP and the TSP (the solution structure is the same, only the cost of a solution is different), as a first implementation of the ACO algorithm for the PTSP, we consider an adaptation to the PTSP of the ACS algorithm by Dorigo and Gambardella [69], which was successfully applied to the TSP. We call this algorithm probabilistic ACS, or pACS.

In this section, we describe in detail how pACS and ACS work, and we show some preliminary experimental evaluation of pACS that also motivates the subsequent investigations of the following sections.

5.1.1 The pACS algorithm

The main principles of the ACO metaheuristic have been described in Section 3.1. Here, we focus on one particular ACO algorithm, the pACS, whose pseudocode skeleton is shown by Algorithm 8. The procedures Initialization, ConstructAntSolution, and

Algorithm 8 pACS

```
1: Initialization
2: for iteration k = 1, 2, ... do
3:    Initialize best ant solution λ_BA
4:    for ant a = 1, 2, ..., m do
5:       ConstructAntSolution [each ant constructs its solution λ_a]
6:       if E[L(λ_a)] < E[L(λ_BA)] then
7:          set λ_BA = λ_a
8:       end if
9:    end for
10:   if E[L(λ_BA)] < E[L(λ_BSF)] then
11:      set λ_BSF = λ_BA
12:   end if
13:   GlobalPheromoneUpdate
14: end for
```

GlobalPheromoneUpdate work as follows.

Initialization consists of four operations: the positioning of m ants on their starting customers, the initialization of the best-so-far-solution λ_{BSF}, the computation and initialization of the heuristic information η, and the initialization of pheromone τ. Note that η and τ are bidimensional matrices of information, where η_{ij} and τ_{ij} are the values of the heuristic information, respectively pheromone, on the arc (i, j) that goes from customer i to customer j. Initialization is done as follows in pACS: the starting customer of each ant is chosen randomly; heuristic information is so that $\eta_{ij} = 1/d_{ij}$, where d_{ij} is the distance between i and j; pheromone values are set all equal to τ_0, which is computed according to

$$\tau_0 = \frac{1}{n \cdot E[L(\lambda_{FI})]}, \tag{5.1}$$

where $E[L(\lambda_{FI})]$ is the expected length of the tour constructed by the Farthest Insertion heuristic (see Section 4.5). As noted in [69], in the denominator of Equation (5.1) any very rough approximation of the optimal solution value would suffice, therefore, the expected cost of other heuristics could be used for the initialization of τ_0. Observe that, in this simple pACS version, η is a static information that is computed just once during the initialization phase.

ConstructAntSolution is the procedure by which each ant probabilistically builds a tour by choosing the next customer to move to on the basis of the two types of information: the pheromone τ and the heuristic information η. When an ant a is on city i, the next city is chosen as follows.

- With probability q_0 (a parameter), a city j that maximizes $\tau_{ij} \cdot \eta_{ij}^{\beta}$ is chosen in the set $J_a(i)$ of the cities not yet visited by ant a. Here, β is a parameter which determines the relative influence of the heuristic information.
- With probability $1 - q_0$, a city j is chosen randomly with a probability given by

$$p_a(i,j) = \begin{cases} \frac{\tau_{ij} \cdot \eta_{ij}^{\beta}}{\sum_{r \in J_a(i)} \tau_{ir} \cdot \eta_{ir}^{\beta}}, & \text{if } j \in J_a(i) \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

Hence, with probability q_0 the ant chooses the best city according to the pheromone trail and to the distance between cities, while with probability $1 - q_0$ it explores the search space in a biased way.

The procedure ConstructAntSolution also takes care of local pheromone updates, where each ant, after it has chosen the next city to move to, applies the following local update rule:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0, \tag{5.3}$$

where τ_0 is the initial pheromone parameter, and ρ, $0 < \rho \leq 1$, is another parameter. The effect of the local updating rule is to make less desirable an arc which has already been chosen by an ant, so that the exploration of different tours is favored during one iteration of the algorithm.

After each ant has built its solution, the best ant solution λ_{BA} is updated and stored (steps 6 to 8 of Algorithm 8) for future comparison with the best-so-far-solution λ_{BSF} (steps 10 to 12 of Algorithm 8). After λ_{BSF} has also been updated, the GlobalPheromoneUpdate function is applied to modify pheromone on arcs belonging to λ_{BSF} with the following global updating rule

$$\tau_{ij} \leftarrow (1 - \alpha) \cdot \tau_{ij} + \alpha \cdot \Delta\tau_{ij}, \tag{5.4}$$

where

$$\Delta\tau_{ij} = \frac{1}{E[L(\lambda_{BSF})]} \tag{5.5}$$

with $0 < \alpha \leq 1$ being the pheromone decay parameter, and $E[L(\lambda_{BSF})]$ is the expected cost of the best-so-far solution.

As we have already pointed out, pACS is a straightforward adaptation of the ACS algorithm originally designed for the TSP [69]. Basically, if in the above description of pACS (in Equations (5.1) and (5.5), and in steps 6 to 10 of Algorithm 8) we substitute the expected length of a solution with the length of it, we obtain ACS. Thus, from the point of view of code, pACS is very similar to ACS. Note, however, that from the complexity point of view there are more important differences, since the expected

length of a solution needs $O(n^2)$ time to be computed, while the length of a solution only requires $O(n)$ time. It is useful here to consider in more detail what is the asymptotic time complexity of each iteration of pACS, as a function of the number of customers n and of the number of ants m:

$$\begin{aligned} t(\text{one iteration of pACS or ACS}) = {} & m \cdot t(\mathsf{ConstructAntSolution}) \\ & + m \cdot t(\text{computation of the objective value}) \\ & + t(\mathsf{GlobalPheromoneUpdate}). \end{aligned} \tag{5.6}$$

Procedures ConstructAntSolution and GlobalPheromoneUpdate require respectively $O(n^2)$ and $O(n)$ time both in pACS and ACS. Thus, we have

$$t(\text{one iteration of pACS}) = 2m \cdot O(n^2) + O(n) = O(n^2), \tag{5.7}$$

$$t(\text{one iteration of ACS}) = m \cdot O(n^2) + (m+1)O(n) = O(n^2). \tag{5.8}$$

The asymptotic time complexity is the same in the two algorithms ($O(n^2)$), but the constant involved are different. In practice, given the same computation time is allowed, pACS will be able to perform far fewer iterations than ACS. These observations motivate the investigation, done in Section 5.2, of other variants of ACS for solving the PTSP, where fast approximations of the objective function are used, instead of the exact one based on the expected length of a solution.

5.1.2 Experimental analysis

5.1.2.1 Computational environment

All the experiments reported in this Section have been run on a machine with two processors Intel(R) Xeon(TM) CPU 1.70GHz, running the GNU/Linux Debian 2.4.27 operating system. All algorithms have been coded in C++ under the same development framework.

5.1.2.2 Tuning

The tunable parameters of pACS, as described in the previous section, are

- m, the number of ants;
- q_0, the probability that the next customer is chosen deterministically;
- β, the power of heuristic information exponent;
- ρ, the local evaporation factor;
- α, the global evaporation factor.

We have verified empirically that pACS is not very sensitive to parameters m, ρ, and α, when these have values near the ones suggested by Dorigo and Stützle [72] for the ACS algorithm applied to the TSP. Thus, according to [72], we set $m = 10$, and

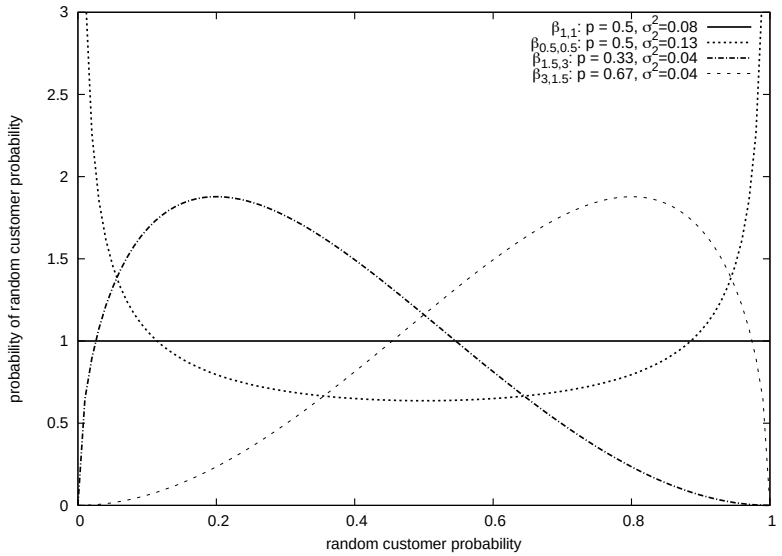


Figure 5.1: Probability distributions used for randomly generating heterogeneous customer probabilities in the tuning PTSP instances.

$\rho = \alpha = 0.1$. Tuning of parameters q_0 and β was done considering, respectively, three and four different values as reported in Table 5.1. This results in twelve different pACS programs to be run on a set of tuning PTSP instances.

q_0	0.95		0.98	1
β	1	2	3	4

Table 5.1: Values of q_0 and β parameters considered for the tuning of pACS.

The set of tuning PTSP instances has been constructed in the following way. Customers coordinates are from two instances of the TSPLIB [178], namely eil51.tsp and pcb442.tsp, with, respectively 51 and 442 customers. Customers probabilities have been generated according to different probability configurations, both homogeneous and heterogeneous. Homogeneous probabilities p belong to the set $\{0.25; 0.5; 0.75\}$. Heterogeneous probabilities have been generated according to the 'beta probability distribution' (Equation (4.5)), with four couples of a and b parameters, corresponding to the four different probability distributions plotted in Figure 5.1. The customers probability configurations considered for each of the two TSP instances of the tuning set are summarized in Table 5.2. Note that these tuning instances do not belong to the PTSP benchmark described in Section 4.3.

The pACS algorithm was run for $n^2/100$ CPU seconds on each tuning instance, and a summary of the results is reported by Figure 5.2. The best choice appears to be $q_0 = 0.95$ and $\beta = 3$. A summary of the parameter set used for the experimental

Probability distribution	p	σ^2
$\delta(p)$	0.25	0
$\delta(p)$	0.5	0
$\delta(p)$	0.75	0
$\beta_{0.5,0.5}(\cdot)$	0.5	0.13
$\beta_{1,1}(\cdot)$	0.5	0.08
$\beta_{1.5,3}(\cdot)$	0.33	0.04
$\beta_{3,1.5}(\cdot)$	0.67	0.04

Table 5.2: Average customers probability p and variance σ^2 characterize the 12 customers probability configurations generated for the PTSP tuning set of pACS. The first three rows correspond to homogeneous PTSP instances, and the last four to heterogeneous PTSP instances. The probability distribution $\beta_{a,b}(\cdot)$ is described by Equation (4.5) in Section 4.3

.

m	q_0	β	ρ	α
10	0.95	3	0.1	0.1

Table 5.3: Parameter set of pACS chosen after tuning.

evaluation of pACS is reported in Table 5.3.

5.1.2.3 Results

The first requirement for a metaheuristic such as pACS is that it performs better than other simple heuristics (otherwise there would be no point in developing a metaheuristic). This requirement is satisfied by pACS, as it is clear from a comparison with simple heuristics of the average results over all the PTSP benchmark in Table 5.4. The simple heuristics find results that are on average more than 90% above the optimal solution lower bound (LB) (as computed in Section 4.4), while pACS is on average just 81.2% above LB. However, the time required by pACS for achieving its performance is much higher than other heuristics (except for RS (Random Search)). This is quite obvious, since pACS generates a very high number of solutions in its iterations, until the available run time is over, while the other simple heuristics (except for RS) generate at most n solutions, and then stop.

The performance of pACS with respect to the lower bound of optimal PTSP solutions is graphically shown in Figure 5.3, where the relative difference of the best found solution of pACS is plotted as a function of three different parameters characterizing PTSP instances, namely average customers probability, customers probability variance, and number of customers. We remark that comparisons with respect to the lower bound are useful for having an objective and repeatible way of evaluating an algorithm for the PTSP, but they do not indicate the amount by which an algorithm really deviates from optimal solutions. The reason is that we do not know how near is

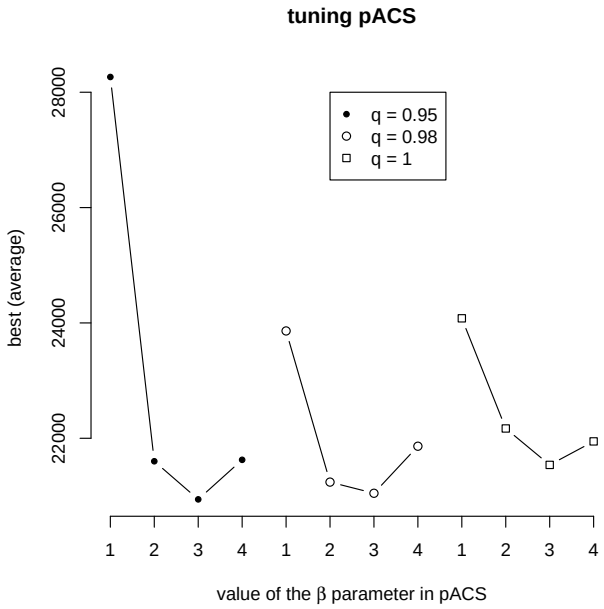


Figure 5.2: Tuning results of pACS obtained by varying q_0 and β parameters. The best choice appears to be $q_0 = 0.95$ and $\beta = 3$.

algorithm	% above LB	time (seconds)
pACS	81.2%	2453.7
FI	90.2%	0.1
SFC	111.6%	0.1
NN	111.1%	35.5
RAD	388.8%	0.1
RS	1228.6%	2712.0

Table 5.4: Aggregated results showing average values over all instances of the PTSP benchmark.

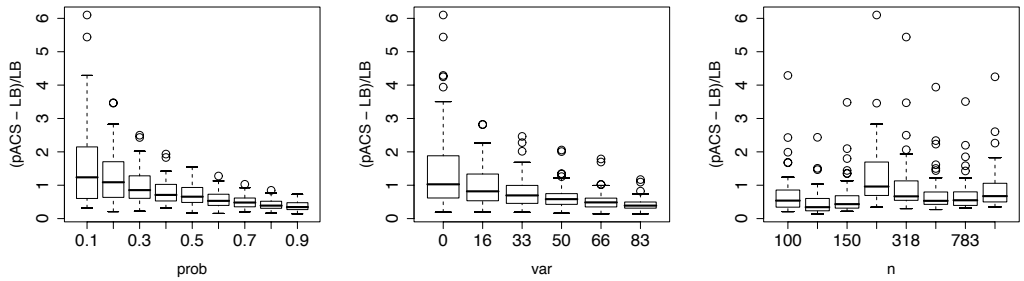


Figure 5.3: Relative distance of the best result found by pACS from the lower bound of the optimal PTSP solution (LB). On the horizontal axis there is, respectively, average customers probability (left), variance of the customers probability (center), number of customers (right). Variance values correspond to the percentage of the maximum variance in the 'beta probability distribution' used to generate random customers probabilities, as explained in Section 4.3.

the lower bound to optimal solutions. For example, Figure 5.3 shows that the average distance from the lower bound is bigger for smaller customers probability, but this does not necessarly mean that pACS finds poorer solutions when customers probability is small, in fact, it could be also the case that, for this type of PTSP instances, the lower bound underestimates the optimal solutions more than for instances characterized by high customers probability.

There is a third type of comparison which is of great importance for evaluating the usefulness of a metaheuristic for the PTSP in general, and of pACS in particular. This is the comparison with the solution quality of optimal TSP solutions evaluated by the PTSP objective function. If the performance of an algorithm developed on purpose for the PTSP is not superior to that of optimal TSP solutions, there would be no point in developing such algorithm. This type of comparison is particularly important here because the TSP is one of the most well known problems in the field of operations research, and very powerful, freely available, algorithms exist for solving to the optimum even very big instances of the TSP. In this thesis, we have used the Concorde software [63] for finding the optimal solution of the TSP instances from which our PTSP benchmark has been generated (see Section 4.3). A boxplot of the ranks of our algorithms including the optimal TSP solutions, is shown in Figure 5.4. The meaning of the vertical line joining two algorithms on the left of the plot is the following: according to the Friedman two-way analysis of variance by ranks [64], the difference between the two algorithms is not statistically significant at a confidence level of 95%. Thus, if two algorithms are *not* linked by the vertical line, their difference is statistically significant. The figure shows that pACS is indeed better than optimal TSP solutions.

It is now interesting to see how the relative gain of pACS with respect to optimal TSP solutions varies by varying different instance characteristics. This is shown by Figure 5.5. The figure reveals at least two important facts. First, the performance of

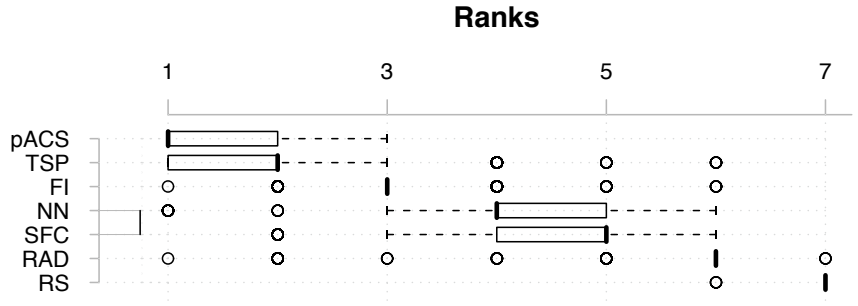


Figure 5.4: Boxplot of the distribution of ranks of pACS, TSP (the optimal solution of the TSP corresponding to the PTSP), and simple heuristics described in Section 4.5. The vertical line on the left of the plot which joins NN and SFC means that these two algorithms are not significantly different at a confidence level of 95%, according to the Friedman two-way analysis of variance by ranks [64]. The interpretation of a box is the following: the solid line inside the box is the median rank, the limits of the box are the lower and upper quartile, the 'whiskers' are the smallest and largest values (outliers excepted), and the circles are the outliers.

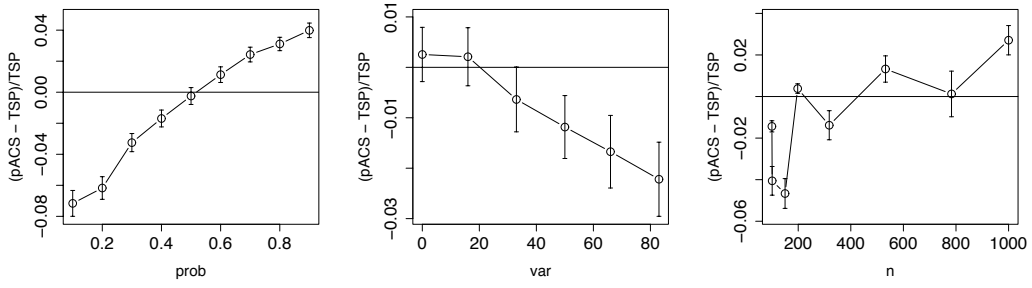


Figure 5.5: Relative gain of the best solution found by pACS over optimal TSP solutions, versus, respectively, average customers probability (left), variance of the customers probability (center), number of customers (right). Circles correspond to the average of the relative gain, and the length of error bars equals its standard deviation.

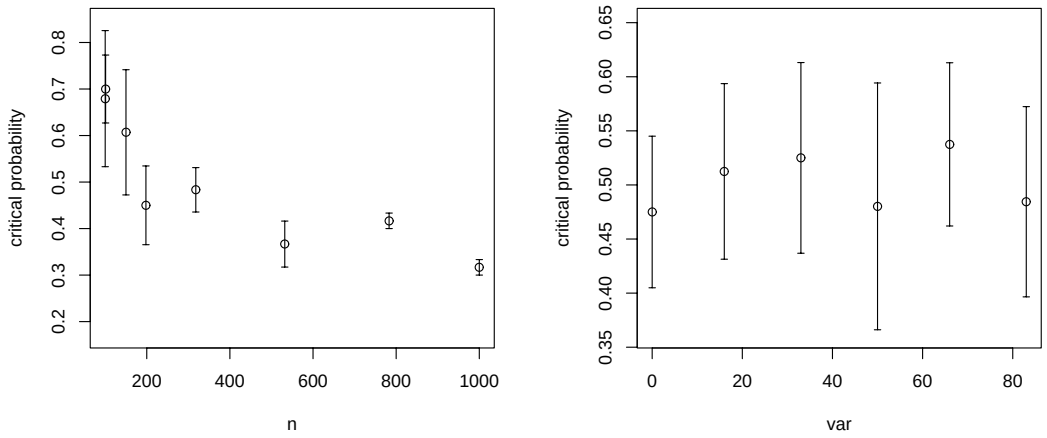


Figure 5.6: Critical probability below which pACS finds solutions whose quality is better than the quality of the optimal TSP solution. Circles correspond to the average critical probability, and the length of error bars equals its standard deviation.

pACS is very different for different average customers probability. In particular, there is a *critical probability* (circa 0.5), under which it is really worth solving PTSP instances by pACS, while above the critical probability the problem can be better treated like a TSP. Second, the variance of customers probability has also a great influence on pACS performance, since, the higher the variance, the more important is the gain of pACS with respect to the optimal TSP solution. Also the number-of-customers factor, shown in the right part of Figure 5.5, seems to influence the performance of pACS, which performs better for smaller number of customers.

It can be useful to have a closer look at the critical probability at which the pACS ceases to find better solutions than the optimal TSP. In Figure 5.6, we have plotted the critical probability by varying the number of customers and the variance of customers probability. It seems that the critical probability goes from a maximum of about 0.8 for instances with a small number of customers to a minimum of about 0.3 for big instances, independently of the variance of customers probability. Detailed results obtained by pACS are shown in Table A.1 in Appendix A.

5.2 The use of objective function approximations in ACO

5.2.1 Motivation

As we have seen, the objective function of the PTSP is computationally expensive, since it needs to consider all the $O(n^2)$ arcs joining couples of customers. For this reason, it is worth investigating the design of fast approximations of the objective function, to be used in optimization algorithms and in ACO in particular, for speeding up, and thus possibly improving, the optimization process.

In this section we present and analyze three types of objective function approximations and three corresponding variations of the pACS algorithm. The first approximation consists in using the TSP objective function instead of the PTSP one, that is, in using the length instead of the expected length of an a priori tour. We call this type of approximation *TSP approximation*. The second approximation is based on the observation that, for a given a priori solution, some edges have a very small probability of being actually travelled, and therefore they may be neglected. We call this type of approximation *Depth approximation* (this naming convention will become clear later in this chapter). These first two approximations belong to the class of ad hoc approximations, as described in Section 2.4.1. The third type of approximation corresponds to the *Sampling approximation* introduced in Section 2.4.2, and consists in computing the sample average (Equation (2.13)) of an a priori tour.

There are several questions of interest, when studying the use of objective function approximations inside an ACO algorithm, such as, for example: verifying whether a fast approximation allows the algorithm to find better solutions than using the exact objective; investigating the effectiveness of different ways of alternating the use of approximated and exact objective; investigating how the tuned parameters of the algorithm change, when using objective function approximations, what measure is more appropriate to guess the effectiveness of an approximation (error, absolute error, correlation with the exact objective, speed, others?). Some of these questions have already been addressed in the context of the PTSP, and it is thus uninteresting to address them again here. In particular, Branke and Guntsch [51, 52] propose the Depth approximation and investigate its effectiveness inside ACO. They show that this approximation accelerates convergence without significantly worsening the solution quality. Gutjahr [100] briefly addresses the PTSP as a pre-test for verifying the performance of S-ACO, an algorithm that uses the Sampling approximation with a variable number of samples for estimating the objective function. In [100], S-ACO has been only tested on very small PTSP instances, since the goal was then to apply it to a problem where there is not a closed-form expression for the objective, and estimation by simulation is the only alternative. Here, the questions we want to address by considering the two ad hoc approximations (TSP- and Depth approximations) and the Sampling approximation are the following:

1. by means of the TSP- and Depth approximations, we want to see which 'precision' measure of the ad hoc approximation is more appropriate to guess the effectiveness of an approximation, or, in other terms, we want to see which precision measure

is more strongly correlated with the solution quality of an ACO algorithm, when ACO uses these ad hoc approximations;

2. we want to verify the quality achieved by an ACO algorithm based on the Sampling approximation.

The usefulness of answering the first question is that, when facing a different SCOP than the PTSP, one may be in a situation where several ad hoc approximations can be used, and the implementation and test of each of them inside ACO in order to find the best one could be a very time consuming task. Thus, some criteria to select a small set of promising ad hoc approximations before their actual implementation inside an ACO, can be useful to save a lot of time.

The interest in the second question is specific to the type of approximation. In fact, the Sampling approximation is a very general type of approximation, that can be used in any SCOP, especially when a closed-form expression for the objective function is missing. In these situations though, it is difficult to evaluate the true performance of the algorithm, precisely because an exact, closed-form expression for the objective function is missing. Therefore, the PTSP gives a good occasion to evaluate the performance of sampling-based algorithm. In a certain sense, our goal is to extend the investigation of [100], which was limited to very few small PTSP instances.

The remainder of this section is organized as follows. The TSP-, Depth-, and Sampling approximations are described, together with their integration into pACS, respectively in Section 5.2.2, 5.2.3, and 5.2.4. In section 5.2.5 we answer by experimental investigation to the two questions described above.

5.2.2 The pACS-T algorithm based on the TSP approximation

5.2.2.1 The TSP approximation

The TSP approximation consists in using the length of an a priori tour instead of its expected length. Thus, given an a priori tour λ, the TSP approximation is based in the computation of

$$L(\lambda) = \sum_{i=1}^{n} d\left(\lambda(i), \lambda(i+1)\right) + d\left(\lambda(n), \lambda(1)\right). \tag{5.9}$$

This approximation can be computed in just $O(n)$ computation time, which is much faster than the $O(n^2)$ computation time of the PTSP objective function. Nevertheless, it is not very accurate, since it does not consider the customers probability. The TSP approximation is particularly inaccurate for PTSP instances with low customers probabilities, but it is quite precise for instances with customers probabilities near 1.

5.2.2.2 The pACS-T algorithm

The pACS-T algorithm (Algorithm 9), is essentially equivalent to the ACS algorithm for the TSP [69], since it exploits for the search of solutions exclusively the length

of a solution (the TSP objective function), instead of the expected length (the PTSP objective function). Differences with respect to pACS are in steps 6 to 13, and 15 of Algorithm 9. Best-ant and best-so-far solutions are updated on the base of their length, and global pheromone update is done with the same rule as in pACS (Equation (5.4)), but with a different quantity of reinforce pheromone (here, $\Delta\tau_{ij} = L(\lambda_{BSF})^{-1}$, while in pACS $\Delta\tau_{ij} = E[L(\lambda_{BSF})]^{-1}$). In order to evaluate the final solution quality of pACS-T, the exact expected cost is computed just once at the end for evaluating the best-so-far-solution λ_{BSF}.

The Initialization procedure is the same as in pACS. In particular, the initial level of pheromone is computed based on the exact expected cost of the Farthest Insertion solution according to Equation (5.1). Also the ConstructAntSolution procedure is the same as in pACS, since it exploits pheromone and heuristic information in the same way. Note, however, that pheromone levels here will evolve differently than in pACS, due to the different amount of pheromone deposited, and due to the fact that λ_{BSF} solutions here are different than λ_{BSF} solutions in pACS.

Algorithm 9 pACS-T

1: Initialization
2: **for** iteration $k = 1, 2, \ldots$ **do**
3: Initialize best ant solution λ_{BA}
4: **for** ant $a = 1, 2, \ldots, m$ **do**
5: ConstructAntSolution [each ant constructs its solution λ_a]
6: **if** $L(\lambda_a) < L(\lambda_{BA})$ **then**
7: set $\lambda_{BA} = \lambda_a$
8: **end if**
9: **end for**
10: **if** $L(\lambda_{BA}) < L(\lambda_{BSF})$ **then**
11: set $\lambda_{BSF} = \lambda_{BA}$
12: **end if**
13: GlobalPheromoneUpdate [using $\Delta\tau_{ij} = L(\lambda_{BSF})^{-1}$]
14: **end for**
15: Compute $E[L(\lambda_{BSF})]$

5.2.3 The pACS-D algorithm based on the Depth approximation

5.2.3.1 The Depth approximation

We say that an edge e_{ij} has depth $\theta \in \{0, 1, \ldots, n-2\}$ with respect to a given a priori tour λ if there are exactly θ cities on the tour between i and j. A high depth θ for an edge implies a small probability for the edge to be actually part of a realized tour, since this would mean that a large number θ of consecutive customers do not require a visit. Therefore, edges with higher depth should impact less on the objective value of an a priori tour. Let us now be more precise.

Given an a priori tour λ and depth values $\theta = 0, 1, ..., n-2$, the edges of the PTSP instance under consideration may be grouped in sets $\lambda^{(\theta)}$, each set containing edges of the same depth. Note that the edge set $\lambda^{(\theta)}$ contains a number of subtours (that is, tours which visit a subset of the customers) equal to $\gcd(n, \theta+1)$[1]. For instance, the set $\lambda^{(0)}$ contains exactly the edges of the a priori tour. In general, however, it may be the case that the subtours formed in such a way can never be realized under the a priori strategy. As an example of edge partition according to the depth, Figure 5.7 shows, from left to right, an a priori tour λ (which coincides with the set of edges $\lambda^{(0)}$), $\lambda^{(1)}$ and $\lambda^{(2)}$ for a PTSP with 8 customers. Note that $\lambda^{(1)}$ contains two subtours which could be actually realized under the a priori strategy, but $\lambda^{(2)}$ contains one single tour that visits all customers, and therefore cannot be realized under the a priori strategy (since, if all customers are to be visited, then they are visited according to the a priori tour λ).

Given the partition of edges according to their depth $\theta = 0, 1, 2, ..., n-2$, the exact PTSP objective function may be written as a sum of terms of increasing depth:

$$E[L(\lambda)] = \sum_{\theta=0}^{n-2} \left[\sum_{j=1}^{n} d(\lambda(j), \lambda(j+\theta+1)) \cdot p_{\lambda(j)} p_{\lambda(j+\theta+1)} \prod_{j+1}^{j+\theta} q_{\lambda} \right], \tag{5.10}$$

where $\lambda = (\lambda(1), \lambda(2), ..., \lambda(n))$ is the a priori tour. In the homogeneous PTSP, where $p_i = p$ for all customers $i \in V$, Equation (5.10) may be written as

$$E[L(\lambda)] = \sum_{\theta=0}^{n-2} L(\lambda^{(\theta)}) p^2 (1-p)^{\theta} \tag{5.11}$$

where $L(\lambda^{(\theta)}) \equiv \sum_{j=1}^{n} d(j, (j+1+\theta))$. The $L(\lambda^{(\theta)})$'s have the combinatorial interpretation of being the sum of the lengths of the collection of subtours in $\lambda^{(\theta)}$. It is now easy to see how the probability of an edge of depth θ to be used decreases with θ. In the homogeneous PTSP (Equation (5.11)) this probability is $p^2(1-p)^{\theta}$, and in the heterogeneous PTSP (Equation (5.10)) it also involves a product over $\theta+2$ probability coefficients. Therefore, the probability of an edge of depth θ to be used (and therefore the weight of such and edge in the objective value) decreases *exponentially* with θ. The idea, in Depth approximation, is to stop the evaluation of the objective function after a fixed depth has been reached. We define

$$E_{D_\theta}[L(\lambda)] = \sum_{r=0}^{\theta} \left[\sum_{j=1}^{n} d(\lambda(j), \lambda(j+r+1)) \cdot p_{\lambda(j)} p_{\lambda(j+r+1)} \prod_{j+1}^{j+r} q_{\lambda} \right], \tag{5.12}$$

and for the homogeneous PTSP this simplifies to

$$E_{D_\theta}[L(\lambda)] = \sum_{r=0}^{\theta} L(\lambda^{(r)}) p^2 (1-p)^{r}. \tag{5.13}$$

[1]The term 'gcd' stays for 'greatest common divisor'.

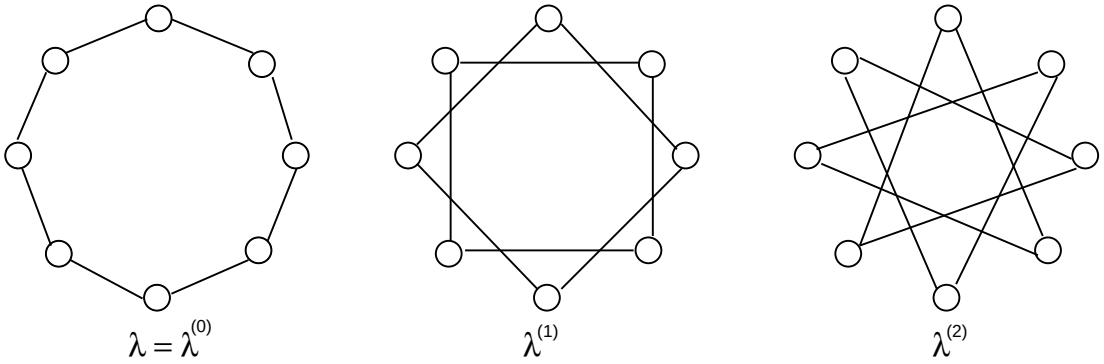


Figure 5.7: Edges grouped in sets according to their depth θ. The picture shows the sets with $\lambda^{(\theta)}$ with $\theta = 0, 1, 2$. Note that $\lambda(0)$ corresponds to the set of edges forming the a priori tour.

The time complexity of Depth approximation is $O(\theta n)$, therefore, the smallest the θ, the bigger the time gain of Depth approximation with respect to the $O(n^2)$ exact objective function. Nevertheless, the smallest the θ, the bigger is the difference, or error, between approximated and exact computation.

5.2.3.2 The pACS-D algorithm

The algorithm pACS is modified in a straightforward manner in order to obtain pACS-D, an algorithm that exploits the Depth approximation for the evaluation of the quality of solutions. The pseudocode of pACS-D is shown in Algorithm 10. The Initialization procedure performs the same tasks as in pACS, but also sets the depth of the approximation to be used for the computation of E_{D_θ}. Differences between pACS-D and pACS are in steps 6 to 13 of Algorithm 10. Best-ant and best-so-far solutions are updated on the base of their Depth approximation expected values, and global pheromone update is done with the same rule as in pACS (Equation (5.4)), but with a different quantity of reinforce pheromone (here, $\Delta\tau_{ij} = E_{D_\theta}[L(\lambda_{BSF})]^{-1}$, while in pACS $\Delta\tau_{ij} = E[L(\lambda_{BSF})]^{-1}$). In order to evaluate the final solution quality of pACS-D, the exact expected cost is computed just once at the end (step 15 of Algorithm 10) for evaluating the best-so-far-solution λ_{BSF}.

5.2.4 The pACS-S algorithm based on the Sampling approximation

5.2.4.1 The Sampling approximation

The Sampling approximation is based on the observation that the PTSP objective function computes an expected value of a random quantity, and therefore it may be estimated by a sample average of the type of Equation (2.13) introduced in Section 2.4.2. More precisely, given an a priori tour λ, the objective function may be written

Algorithm 10 pACS-D

```
Initialization [like in pACS, plus setting of the depth of approximation θ]
for iteration k = 1, 2, ... do
    Initialize best ant solution λ_BA
    for ant a = 1, 2, ..., m do
        ConstructAntSolution [each ant constructs its solution λ_a]
        if E_{D_θ}[L(λ_a)] < E_{D_θ}[L(λ_BA)] then
            set λ_BA = λ_a
        end if
    end for
    if E_{D_θ}[L(λ_BA)] < E_{D_θ}[L(λ_BSF)] then
        set λ_BSF = λ_BA
    end if
    GlobalPheromoneUpdate [using Δτ_ij = (E_{D_θ}[L(λ_BSF)])^{-1}]
end for
Compute E[L(λ_BSF)]
```

as

$$E[L(\lambda)] = \sum_{\omega \subseteq V} p(\omega) L(\lambda_{|\omega}). \tag{5.14}$$

In the above expression, ω is a subset of the set of customers V, $L(\lambda_{|\omega})$ is the length of the tour λ, pruned in such a way as to only visit the customers in *omega*, skipping the others, and $p(\omega)$ is the probability for the subset of customers ω to require a visit:

$$p(\omega) = \prod_{i \in \omega} p_i \prod_{i \in V \setminus \omega} q_i. \tag{5.15}$$

The objective function, as expressed by Equation (5.14), computes the expected length of the tour λ, over all possible random subsets of customers.

The idea, in Sampling approximation, is to *estimate* the exact expected cost (5.14) through sampling in the following way. The length $L(\lambda)$ is a discrete random variable, taking the value $L(\lambda_{|\omega})$ with probability $p(\omega)$. Let ω_i, $i \in 1, 3, ..., N$ be subsets of the original set V of n customers sampled independently with probability $p(\omega_i)$. The Sampling approximation to $E[L(\lambda)]$ is the following

$$E_{S_N}[L(\lambda)] = \frac{1}{N} \sum_{i=1}^{N} L(\lambda_{|\omega_i}). \tag{5.16}$$

The time complexity of Sampling approximation is $O(Nn)$, therefore, the smaller the sample size N, the bigger the time gain of Sampling approximation with respect to the $O(n^2)$ exact objective function. Nevertheless, the smaller N, the bigger the difference, or error, between approximated and exact computation.

Some indication of how the error decreases by increasing the sample size N is given by a fundamental result of probability theory, the so called Strong Law of Large Numbers [95], whose statement is as follows.

Theorem 1 (Strong Low of Large Numbers) *Given an infinite sequence X_1, X_2, $X_3, \ldots$, of independent and identically distributed random variables with common expected value μ and finite variance σ^2, then the sample average*

$$\overline{X}_N = \frac{1}{N}(X_1 + X_2 + X_3 + \ldots + X_N) \tag{5.17}$$

satisfies

$$P\left(\lim_{N\to\infty} \overline{X_N} = \mu\right) = 1 \tag{5.18}$$

that is, the sample average converges almost surely to μ.

This theorem guarantees that, for any λ, $E_N[L(\lambda)]$ converges to $E[L(\lambda)]$ with probability 1 for $N \to \infty$. The 'speed' of convergence of the sample average to the exact expected value may be estimated by evaluating the variance of the sample average. Under the hypotheses of Theorem 1, the variance of the sample average is

$$\begin{aligned} \mathrm{Var}(\overline{X}_N) &= \frac{1}{n^2}\mathrm{Var}(X_1 + X_2 + X_3 + \ldots + X_N) \\ &= \frac{1}{N^2}(\mathrm{Var}(X_1) + \mathrm{Var}(X_2) + \mathrm{Var}(X_3) + \ldots + \mathrm{Var}(X_N)) \\ &= \frac{1}{N^2} N\ \mathrm{Var}(X_1) = \frac{\sigma^2}{N}, \end{aligned} \tag{5.19}$$

Thus, the standard deviation of the sample average is equal to $\sigma/\sqrt{N}$ (with σ being the standard deviation of the random variables $X_1, X_2, X_3, \ldots$). In the context of the PTSP one could say that the error of the Sampling approximation roughly goes as $O(1/\sqrt{N})$.

5.2.4.2 The pACS-S algorithm

In the first part of this thesis, in Section 3.1, we have considered a few papers that focused on ACO algorithms for SCOPs where the objective function is estimated by sampling. One of the main conclusions that these first papers achieve is that, when Sampling approximation is used, the number of samples should be chosen dynamically during the run of the algorithm, and in particular the number of samples should increase during the run. Thus, we have also considered a variable-sample algorithm, pACS-S, whose pseudocode is shown in Algorithm 11. The number of samples N is a variable which is linear in the iteration counter k, and quadratic in the number of customers n, similarly to what has been done by Gutjahr in [100]. More precisely, N is computed according to the following rule

$$N = c + \lfloor b \cdot n^2 \cdot k \rfloor, \tag{5.20}$$

where c and b are two parameters. In pACS-S, after each solution has been constructed, it is evaluated by the Sampling approximation with a freshly generated set of samples (step 8 of Algorithm 11). The same set of samples is used to re-evaluate the best-ant solution λ_{BA}. After the ants construction phase is over, at each iteration a new set of samples is generated, in order to update the best-so-far solution λ_{BSF} (step 13 to 17 of Algorithm 11). Finally, similarly to what is done in pACS-T and pACS-D, the final best-so-far solution is evaluated once with the exact objective function (step 20 of Algorithm 11).

Algorithm 11 pACS-S with variable number of samples

1: Initialization [like in pACS]
2: **for** iteration $k = 1, 2, \ldots$ **do**
3: Initialize best ant solution λ_{BA}
4: Set N = NumberOfSamples(k) [apply Equation (5.20)]
5: **for** ant $a = 1, 2, \ldots, m$ **do**
6: ConstructAntSolution [each ant constructs its solution λ_a]
7: GenerateSamples(N)
8: Compute $E_{S_N}[L(\lambda_a)]$ and re-compute $E_{S_N}[L(\lambda_{BA})]$ using the last generated samples
9: **if** $E_{S_N}[L(\lambda_a)] < E_{S_N}[L(\lambda_{BA})]$ **then**
10: set $\lambda_{BA} = \lambda_a$
11: **end if**
12: **end for**
13: GenerateSamples(N)
14: Re-compute $E_{S_N}[L(\lambda_{BA})]$ and $E_{S_N}[L(\lambda_{BSF})]$ using the last generated samples
15: **if** $E_{S_N}[L(\lambda_{BA})] < E_{S_N}[L(\lambda_{BSF})]$ **then**
16: set $\lambda_{BSF} = \lambda_{BA}$
17: **end if**
18: GlobalPheromoneUpdate [using $\Delta\tau_{ij} = (E_{S_N}[L(\lambda_{BSF})])^{-1}$]
19: **end for**
20: Compute $E[L(\lambda_{BSF})]$

5.2.5 Experimental analysis

5.2.5.1 Tuning

For pACS-T and pACS-D, the same parameters of pACS have been adopted (see Table 5.3 and Section 5.1.2.2), namely $q_0 = 0.95$, $\beta = 3$, $m = 10$, $\rho = \alpha = 0.1$.

For pACS-S, we have considered two settings, one where the number of samples N is kept fixed through all the iterations, and a second one where a variable number of samples is used. In this case, N is increased at each iteration of the algorithm, according to Equation (5.20), where c and b are two parameters to be tuned. Even if we were interested in only evaluating the performance of pACS-S with a variable number of

samples, we decided to execute tuning also in the fixed number of samples setting, in order to have an idea of the difference between the two. Table 5.5 summarizes the parameter values tested for tuning pACS-S. A total of 432 parameter sets has been considered. For each parameter set, pACS-S has been run on a set of 14 tuning

q_0	0.95 0.98 1
β	1 2 3 4
N (fixed)	2 10 50 200
c (N variable)	1 10 50
b (N variable)	10^{-3} 10^{-4} 10^{-5}

Table 5.5: Parameter values considered for tuning pACS-S.

instances (the same used for pACS, as described in Section 5.1.2.2). The parameter set which corresponds to the lowest average best-found-solution-value by pACS-S is a fixed number of samples setting, with β and q_0 equal to the best pACS parameters (that is, $\beta = 3$ and $q_0 = 0.95$), and with $N = 50$. In the variable number of samples setting, the best parameters are $c = 1$, $b = 10^{-5}$, and again $\beta = 3$ and $q_0 = 0.95$. This is the parameter set that has been selected for executing pACS-S on the PTSP benchmark.

5.2.5.2 Results

As we anticipated in Section 5.2.1, the first issue that we address here experimentally is which 'precision' measure of an ad hoc approximation (TSP and Depth approximation) is more strongly correlated with the solution quality of an ACO based on the ad hoc approximation. We consider two precision measures: the linear correlation between the approximation and the exact objective function, and the absolute error of the approximation. In order to compute these two precision measures, we have slightly modified the code of pACS-T and pACS-D, by computing also the exact PTSP objective value each time that a new best-so-far solution is encountered. In practice, we added the instruction 'Compute $E[L(\lambda_{BSF})]$' in the **if** statement starting at line 10 of Algorithm 9 and Algorithm 10. After running pACS-T or pACS-D, the linear correlation r between the ad hoc approximation E_{proxy} (with *proxy* equal to T or to D_θ) and the exact objective E has been computed as follows:

$$r = \frac{\sum_{\lambda_{BSF}} \left(E_{proxy}[L(\lambda_{BSF})] - \overline{E_{proxy}}\right) \left(E[L(\lambda_{BSF})] - \overline{E}\right)}{\sqrt{\sum_{\lambda_{BSF}} \left(E_{proxy}[L(\lambda_{BSF})] - \overline{E_{proxy}}\right)^2 \sum_{\lambda_{BSF}} \left(E[L(\lambda_{BSF})] - \overline{E}\right)^2}}, \tag{5.21}$$

where sums are done over all encountered best-so-far solutions, and $\overline{E_{proxy}}$ and $\overline{E}$ are, respectively, the average approximated value and average exact objective value over all the encountered best-so-far solutions. The other precision measure is the average absolute error ϵ, which is computed as follows

$$\epsilon = \frac{1}{n_{BSF}} \sum_{\lambda_{BSF}} \left| E_{proxy}[(\lambda_{BSF})] - E[(\lambda_{BSF})] \right|, \tag{5.22}$$

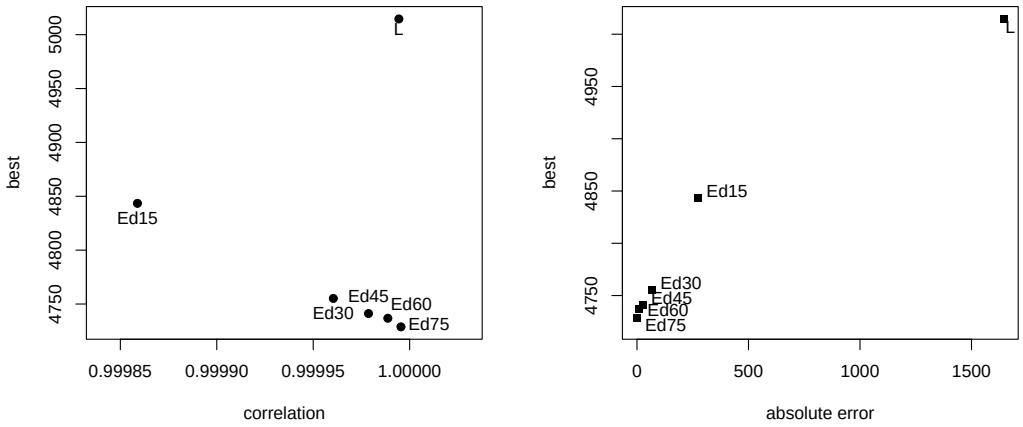


Figure 5.8: Average solution quality versus, respectively, the correlation r between exact and approximated objective function (left), and the absolute error ϵ of the approximated objective function.

where n_{BSF} is the number of best-so-far solutions encountered by the algorithm (pACS-T or pACS-D). In order to evaluate which measure between r and ϵ is better correlated with the solution quality of pACS-T and pACS-D, we have run once each algorithm on all the instances of the PTSP benchmark. The run time of the algorithms has been set equal to twice the time given to pACS, in order not to penalize pACS-T and pACS-D due to the fact that each new best-so-far solution is evaluated twice (once with the approximated objective, and once with the exact objective). For the Depth approximation, we have considered depth values equal to 15, 30, 45, 60, and 75. For each depth value, a different run of pACS-D has been performed. In order to see whether r or ϵ is better correlated with the average solution quality of our ACO algorithms, we have produced scatterplots of the average solution quality versus the precision measure as shown in Figure 5.8. From the plots, it is clear that the absolute error (ϵ) is more strongly correlated with the solution quality than the linear correlation (r). In particular, the TSP approximation (denoted by L in the Figure) has a very high linear correlation with the exact objective, but the average solution quality of pACS-T is very low. The justification of this fact is that the TSP approximation has the highest absolute error, as shown in the right scatterplot of Figure 5.8. We think that this result is reasonable, but not obvious. In fact, even if the TSP approximation is quite imprecise, it is also very fast, and much faster than the Depth approximation. Thus, pACS-T can perform many more iterations than pACS-D, and has more time to select good solutions. It seems, though, that the precision of the objective function is more important than the guidance due to the ACO intensification mechanism based on the use of pheromone.

Let us now analyze our second issue of interest as described in Section 5.2.1, that

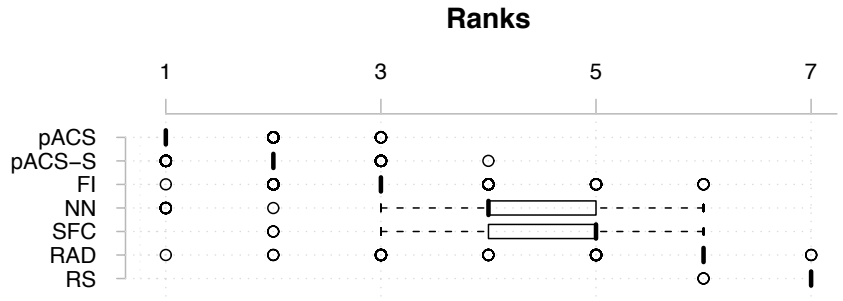


Figure 5.9: Ranking of algorithms on the PTSP benchmark. According to the Friedman two-way analysis of variance by ranks [64], the algorithms have statistically significant different ranks, at a confidence level of 95%.

algorithm	% above LB	time (seconds)
pACS	81.2%	2453.7
pACS-S	83.7%	2367.6
FI	90.2%	0.1
NN	111.6%	35.5
SFC	111.1%	0.1
RAD	388.8%	0.1
RS	1228.6%	2712.0

Table 5.6: Average results of pACS-S compared to other heuristics over all the PTSP benchmark.

is, the quality achieved by pACS-S, which never uses the exact objective function, but is completely based on the Sampling approximation. The ranking of pACS-S with respect to pACS and to other simple heuristics is reported in Figure 5.9, while average deviations from the lower bound are reported in Table 5.6. Interestingly, pACS-S achieves a quite good performance, being better than all simple heuristics, and being worse than pACS of roughly 2%. As we have learned from the experimental analysis of pACS in Section 5.1.2.3, it is important to see how the solution quality varies for different PTSP instance parameters, particularly the average customers probability and the variance of the customers probability. The quality of pACS-S with respect to the optimal TSP solution, by varying the PTSP instance parameters, is shown in Figure 5.10. For comparison purposes, also the results of pACS are reported in the Figure. We can observe that pACS-S has a behavior which is very similar to that of pACS, only being shifted above of roughly 2%. Detailed results of pACS-S on the PTSP instances with average probability smaller than or equal to 0.5 are reported by Table A.2 in Appendix A.

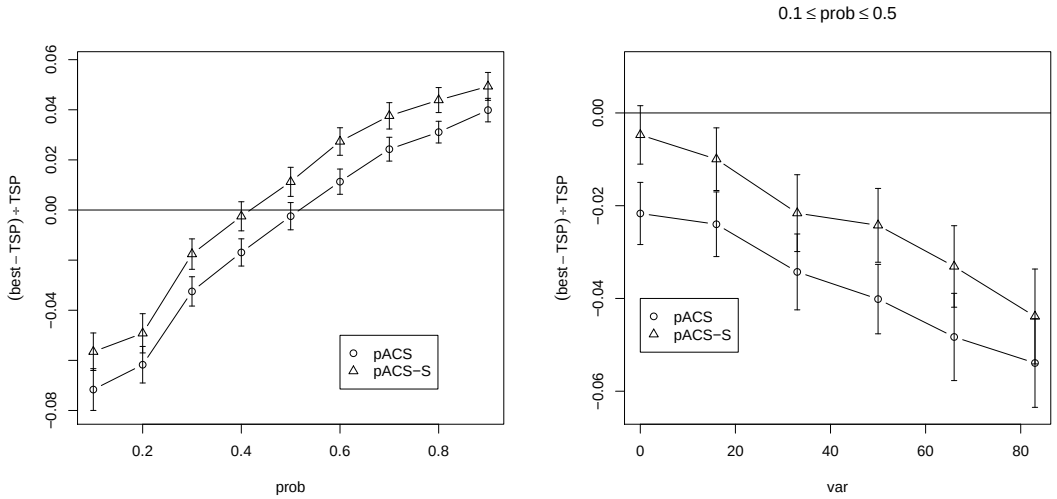


Figure 5.10: Relative gain of pACS-S and pACS over the optimal TSP solution, versus, respectively, average customers probability (left), and variance of the customers probability (right). Points are averages computed over the the complete PTSP benchmark (left) and over subset of PTSP instances with customers probability smaller than or equal to 0.5 (right). The length of error bars corresponds to the standard deviation of the gain.

5.3 Overview of the results

The following are the most important facts observed thanks to the experimental analyses of this section.

- Our straightforward ACO implementation of pACS obtains on average results that are roughly 10% better than the best of the simple heuristics (which is FI (Farthest Insertion)). This is a confirmation that what we can consider a minimal requirement for a metaheuristic is satisfied by ACO.

- When evaluating the performance of an algorithm for the PTSP, it is very important to consider the factors related to the degree of stochasticity of a PTSP instance, since the performance can vary a lot for different degrees of stochasticity. In our benchmark these factors are the average customers probability and the variance of the customers probability.

- pACS, as compared to the expected value of the optimal solution of the TSP, can be qualified indeed very differently, depending on the value of the average customers probability, and of the variance of it. In particular, there is a *critical probability*, under which PTSP instances are really worth solving by pACS, while above the critical probability the problem can be better treated like a TSP. The

average critical probability is 0.5, but in general, the higher the number of customers, the lower the critical probability. Moreover, the higher the variance of customers probability, the more important is the gain of pACS with respect to the optimal TSP solution. Where pACS is most effective, it finds results that are on average 8% better than the optimal TSP solution.

- In considering two ad hoc approximations for the PTSP objective function, we have seen that the absolute error of the approximation is strongly correlated with the performance of the pACS algorithm based on the ad hoc approximation. The same is not true for the linear correlation between the ad hoc approximation and the exact objective: an ad hoc approximation (in this case, the TSP approximation) can be strongly correlated with the exact objective, but its use inside pACS can be degrading. This fact has important consequences, since, when designing ad hoc approximations for a SCOP, one could be tempted to choose one which from preliminary experiments seems well correlated with the exact objective function. But this choice could be, as we have seen, quite bad.

- When solving a SCOP by means of a sampling-based ACO such as our implementation of pACS-S, it is very likely that its performance satisfies at least the minimal requirement of being better than simple heuristics solving a related DCOP. This is, at least, what we observe for the PTSP.

- In our problem, pACS-S found results constantly worse than pACS of roughly 2%, but the qualitative behavior with respect to factors determining the stochasticity of PTSP instances is the same as the pACS algorithm based on the exact objective.

Chapter 6

Local search

This chapter explores the situations that may arise when designing a local search algorithm for a SCOP that, as we have seen in the first part of this thesis, usually has a computationally expensive objective function. The different possibilities that one has in dealing with the complexity of the objective function are analyzed in Section 6.1. Section 6.2 introduces two local search operators, the 2-p-opt and the 1-shift, that are the focus of the next sections. Sections 6.3, 6.4 and 6.5 develop different strategies to deal with the complexity of the objective function and apply them to the 2-p-opt and 1-shift local search operators. Finally, Section 6.6 summarizes the most important contributions of this chapter.

6.1 The issue of complexity and three options to deal with it

The efficiency of a local search algorithm depends on several factors, such as the starting solution, the neighborhood structure, and the speed of exploration of the neighborhood. In particular, the speed of exploration of the neighborhood depends on the time required for computing the difference of the objective value between couples of neighboring solutions (such difference is also referred to as 'move cost').

As we have pointed out in the first part of this thesis, the objective function of SCOPs is computationally demanding, and the PTSP is no exception, since it requires $O(n^2)$ computation time. This can be a serious issue when designing local search algorithms for SCOPs, because it may imply the impossibility of a fast exploration of any neighborhood.

In general there are three possibilities when designing a local search algorithm for a SCOP:

1. Computing the move cost by means of the exact full computation of the difference in the objective value of couples of solutions;

2. Computing the exact move cost more efficiently, by exploiting some particular neighborhood and exploration strategy, that allows recursive computations;

3. Computing an approximated move cost efficiently, by means of fast approximated expressions for the difference in the objective value of couples of solutions;

Option 1 is the easiest to realize, because it can be used for any neighborhood structure, and it does not require any particular search strategy, given that the objective function is explicitly available, like in the PTSP. The problem is of course that it can be very inefficient, due to the computational complexity of the objective function. We will show the impracticality of this approach for the PTSP in Section 6.3.

Option 2, when possible, is in principle the best one. The main problem with this alternative is that it can be very difficult to find a neighborhood structure that allows for a recursive exact computation of the move cost and, given the SCOP, there is no way a priori to say if such a neighborhood even exists. Fortunately, for the PTSP we have been able to find, for two local search operators (the 2-p-opt and the 1-shift) fast and exact recursive move cost computation expressions. The two operators are described in Section 6.2, and their recursive move cost evaluation is derived in Section 6.4.

Option 3 can be a valid alternative to Option 1, when the move cost approximation is both fast and accurate enough to guide the search towards good solutions. Trying this alternative may also be better than trying Option 2, since sometimes it can be easier to find ad hoc approximations of the move cost, instead of finding a neighborhood and a local search operator that allows fast recursive computation. However, finding good approximations of a move cost may require some effort too. Moreover, it is possible that a given approximation is applicable only to a particular neighborhood structure. For the PTSP, we will consider three different move cost approximations in Section 6.5.

6.2 The 2-p-opt and the 1-shift operators

In the literature, there are two local search procedures created specifically for the PTSP that evaluate the move cost in terms of expected value: the 2-p-opt and the 1-shift. The 2-p-opt neighborhood of an a priori tour is the set of tours obtained by reversing a section of the a priori tour (that is, a set of consecutive nodes) such as the example in Figure 6.1. The 2-p-opt is the probabilistic version of the famous 2-opt procedure created for the TSP [133]. The 2-p-opt and the 2-opt are identical in terms of local search neighborhoods, but greatly differ in the cost computation. The change in the TSP objective value (the tour length) can be easily computed in constant time, while the same cannot be said for the PTSP objective value. The 1-shift neighborhood of an a priori tour is the set of tours obtained by moving a node which is at position i to position j of the tour, with the intervening nodes being shifted backwards one space accordingly, as in Figure 6.2. The number of neighbors generated by 2-p-opt and 1-shift moves applied to one a priori tour is $O(n^2)$.

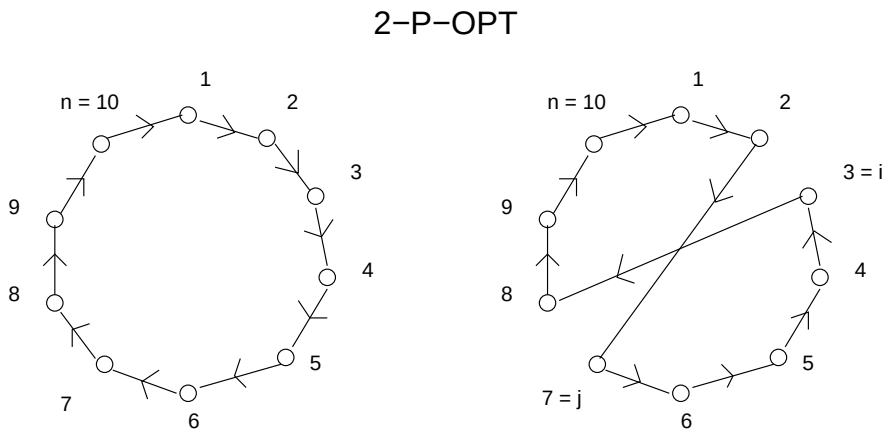


Figure 6.1: A tour $\zeta = (1, 2, ..., i, i+1, ..., j, j+1, ..., n)$ (left) and tour $\zeta_{i,j}$ belonging to its 2-p-opt neighborhood (right) obtained from ζ by reversing the section $(i, i+1, ..., j)$, with $n = 10$, $i = 3$, $j = 7$.

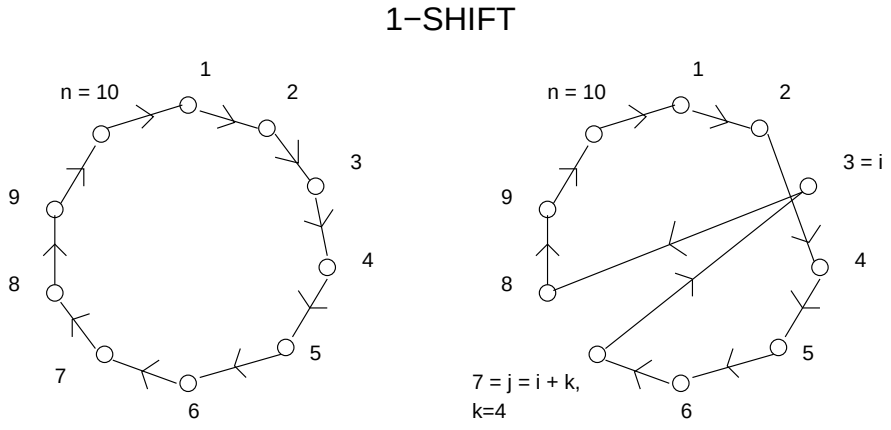


Figure 6.2: A tour $\zeta = (1, 2, ..., i, i+1, ..., j, j+1, ..., n)$ (left) and a tour $\zeta_{i,j}$ belonging to its 1-shift neighborhood (right) obtained from ζ by moving node i to position j and shifting backwards the nodes $(i+1, ..., j)$, with $n = 10$, $i = 3$, $j = 7$.

6.3 The infeasibility of using the exact full objective function

Given that both the 2-p-opt and the 1-shift neighborhoods of any a priori tour contain $O(n^2)$ neighbors, and given that the PTSP objective value of a solution requires $O(n^2)$ computation time, the exploration of a 2-p-opt or a 1-shift neighborhood of any a priori tour requires $O(n^4)$ computation time, when the exact full objective function is used to evaluate the move cost. The same operation in the TSP would require only $O(n^2)$ time, since the objective value (length) difference between two neighboring solutions can be computed in constant time. This is also true for the recursive expressions that we derive in Section 6.4 for the 1-shift and 2-p-opt local search.

An $O(n^4)$ local search is in practice not feasible. As an example, consider the 100-customers PTSP instance kroA100 with homogeneous customers probability equal to 0.5. If we generate one solution by means of the Space Filling Curve heuristic, and we apply just once the 1-shift local search to this solution, the time required to end the search on a 1.7 GhZ processor is 79.3 CPU seconds, while the time needed when the $O(n^2)$ version of 1-shift (derived in Section 6.4) is only 0.19 CPU seconds. Thus, the evaluation of the move cost by using the exact full objective is more than 400 times slower. The time progression of the search for this example is shown in Figure 6.3.

6.4 Exact recursive local search

In the PTSP, it has been luckily possible to derive exact recursive and efficient move cost expressions for the 2-p-opt and 1-shift operators. The recursion on one side allows to store arrays of partial information for future computation and to save time, and on the other side, it forces to explore the 2-p-opt and 1-shift neighborhoods in a fixed lexicographic order as a condition to be able to store the necessary arrays of information.

This section is organized as follows. The first attempts to derive exact recursive expressions for the cost of 2-p-opt and 1-shift moves date back in the Operations Research history. Past work on this issue is reviewed in Section 6.4.1. In Section 6.4.2, we derive new and correct expressions for computing the cost of 2-p-opt and 1-shift local search moves for the general case of heterogeneous customers probabilities. In Section 6.4.3 we specialize the derivation of Section 6.4.2 to the case of homogeneous customers probabilities. Finally, in Section 6.4.4 we perform a computational test showing the effects of using, in the homogeneous PTSP, the incorrect expressions published in the past in [24] and in [27], and the improvement due to the use of the correct cost values.

6.4.1 A bit of history

For the homogeneous PTSP, Bertsimas proposed move evaluation expressions in [27] that explore the neighborhood of a solution (that is, that verify whether an improving 2-p-opt or 1-shift move exists) in $O(n^2)$ time. The intent of Bertsimas' equations is to provide a recursive means to quickly compute the exact change in expected value

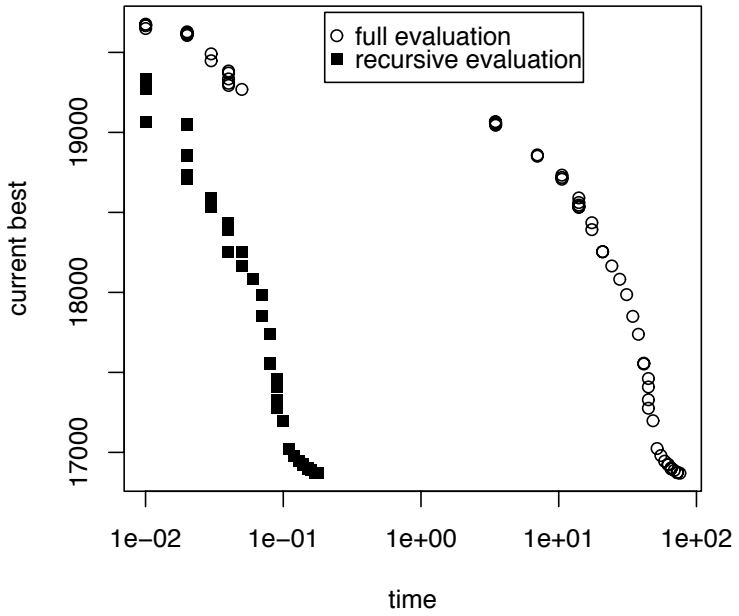


Figure 6.3: Time progression of the 1-shift local search with two types of move cost evaluation. The full exact evaluation is more than 400 times slower than the recursive one. The 1-shift local search of this plot is applied to a solution obtained by the Space Filling Curve heuristic for solving the kroA100 instance with 100 customers and homogeneous customers probability equal to 0.5.

associated with either a 2-p-opt or 1-shift procedure. Evaluating the cost of a local move by computing the cost of two neighboring solutions and then evaluating their difference would require much more time ($O(n^4)$) than a recursive approach. Unfortunately, as we have shown in [41], Bertsimas' expressions do not correctly calculate the change in expected tour length. The correction of equations for computing the cost of local search moves that we propose in this thesis confirms that it is possible to explore both the 2-p-opt and 1-shift neighborhood of a solution in $O(n^2)$ time, and does, as expected, create significant improvement in the already good results for the homogeneous PTSP.

For the heterogeneous PTSP, Bertsimas et al. [26], report computational results of 2-p-opt and 1-shift local search algorithms applied to some small PTSP instances. The results in [26] are based on the work of Chervi [62], who proposed recursive expressions for the cost of 2-p-opt and 1-shift moves for the heterogeneous PTSP. Chervi's expressions explore the 2-p-opt and 1-shift neighborhoods in $O(n^3)$ time, suggesting that it is not possible to retain the $O(n^2)$ complexity of the homogeneous PTSP. Moreover, Chervi's expressions reduce to the incorrect expressions for the homogeneous PTSP

published in [27], when all customer probabilities are equal, and therefore are also not correct. After deriving correct expressions for the computation of the change in expected tour length, we demonstrate that the neighborhood of a solution for this important problem may be explored in $O(n^2)$ time, thus retaining the same complexity as the homogeneous case.

6.4.2 Derivation of local search costs for the heterogeneous PTSP

6.4.2.1 The cost of 2-p-opt moves

Consider, without loss of generality, a tour $\zeta = (1, 2, ..., i, i+1, ..., j, j+1, ..., n)$, and denote by $\zeta_{i,j}$ a tour obtained by reversing a section $(i, i+1, ..., j)$ of ζ, where $i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, n\}$, and $i \neq j$ (see Figure 6.1). Note that if $j < i$, the reversed section includes n. Let $\Delta E_{i,j}$ denote the change in the expected tour length $E[L(\zeta_{i,j})] - E[L(\zeta)]$. We will derive a set of recursive formulas for $\Delta E_{i,j}$ that can be used to efficiently evaluate a neighborhood of 2-p-opt moves. To describe this procedure, we first introduce a few definitions. Let $S, T \subseteq N$ be subsets of nodes, with λ representing any a priori tour, and $\lambda(i)$ representing the customer in the ith position on this tour such that $\lambda = (\lambda(1), \lambda(2), \ldots, \lambda(n))$. The product defined by Equation (4.3) can be easily generalized by replacing q_t with $q_{\lambda(t)}$ and q_u with $q_{\lambda(u)}$.

Definition 7 $E[L(\lambda)]\big|_{T \to S} = \sum_{\lambda(i) \in S, \lambda(j) \in T, i \neq j} d(\,\lambda(i),\, \lambda(j)\,) p_{\lambda(i)} p_{\lambda(j)} \prod_{i+1}^{j-1} q_\lambda$, *that is, the contribution to the expected cost of* λ *due to the arcs from the nodes in* S *to the nodes in* T.

Note that $E[L(\lambda)]\big|_{T \to S} = E[L(\lambda)]$, when $T = S = N$.

Definition 8 $E[L(\lambda)]\big|_{T \leftrightarrow S} = E[L(\lambda)]\big|_{T \to S} + E[L(\lambda)]\big|_{S \to T}$

For the two a priori tours ζ and $\zeta_{i,j}$ we introduce

Definition 9 $\Delta E_{i,j}\big|_{T \leftrightarrow S} = E[L(\zeta_{i,j})]\big|_{T \leftrightarrow S} - E[L(\zeta)]\big|_{T \leftrightarrow S}$, *that is, the contribution to* $\Delta E_{i,j}$ *due to the arcs from the nodes in* S *to the nodes in* T *and from the nodes in* T *to the nodes in* S.

Unlike the TSP, the expected cost of an a priori tour involves the arcs between all of the nodes. The ordering of the nodes on the a priori tour simply affects the probability of an arc being used, and this probability determines the contribution this arc makes to the expected cost of the tour. The change in expected tour length, $\Delta E_{i,j}$, resulting from a reversal of a section is thus based on the change in probability, or weight, placed on certain arcs in the two tours ζ and $\zeta_{i,j}$. While computing $\Delta E_{i,j}$ it is thus necessary to evaluate the weight change of *each* arc. The change in weight on an arc is influenced by how many of its endpoints are included in the reversed section. Because of this, it is useful to consider the following partitions of the node set.

Definition 10 $inside_{i,j} = \{i, \ldots, j\}$, *that is, the section of* ζ *that is reversed to obtain* $\zeta_{i,j}$

Definition 11 $outside_{i,j} = N \setminus inside_{i,j}$.

Using the above definitions, $\Delta E_{i,j}$ may be expressed as

$$\Delta E_{i,j} = \Delta E_{i,j}\big|_{inside_{i,j} \to inside_{i,j}} + \Delta E_{i,j}\big|_{outside_{i,j} \to outside_{i,j}} + \Delta E_{i,j}\big|_{inside_{i,j} \leftrightarrow outside_{i,j}}. \tag{6.1}$$

It is not difficult to verify that the contributions to $\Delta E_{i,j}$ due to $\Delta E_{i,j}\big|_{inside_{i,j} \to inside_{i,j}}$ and to $\Delta E_{i,j}\big|_{outside_{i,j} \to outside_{i,j}}$ are zero. The contribution to $\Delta E_{i,j}$ due to arcs between inside and outside (which is now equal to $\Delta E_{i,j}$) may be split into three components:

$$\begin{aligned} \Delta E_{i,j}\big|_{inside_{i,j} \leftrightarrow outside_{i,j}} = & E[L(\zeta_{i,j})]\big|_{inside_{i,j} \to outside_{i,j}} + \\ & E[L(\zeta_{i,j})]\big|_{outside_{i,j} \to inside_{i,j}} - \\ & E[L(\zeta)]\big|_{inside_{i,j} \leftrightarrow outside_{i,j}}, \end{aligned} \tag{6.2}$$

where the three terms on the right hand side of the last equation are, respectively, the contribution to $E[L(\zeta_{i,j})]$ due to the arcs going from $inside_{i,j}$ to $outside_{i,j}$, the contribution to $E[L(\zeta_{i,j})]$ due to the arcs going from $outside_{i,j}$ to $inside_{i,j}$, and the contribution to $E[L(\zeta)]$ due to arcs joining the two customer sets in both directions. For compactness, these three components will be referenced hereafter by the notation:

$$E^{(1)}_{i,j} = E[L(\zeta_{i,j})]\big|_{inside_{i,j} \to outside_{i,j}} \tag{6.3}$$

$$E^{(2)}_{i,j} = E[L(\zeta_{i,j})]\big|_{outside_{i,j} \to inside_{i,j}} \tag{6.4}$$

$$E^{(3)}_{i,j} = E[L(\zeta)]\big|_{inside_{i,j} \leftrightarrow outside_{i,j}}. \tag{6.5}$$

We may rewrite the expected tour length change $\Delta E_{i,j}$ as follows

$$\Delta E_{i,j} = E^{(1)}_{i,j} + E^{(2)}_{i,j} - E^{(3)}_{i,j}. \tag{6.6}$$

Unlike the TSP, there is an expected cost associated with using an arc in a forward direction as well as a reverse direction, and these costs are usually not the same. The expected costs are based on which customers would have to be "skipped" in order for the arc to be needed in the particular direction. For example, the weight on arc $(1, 2)$ is based only on the probability of nodes 1 and 2 requiring a visit, whereas the weight on arc $(2, 1)$ is also based on the probability of nodes $(3, 4, \cdots, n)$ not requiring a visit. (The tour will travel directly from the 2 to 1 only if none of the rest of the customers on the tour are realized.) For the homogeneous PTSP, the equations are much simpler since the expected cost is based on the number of nodes that are skipped, not which nodes are skipped. This difference dictates the new set of equations we present here.

We will now derive recursive expressions for $E^{(1)}_{i,j}$, $E^{(2)}_{i,j}$, $E^{(3)}_{i,j}$, respectively, in terms of $E^{(1)}_{i+1,j-1}$, $E^{(2)}_{i+1,j-1}$ and $E^{(3)}_{i+1,j-1}$. These recursions are initialized with the expressions

corresponding to entries (i,i) and $(i,i+1)$ for all i. We will derive these expressions quite easily later. First, let us focus on the case where $j = i+k$ and $2 \leq k \leq n-2$. The case $k = n-1$ may be neglected because it would lead to a tour that is reversed with respect to ζ, and, due to the symmetry of distances, this reversed tour would have the same expected length as ζ. Let us consider the tour $\zeta_{i+1,j-1} = (1,2,...,i-1,i,j-1,j-2,...,i+1,j,j+1...,n)$ obtained by reversing section $(i+1,...,j-1)$ of ζ. We can make three important observations. The first one is that the partitioning of customers with respect to $\zeta_{i,j}$ is related to the partitioning with respect to $\zeta_{i+1,j-1}$ in the following way:

$$inside_{i,j} = inside_{i+1,j-1} \cup \{i,j\} \tag{6.7}$$

$$outside_{i,j} = outside_{i+1,j-1} \setminus \{i,j\}. \tag{6.8}$$

The second observation is that for any arc (l,r), with $l \in inside_{i+1,j-1}$ and $r \in outside_{i,j}$, the weight on the arc in the expected total cost equation for $\zeta_{i,j}$ can be obtained by multiplying the weight that the arc has in $\zeta_{i+1,j-1}$ by q_i and dividing it by q_j. One way to see this is to compare the set of skipped customers in $\zeta_{i,j}$ with the set of skipped customers in $\zeta_{i+1,j-1}$. In $\zeta_{i,j}$, the fact that arc (l,r) is used implies that customers $(l-1,l-2,\ldots,i+1,i,j+1,\ldots,r-1)$ are skipped, while in $\zeta_{i+1,j-1}$ using arc (l,r) implies that customers $(l-1,l-2,\ldots,i+1,j,j+1,\ldots,r-1)$ are skipped. Therefore, the set of skipped customers in $\zeta_{i+1,j-1}$ is equal to the set of skipped customers in $\zeta_{i,j}$ except for customer j in $\zeta_{i+1,j-1}$, which is replaced by customer i in $\zeta_{i,j}$. In terms of probabilities, our second observation can be expressed as

$$E[L(\zeta_{i,j})]\big|_{inside_{i+1,j-1} \to outside_{i,j}} = \frac{q_i}{q_j} E[L(\zeta_{i+1,j-1})]\big|_{inside_{i+1,j-1} \to outside_{i,j}}. \tag{6.9}$$

The third important observation is similar to the previous one, but it refers to arcs going in the opposite direction. More precisely, for any arc (r,l), with $r \in outside_{i,j}$ and $l \in inside_{i+1,j-1}$, the weight of the arc in $\zeta_{i,j}$ can be obtained by multiplying the weight that the arc has in $\zeta_{i+1,j-1}$ by q_j and by dividing it by q_i. It is not difficult to verify this using the same argument as in the previous observation. Similar to the second observation, the third observation can be expressed as

$$E[L(\zeta_{i,j})]\big|_{outside_{i,j} \to inside_{i+1,j-1}} = \frac{q_j}{q_i} E[L(\zeta_{i+1,j-1})]\big|_{outside_{i,j} \to inside_{i+1,j-1}}. \tag{6.10}$$

Now, by Equations (6.3), (6.4) and (6.7) we can write

$$E^{(1)}_{i,j} = E[L(\zeta_{i,j})]\big|_{inside_{i+1,j-1} \to outside_{i,j}} + E[L(\zeta_{i,j})]\big|_{\{i,j\} \to outside_{i,j}} \tag{6.11}$$

$$E^{(2)}_{i,j} = E[L(\zeta_{i,j})]\big|_{outside_{i,j} \to inside_{i+1,j-1}} + E[L(\zeta_{i,j})]\big|_{outside_{i,j} \to \{i,j\}}. \tag{6.12}$$

By combining Equation (6.9) with Equation (6.11), we obtain

$$E^{(1)}_{i,j} = \frac{q_i}{q_j} E[L(\zeta_{i+1,j-1})]\big|_{inside_{i+1,j-1} \to outside_{i,j}} + E[L(\zeta_{i,j})]\big|_{\{i,j\} \to outside_{i,j}}, \tag{6.13}$$

which, by Equation (6.8), becomes

$$\begin{aligned} E^{(1)}_{i,j} =& \frac{q_i}{q_j} E[L(\zeta_{i+1,j-1})]\big|_{inside_{i+1,j-1} \to outside_{i+1,j-1}} \\ & - \frac{q_i}{q_j} E[L(\zeta_{i+1,j-1})]\big|_{inside_{i+1,j-1} \to \{i,j\}} + E[L(\zeta_{i,j})]\big|_{\{i,j\} \to outside_{i,j}}. \end{aligned} \tag{6.14}$$

We can rewrite this to obtain the following recursion

$$E^{(1)}_{i,j} = \frac{q_i}{q_j} E^{(1)}_{i+1,j-1} - \frac{q_i}{q_j} E[L(\zeta_{i+1,j-1})]\big|_{inside_{i+1,j-1} \to \{i,j\}} + E[L(\zeta_{i,j})]\big|_{\{i,j\} \to outside_{i,j}}. \tag{6.15}$$

In an analogous way, we can create a recursive expression for $E^{(2)}_{i,j}$. By first combining Equation (6.9) with Equation (6.12), and then applying (6.8), we obtain

$$E^{(2)}_{i,j} = \frac{q_j}{q_i} E^{(2)}_{i+1,j-1} - \frac{q_j}{q_i} E[L(\zeta_{i+1,j-1})]\big|_{\{i,j\} \to inside_{i+1,j-1}} + E[L(\zeta_{i,j})]\big|_{outside_{i,j} \to \{i,j\}}. \tag{6.16}$$

Let us now focus on $E^{(3)}_{i,j}$. This term refers to the original tour ζ. Therefore, in order to get a recursive expression in terms of $E^{(3)}_{i+1,j-1}$, we must isolate the contribution to $E^{(3)}_{i,j}$ due to arcs going from $inside_{i+1,j-1}$ to $outside_{i+1,j-1}$ and vice versa. Thus, by combining Equation (6.5) with both (6.7) and (6.8) we obtain

$$E^{(3)}_{i,j} = E^{(3)}_{i+1,j-1} - E[L(\zeta)]\big|_{\{i,j\} \leftrightarrow inside_{i+1,j-1}} + E[L(\zeta)]\big|_{\{i,j\} \leftrightarrow outside_{i,j}}. \tag{6.17}$$

Now we complete the derivation by showing that it is possible to express the 'residual' terms on the right hand side of $E^{(s)}_{i,j}$, $s = 1, 2, 3$ in Equations (6.15), (6.16), (6.17) in terms of the four two-dimensional matrices Q, $\overline{Q}$, A and B, defined as follows

$$Q_{i,j} = \prod_{i}^{j} q, \qquad \overline{Q}_{i,j} = \prod_{j+1}^{i+n-1} q, \tag{6.18}$$

and

$$A_{i,k} = \sum_{r=k}^{n-1} d(i, i+r) p_i p_{i+r} Q_{i+1,i+r-1} \tag{6.19}$$

$$B_{i,k} = \sum_{r=k}^{n-1} d(i-r, i) p_{i-r} p_i Q_{i-r+1,i-1}. \tag{6.20}$$

where $1 \leq k \leq n-1$, $1 \leq i, j \leq n$. Expressing the $E^{(s)}_{i,j}$ expressions in terms of these defined matrices allows to minimize the number of calculations necessary in evaluating a neighborhood of local search moves, since Q, $\overline{Q}$, A and B must be recomputed only when a new current tour ζ is defined.

Let us now focus on the 'residual' terms on the right hand side of $E^{(s)}_{i,j}$, $s = 1, 2, 3$ in Equations (6.15), (6.16), (6.17). Recalling that $j = i + k$, the second term on the right hand side of Equation (6.15) is the following

$$-\frac{q_i}{q_j}E[L(\zeta_{i+1,j-1})]\Big|_{inside_{i+1,j-1}\to\{i,j\}} = -\frac{q_i}{q_j}\sum_{t=1}^{k-1} d(i+t,i)p_{i+t}p_i Q_{i+1,i+t-1}\overline{Q}_{i,j-1} \\ -\frac{q_i}{q_j}\sum_{t=1}^{k-1} d(i+t,i+k)p_{i+t}p_{i+k}Q_{i+1,i+t-1}. \tag{6.21}$$

The right hand side of the above equation is in two pieces. In the first piece, the factor $\overline{Q}_{i,j-1}$ may be taken out from the sum and, by applying the definition of A from Equation (6.19) to the remaining terms in the sum, we get $-\frac{q_i}{q_j}\overline{Q}_{i,j-1}(A_{i,1} - A_{i,k})$. Also the second piece can be expressed in terms of the A matrix, but it requires a bit more work. First, we substitute $(q_i/q_j)Q_{i+1,i+t-1}$ with $Q_{i,i+t-1}/q_j$. Then, we multiply and divide it by the product $q_j q_{j+1} \dots q_{i+n-1}$, and we obtain the term $Q_{j+1,i+t-1}/Q_{j,i+n-1}$, whose denominator (which is equivalent to $\overline{Q}_{i,j-1}$) may be taken out from the sum. Finally by replacing $i + t$ with $j + n - k + t$, and by applying the definition of A to the remaining terms in the sum, the second piece of the right hand side of Equation (6.21) becomes $\frac{1}{\overline{Q}_{i,j-1}}A_{j,n-k+1}$, and the whole Equation (6.21) may be rewritten as

$$-\frac{q_i}{q_j}E[L(\zeta_{i+1,j-1})]\Big|_{inside_{i+1,j-1}\to\{i,j\}} = -\frac{q_i}{q_j}\overline{Q}_{i,j-1}(A_{i,1}-A_{i,k}) - \frac{1}{\overline{Q}_{i,j-1}}A_{j,n-k+1}. \tag{6.22}$$

The rightmost term of Equation (6.15) may be written as

$$E[L(\zeta_{i,j})]\Big|_{\{i,j\}\to outside_{i,j}} = \sum_{r=1}^{n-k-1} d(i,i+k+r)p_i p_{i+k+r}Q_{i+k+1,i+k+r-1} \\ + \sum_{r=1}^{n-k-1} d(i+k,i+k+r)p_{i+k}p_{i+k+r}Q_{i+k+1,i+k+r-1}Q_{i,i+k-1}. \tag{6.23}$$

By applying the definition of A from Equation (6.19) to the right hand side of the last equation we obtain

$$E[L(\zeta_{i,j})]\Big|_{\{i,j\}\to outside_{i,j}} = \frac{q_i}{q_j}\frac{1}{Q_{i,j-1}}A_{i,k+1} + Q_{i,j-1}(A_{j,1} - A_{j,n-k}). \tag{6.24}$$

The second term on the right hand side of Equation (6.16) is the following

$$-\frac{q_j}{q_i}E[L(\zeta_{i+1,j-1})]\Big|_{\{i,j\}\to inside_{i+1,j-1}} = -\frac{q_j}{q_i}\sum_{t=1}^{k-1} d(i,i+k-t)p_i p_{i+k-t}Q_{i+k-t+1,i+k-1} \\ -\frac{q_j}{q_i}\sum_{t=1}^{k-1} d(i+k,i+k-t)p_{i+k}p_{i+k-t}Q_{i+k-t+1,i+k-1}\overline{Q}_{i+1,j}, \tag{6.25}$$

which, by applying the definition of B from Equation (6.20), becomes

$$-\frac{q_j}{q_i}E[L(\zeta_{i+1,j-1})]\Big|_{\{i,j\}\to inside_{i+1,j-1}} = -\frac{q_j}{q_i}\frac{1}{\overline{Q}_{i,j-1}}B_{i,n-k+1} - \overline{Q}_{i,j-1}(B_{j,1}-B_{j,k}). \quad (6.26)$$

The rightmost term of Equation (6.16) may be written as

$$\begin{aligned} E[L(\zeta_{i,j})]\Big|_{outside_{i,j}\to\{i,j\}} = &\sum_{r=1}^{n-k-1} d(i-r,i)p_{i-r}p_iQ_{i-r+1,i-1}Q_{i+1,i+k} \\ &+ \sum_{r=1}^{n-k-1} d(i-r,i+k)p_{i-r}p_{i+k}Q_{i-r+1,i-1}. \end{aligned} \quad (6.27)$$

By applying the definition of B from Equation (6.20) to the right hand side of the last equation we obtain

$$E[L(\zeta_{i,j})]\Big|_{outside_{i,j}\to\{i,j\}} = \frac{q_j}{q_i}Q_{i,j-1}(B_{i,1}-B_{i,n-k}) + \frac{1}{Q_{i,j-1}}B_{j,k+1}. \quad (6.28)$$

The second term on the right hand side of Equation (6.17) is the following

$$\begin{aligned} -E[L(\zeta)]\Big|_{\{i,j\}\leftrightarrow inside_{i+1,j-1}} = &-\sum_{t=1}^{k-1} d(i,i+t)p_ip_{i+t}Q_{i+1,i+t-1} \\ &-\sum_{t=1}^{k-1} d(i+k,i+t)p_{i+k}p_{i+t}Q_{i-n+k+1,i+t-1} \\ &-\sum_{t=1}^{k-1} d(i+k-t,i)p_{i+k-t}p_iQ_{i+k-t+1,i+k-1}Q_{i+k,i+n-1} \\ &-\sum_{t=1}^{k-1} d(i+k-t,i+k)p_{i+k-t}p_{i+k}Q_{i+k-t+1,i+k-1}, \end{aligned} \quad (6.29)$$

which, by applying the definition of A (Equation (6.19)) and B (Equation (6.20)), becomes

$$-E[L(\zeta)]\Big|_{\{i,j\}\leftrightarrow inside_{i+1,j-1}} = -\left(A_{i,1}-A_{i,k}+A_{j,n-k+1}+B_{i,n-k+1}+B_{j,1}-B_{j,k}\right). \quad (6.30)$$

The rightmost term of Equation (6.17) is the following

$$\begin{aligned} E[L(\zeta)]\big|_{\{i,j\}\leftrightarrow outside_{i,j}} = & \sum_{r=1}^{n-k-1} d(i-r,i)p_{i-r}p_iQ_{i-r+1,i-1} \\ & + \sum_{r=1}^{n-k-1} d(i-r,i+k)p_{i-r}p_{i+k}Q_{i-r+1,i+k-1} \\ & + \sum_{r=1}^{n-k-1} d(i,i+k+r)p_ip_{i+k+r}Q_{i+1,i+k+r-1} \\ & + \sum_{r=1}^{n-k-1} d(i+k,i+k+r)p_{i+k}p_{i+k+r}Q_{i+k+1,i+k+r-1}, \end{aligned} \tag{6.31}$$

which, by applying the definition of A (Equation (6.19)) and B (Equation (6.20)), becomes

$$E[L(\zeta)]\big|_{\{i,j\}\leftrightarrow outside_{i,j}} = B_{i,1} - B_{i,n-k} + B_{j,k+1} + A_{i,k+1} + A_{j,1} - A_{j,n-k}. \tag{6.32}$$

By substituting in Equations (6.15), (6.16) and (6.17) the appropriate terms from equations (6.22), (6.24), (6.26), (6.28), (6.30) and (6.32), we obtain the following final recursive equations for the 2-p-opt local search for $j = i + k$ and $k \geq 2$:

$$\Delta E_{i,j} = E^{(1)}_{i,j} + E^{(2)}_{i,j} - E^{(3)}_{i,j}, \tag{6.33}$$

$$\begin{aligned} E^{(1)}_{i,j} = & \frac{q_i}{q_j}E^{(1)}_{i+1,j-1} + q_i\frac{1}{Q_{i,j}}A_{i,k+1} - q_i\overline{Q}_{i,j}(A_{i,1} - A_{i,k}) \\ & - \frac{1}{q_j}\frac{1}{\overline{Q}_{i,j}}A_{j,n-k+1} + \frac{1}{q_j}Q_{i,j}(A_{j,1} - A_{j,n-k}), \end{aligned} \tag{6.34}$$

$$\begin{aligned} E^{(2)}_{i,j} = & \frac{q_j}{q_i}E^{(2)}_{i+1,j-1} - \frac{1}{q_i}\frac{1}{\overline{Q}_{i,j}}B_{i,n-k+1} + \frac{1}{q_i}Q_{i,j}(B_{i,1} - B_{i,n-k}) \\ & + q_j\frac{1}{Q_{i,j}}B_{j,k+1} - q_j\overline{Q}_{i,j}(B_{j,1} - B_{j,k}), \end{aligned} \tag{6.35}$$

$$\begin{aligned} E^{(3)}_{i,j} = E^{(3)}_{i+1,j-1} & -A_{i,1} + A_{i,k} + A_{i,k+1} + A_{j,1} - A_{j,n-k} - A_{j,n-k+1} \\ & + B_{i,1} - B_{i,n-k} - B_{i,n-k+1} - B_{j,1} + B_{j,k} + B_{j,k+1}. \end{aligned} \tag{6.36}$$

For $k = 1$, we can express the three components of $\Delta E_{i,i+1}$ (6.3), (6.4) and (6.5) in terms of A and B and obtain the following equations

$$E^{(1)}_{i,i+1} = \frac{1}{q_{i+1}} A_{i,2} + q_i(A_{i+1,1} - A_{i+1,n-1}) \tag{6.37}$$

$$E^{(2)}_{i,i+1} = q_{i+1}(B_{i,1} - B_{i,n-1}) + \frac{1}{q_i} B_{i+1,2} \tag{6.38}$$

$$E^{(3)}_{i,i+1} = A_{i,2} + A_{i+1,1} - A_{i+1,n-1} + B_{i,1} - B_{i,n-1} + B_{i+1,2}. \tag{6.39}$$

For $j = i$, $\Delta E_{i,i} = 0$ since $\zeta_{i,i} = \zeta$. It is still necessary, though, to compute the three components $E^{(s)}_{i,i}$, $s = 1, 2, 3$ separately, in order to initiate the recursion $E^{(s)}_{i-1,i+1}$, $s = 1, 2, 3$. By expressing (6.3), (6.4) and (6.5) in terms of A and B, we obtain

$$E^{(1)}_{i,i} = A_{i,1} \tag{6.40}$$

$$E^{(2)}_{i,i} = B_{i,1} \tag{6.41}$$

$$E^{(3)}_{i,i} = A_{i,1} + B_{i,1}. \tag{6.42}$$

Note that $\Delta E_{i,i} = E^{(1)}_{i,i} + E^{(2)}_{i,i} - E^{(3)}_{i,i} = 0$, as expected. It is possible to verify that when $p_i = p$ and $q_i = q = 1 - p$, we obtain the same recursive $\Delta E_{i,j}$ expressions as for the homogeneous PTSP in [42].

The time required for evaluating one single 2-p-opt move by means of the recursive equations is $O(1)$, given that the matrices A, B, Q and $\overline{Q}$ are known. For each tour, there are $O(n^2)$ possible 2-p-opt moves, and the time required to compute the matrices A, B, Q and $\overline{Q}$ is $O(n^2)$. Thus, the recursive calculations allow us to evaluate all possible 2-p-opt moves from the current solution in $O(n^2)$ time. With a straightforward implementation that does not utilize recursion, evaluating the 2-p-opt neighborhood would require $O(n^4)$ time instead of $O(n^2)$. Since a local search procedure involves many iterations, these savings can lead to much better solutions.

The expression for $\Delta E_{i,j}$ derived by Chervi in [62] is of the form $\Delta E_{i,j} = \Delta E_{i+1,j-1} + \xi$. This greatly differs from our set of recursive equations. First, the ξ term, as derived in [62], is not computable in $O(1)$ time but is $O(n)$. Second, the expression derived in [62] is incorrect since it reduces to the incorrect 2-p-opt expression for the homogeneous PTSP published in [27] when all customer probabilities are equal.

6.4.2.2 The cost of 1-shift moves

Consider, without loss of generality, a tour $\zeta = (1, 2, ..., i, i+1, ..., j, j+1, ..., n)$, and denote by $\zeta_{i,j}$ a tour obtained from ζ by moving node i to the position of node j and shifting backwards the nodes $(i+1, ..., j)$, where $i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, n\}$, and $i \neq j$ (see Figure 6.2). Note that the shifted section may include n. Let $\Delta E_{i,j}$ denote the change in the expected tour length $E[L(\zeta_{i,j})] - E[L(\zeta)]$. In the following, the correct recursive formula for $\Delta E_{i,j}$ is derived for the 1-shift neighborhood.

Let $j = i + k$. For $k = 1$, the tour $\zeta_{i,i+1}$ obtained by 1-shift is the same as the

one obtained by 2-p-opt, and the expression for $\Delta E_{i,i+1}$ may be derived by applying the equations derived for the 2-p-opt. By summing Equations (6.37), (6.38) and by subtracting Equation (6.39) we find

$$\begin{aligned}\Delta E_{i,i+1} = &\left(\frac{1}{q_{i+1}} - 1\right) A_{i,2} + (q_{i+1} - 1)(B_{i,1} - B_{i,n-1}) \\ &+ (q_i - 1)\left(A_{i+1,1} - A_{i+1,n-1}\right) + \left(\frac{1}{q_i} - 1\right) B_{i+1,2}.\end{aligned} \tag{6.43}$$

We will now focus on the more general case where $2 \leq k \leq n-2$. Again, the case where $k = n-1$ can be neglected because it does not produce any change to the tour ζ. We re-define the notions of *inside*, *outside* and the contributions to the change in expected tour length adapting them for the 1-shift.

Definition 12 $\Delta E_{i,j}\big|_{S \leftrightarrow T} = E[L(\zeta_{i,j})]\big|_{T \leftrightarrow S} - E[L(\zeta)]\big|_{T \leftrightarrow S}$. *This is similar to Definition 9, the only difference being the meaning of the a priori tour $\zeta_{i,j}$, that here is obtained from ζ by a 1-shift move.*

Definition 13 $inside_{i,j} = \{i+1, \ldots, j\}$, *that is, the section of ζ that is shifted to obtain $\zeta_{i,j}$*

Definition 14 $outside_{i,j} = N \setminus (inside_{i,j} \cup \{i\})$

It is not difficult to verify that the weights on arcs between outside nodes and arcs between inside nodes again do not change as a result of the shift. Therefore, the only contribution to $\Delta E_{i,j}$ is given by the change in weight placed on arcs between $inside_{i,j} \cup \{i\}$ nodes and $outside_{i,j}$ nodes, and on arcs between node $\{i\}$ and $inside_{i,j}$ nodes, that is

$$\Delta E_{i,j} = \Delta E_{i,j}\big|_{(inside_{i,j} \cup \{i\}) \leftrightarrow outside_{i,j}} + \Delta E_{i,j}\big|_{\{i\} \leftrightarrow inside_{i,j}}. \tag{6.44}$$

In the following, we derive a recursive expression for each of the two components of $\Delta E_{i,j}$. Let

$$\Delta E_{i,j}\big|_{(inside_{i,j} \cup \{i\}) \leftrightarrow outside_{i,j}} = \Delta E_{i,j-1}\big|_{(inside_{i,j-1} \cup \{i\}) \leftrightarrow outside_{i,j-1}} + \delta, \tag{6.45}$$

and

$$\Delta E_{i,j}\big|_{\{i\} \leftrightarrow inside_{i,j}} = \Delta E_{i,j-1}\big|_{\{i\} \leftrightarrow inside_{i,j-1}} + \gamma. \tag{6.46}$$

Then, by Equation (6.44), we can write the following recursive expression

$$\Delta E_{i,j} = \Delta E_{i,j-1} + \delta + \gamma. \tag{6.47}$$

Now we complete the derivation by showing that it is possible to express the 'residual' terms δ and γ of Equation (6.47) in terms of the already defined matrices A, B, and Q, $\overline{Q}$, defined as follows

$$Q_{i,j} = \prod_{i+1}^{j} q, \qquad \overline{Q}_{i,j} = \prod_{j+1}^{i+n-1} q. \tag{6.48}$$

Expressing the $\Delta E_{i,j}$ expressions in terms of these defined matrices allows to minimize the number of calculations necessary in evaluating a neighborhood of local search moves.

We first re-define the matrices A (Equation (6.19)) and B (Equation (6.20)) in terms of the matrix Q (Equation (6.48))

$$A_{i,k} = \sum_{r=k}^{n-1} d(i, i+r) p_i p_{i+r} Q_{i,i+r-1} \tag{6.49}$$

$$B_{i,k} = \sum_{r=k}^{n-1} d(i-r, i) p_{i-r} p_i Q_{i-r,i-1}. \tag{6.50}$$

Let us now focus on the 'residual' term δ from Equation (6.45). The contribution to $\Delta E_{i,j}$ due to arcs between $inside_{i,j} \cup \{i\}$ and $outside_{i,j}$ for $\zeta_{i,j}$ is the following

$$\begin{aligned}
\Delta E_{i,j}\big|_{(inside_{i,j}\cup\{i\})\leftrightarrow outside_{i,j}} = & \sum_{r=1}^{n-k-1} [d(i-r, i) p_{i-r} p_i Q_{i-r,i-1} (Q_{i,j} - 1) \\
& + d(i, i+k+r) p_i p_{i+k+r} Q_{i+k,i+k+r-1} (1 - Q_{i,j})] \\
& + \sum_{t=1}^{k} \sum_{r=1}^{n-k-1} [d(i-r, i+t) p_{i-r} p_{i+t} Q_{i-r,i-1} Q_{i,i+t-1} (1 - q_i) \\
& + d(i+k-t+1, i+k+r) p_{i+k-t+1} p_{i+k+r} Q_{i+k-t+1,i+k} Q_{i+k,i+k+r-1} (q_i - 1)] ,
\end{aligned} \tag{6.51}$$

while the contribution to $\Delta E_{i,j-1}$ due to arcs between $inside_{i,j-1} \cup \{i\}$ and $outside_{i,j-1}$ for $\zeta_{i,j-1}$ is

$$\begin{aligned}
\Delta E_{i,j-1}\big|_{(inside_{i,j-1}\cup\{i\})\leftrightarrow outside_{i,j-1}} = & \sum_{r=1}^{n-k} [d(i-r, i) p_{i-r} p_i Q_{i-r,i-1} (Q_{i,j-1} - 1) \\
& + d(i, i+k-1+r) p_i p_{i+k-1+r} Q_{i+k-1,i+k+r-2} (1 - Q_{i,j-1})] \\
& + \sum_{t=1}^{k-1} \sum_{r=1}^{n-k} [d(i-r, i+t) p_{i-r} p_{i+t} Q_{i-r,i-1} Q_{i,i+t-1} (1 - q_i) \\
& + d(i+k-t, i+k+r-1) p_{i+k-t} p_{i+k+r-1} Q_{i+k-t,i+k-1} Q_{i+k-1,i+k+r-2} (q_i - 1)] .
\end{aligned} \tag{6.52}$$

The difference between $\Delta E_{i,j}\big|_{(inside_{i,j}\cup\{i\})\leftrightarrow outside_{i,j}}$ and $\Delta E_{i,j-1}\big|_{(inside_{i,j-1}\cup\{i\})\leftrightarrow outside_{i,j-1}}$ will only involve arcs which are connected to nodes i and j, that is,

$$\begin{aligned}\delta = \quad & \text{terms with } i \text{ and } j \text{ in } \Delta E_{i,j}\big|_{(inside_{i,j}\cup\{i\})\leftrightarrow outside_{i,j}} \\ & - \text{ terms with } i \text{ and } j \text{ in } \Delta E_{i,j-1}\big|_{(inside_{i,j-1}\cup\{i\})\leftrightarrow outside_{i,j-1}}.\end{aligned} \tag{6.53}$$

So, by extracting the terms which contain the appropriate arcs from Equation (6.51) and Equation (6.52) and by expressing them in terms of the matrices A (Equation (6.49)) and B (Equation (6.50)) we obtain the following expression for δ

$$\begin{aligned}\delta = & (1-Q_{i,j})\left[\frac{1}{Q_{i,j}}A_{i,k+1} - (B_{i,1} - B_{i,n-k})\right] \\ & + (q_i - 1)\left[(A_{j,1} - A_{j,n-k}) - q_i^{-1}B_{j,k+1}\right] \\ & - (1-Q_{i,j-1})\left[\frac{1}{Q_{i,j-1}}A_{i,k} - (B_{i,1} - B_{i,n-k+1})\right] \\ & - (q_i - 1)\left[(B_{j,1} - B_{j,k}) - q_i^{-1}A_{j,n-k+1}\right],\end{aligned} \tag{6.54}$$

which completes the recursive expression of Equation (6.45). Let us now focus on the 'residual' term γ from Equation (6.46). The contribution to $\Delta E_{i,j}$ due to arcs between $\{i\}$ and $inside$ is the following

$$\begin{aligned}\Delta E_{i,j}\big|_{\{i\}\leftrightarrow inside_{i,j}} = \left(\overline{Q}_{i,j} - 1\right)\sum_{t=1}^{k}[& d(i,i+t)p_i p_{i+t} Q_{i,i+t-1} \\ & - d(i+k-t+1,i)p_{i+k-t+1}p_i Q_{i+k-t+1,i+k}],\end{aligned} \tag{6.55}$$

while the contribution to $\Delta E_{i,j-1}$ due to arcs between $\{i\}$ and $inside_{i,j-1}$ for $\zeta_{i,j-1}$ is

$$\begin{aligned}\Delta E_{i,j-1}\big|_{\{i\}\leftrightarrow inside_{i,j-1}} = \left(\overline{Q}_{i,j-1} - 1\right)\sum_{t=1}^{k-1}[& d(i,i+t)p_i p_{i+t} Q_{i,i+t-1} \\ & - d(i+k-t,i)p_{i+k-t}p_i Q_{i+k-t,i+k-1}].\end{aligned} \tag{6.56}$$

Now, by subtracting Equation (6.56) from Equation (6.55) and by applying the definition of A (Equation (6.49)), and B (Equation (6.50)), we obtain the following expression for γ

$$\begin{aligned}\gamma = & \left(\overline{Q}_{i,j} - 1\right)\left[A_{i,1} - A_{i,k+1} - \frac{1}{\overline{Q}_{i,j-1}}B_{i,n-k})\right] \\ & + \left(1 - \overline{Q}_{i,j-1}\right)\left[A_{i,1} - A_{i,k} - \frac{1}{\overline{Q}_{i,j-1}}B_{i,n-k+1}\right],\end{aligned} \tag{6.57}$$

which completes the recursive expression of Equation (6.46).

By substituting the expression for δ (Equation (6.54)) and γ (Equation (6.57)) in Equation (6.47), we obtain the following final recursive equations for the 1-shift local search for $j = i + k$ and $k \geq 2$:

$$\begin{aligned}\Delta E_{i,j} = \Delta E_{i,j-1} &+ (\overline{Q}_{i,j} - \frac{1}{Q_{i,j}})(q_j A_{i,k} - A_{i,k+1}) \\ &+ (\frac{1}{\overline{Q}_{i,j}} - Q_{i,j})(B_{i,n-k} - \frac{1}{q_j} B_{i,n-k+1}) \\ &+ (1 - \frac{1}{q_j}) Q'_{i,j} B_{i,1} + (\frac{1}{q_i} - 1) B_{j,k+1} \\ &+ (1 - q_j)\overline{Q}_{i,j} A_{i,1} + (1 - \frac{1}{q_i}) A_{j,n-k+1} \\ &+ (q_i - 1)(A_{j,1} - A_{j,n-k}) \\ &+ (1 - q_i)(B_{j,1} - B_{j,k}).\end{aligned} \tag{6.58}$$

Similarly to the 2-p-opt, the time required for evaluating one single 1-shift move by means of the recursive equations is $O(1)$, given that the matrices A, B, Q and $\overline{Q}$ are known. For each tour, there are $O(n^2)$ possible 1-shift moves, and the time required to compute the matrices A, B, Q and $\overline{Q}$ is $O(n^2)$. Thus, the recursive calculations allow us to evaluate all possible 1-shift moves from the current solution in $O(n^2)$ time. With a straightforward implementation that does not utilize recursion, evaluating the 1-shift neighborhood would require $O(n^4)$ time instead of $O(n^2)$. Since a local search procedure involves many iterations, these savings can lead to much better solutions.

The expression for $\Delta E_{i,j}$ derived by Chervi in [62] is of the form $\Delta E_{i,j} = \Delta E_{i,j-1} + \xi'$. Again, the ξ' term, as derived in [62], is not computable in $O(1)$ time but requires $O(n)$. This expression is also incorrect since it reduces to the incorrect 1-shift expression for the homogeneous PTSP published in [27] when all customer probabilities are equal.

6.4.3 Derivation of local search costs for the homogeneous PTSP

In this section we derive the cost of 2-p-opt and 1-shift moves for the homogeneous PTSP, in a similar way as we did for the heterogeneous case in the previous section. The fact that we obtain cost expressions that are equivalent to those that one would obtain from the heterogeneous PTSP cost expression by letting $p_i = p$ and $q_i = q$ for $i = 1, 2, \ldots, n$, is an additional check of their correctness.

6.4.3.1 The cost of 2-p-opt moves

Consider, without loss of generality, a tour $\zeta = (1, 2, ..., i, i + 1, ..., j, j + 1, ..., n)$, and denote by $\zeta_{i,j}$ a tour obtained by reversing a section $(i, i + 1, ..., j)$ of ζ, where $i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, n\}$, and $i \neq j$ (see Figure 6.1) (see Figure 6.1). Note that if $j < i$, the reversed section includes n. Let $\Delta E_{i,j}$ denote the change in the expected

length $E[L(\zeta_{i,j})] - E[L(\zeta)]$. In the following, the correct recursive formula for $\Delta E_{i,j}$ is derived for the 2-p-opt move set. This derivation consists in first developing a formula for $\Delta E_{i,j}$ and then finding a recursive form of this.

Let $j = i+k$ (we are using the notation expressed by (4.1)). For $k = 1$ we believe the expression given in [24] and in [27] for $\Delta E_{i,j}$ (Equation (6.76)) is correct. Therefore, we focus on the case $k \geq 2$ (and $k \leq n-2$). Let $S, T \subseteq N$ be subsets of nodes, with λ representing any a priori tour, and $\lambda(i)$ representing the customer in the ith position on this tour such that $\lambda = (\lambda(1), \lambda(2), \ldots, \lambda(n))$. The contribution to the expected cost of λ due to the arcs from the nodes in S to the nodes in T, denoted by $E[L(\lambda)]\big|_{T \to S}$ and introduced by Definition 7 in section 6.4.2.1, assumes in case of homogeneous probabilities the following form:

$$E[L(\lambda)]\big|_{T \to S} = \sum_{\lambda(i) \in S, \lambda(j) \in T, i \neq j} d(\,\lambda(i),\, \lambda(j)\,) p^2 q^{j-i-1}. \tag{6.59}$$

Definition 8 and Definition 9 introduced section 6.4.2.1 must be adapted accordingly. Given that the set of nodes of tour ζ is partitioned in the the two sets $inside_{i,j}$ and $outside_{i,j}$ (see Definition 10 and Definition 11), the change in expected tour length $\Delta E_{i,j}$ may be expressed as

$$\Delta E_{i,j} = \Delta E_{i,j}\big|_{inside_{i,j} \to inside_{i,j}} + \Delta E_{i,j}\big|_{outside_{i,j} \to outside_{i,j}} + \Delta E_{i,j}\big|_{inside_{i,j} \leftrightarrow outside_{i,j}}. \tag{6.60}$$

It is not difficult to verify that, as in the heterogeneous PTSP, the contributions to $\Delta E_{i,j}$ due to $\Delta E_{i,j}\big|_{inside_{i,j} \to inside_{i,j}}$ and to $\Delta E_{i,j}\big|_{outside_{i,j} \to outside_{i,j}}$ are zero, therefore we can write

$$\Delta E_{i,j} = \Delta E_{i,j}\big|_{inside_{i,j} \leftrightarrow outside_{i,j}}. \tag{6.61}$$

Now, the contribution from arcs between $inside_{i,j}$ and $outside_{i,j}$ to the expected tour length of the undisturbed tour ζ is

$$E[L(\zeta)]\big|_{inside_{i,j} \leftrightarrow outside_{i,j}} = p^2 \sum_{t=1}^{k+1} q^{t-1} \sum_{r=1}^{n-k-1} q^{r-1} [d(i-r, i+t-1) + d(j-t+1, j+r)]. \tag{6.62}$$

In the disturbed tour $\zeta_{i,j}$, the contribution of the same arcs is

$$E[L(\zeta_{i,j})]\big|_{inside_{i,j} \leftrightarrow outside_{i,j}} = p^2 \sum_{t=1}^{k+1} q^{t-1} \sum_{r=1}^{n-k-1} q^{r-1} [d(i+t-1, j+r) + d(i-r, j-t+1)]. \tag{6.63}$$

By subtracting (6.62) from (6.63), we obtain

$$\begin{aligned} \Delta E_{i,j} = p^2 \sum_{t=1}^{k+1} q^{t-1} \sum_{r=1}^{n-k-1} q^{r-1} [&d(i+t-1, j+r) + d(i-r, j-t+1) \\ &- d(i-r, i+t-1) - d(j-t+1, j+r)]. \end{aligned} \tag{6.64}$$

Consider now the case when the section from $i+1$ to $j-1$ of ζ is reversed, leading to the tour $\zeta_{i+1,j-1}$. Then the difference in expected length is

$$\begin{aligned}\Delta E_{i+1,j-1} = p^2 \sum_{t=1}^{k-1} q^{t-1} \sum_{r=1}^{n+1-k} q^{r-1}[&d(i+t, j+r-1) + d(i+1-r, j-t) \\ &-d(i-r+1, i+t) - d(j-t, j+r-1)].\end{aligned} \tag{6.65}$$

From the two above expressions (6.64) and (6.65) it is possible to extract a recursive equation which relates $\Delta E_{i,j}$ to $\Delta E_{i+1,j-1}$

$$\Delta E_{i,j} = \Delta E_{i+1,j-1} + \epsilon. \tag{6.66}$$

Observe that the difference between $\Delta E_{i,j}$ and $\Delta E_{i+1,j-1}$ will only involve arcs which are connected to nodes i and j, that is,

$$\begin{aligned}\epsilon = \quad &\text{terms with } i \text{ and } j \text{ in } \Delta E_{i,j} \\ &- \text{terms with } i \text{ and } j \text{ in } \Delta E_{i+1,j-1}.\end{aligned} \tag{6.67}$$

So, let us extract the terms which contain the appropriate arcs from (6.64) and (6.65). The terms with i in $\Delta E_{i,j}$ are (from Equation (6.64))

$$p^2 \sum_{r=1}^{n-k-1} q^{r-1}[(1-q^k)d(i, j+r) + (q^k-1)d(i-r, i)], \tag{6.68}$$

and the terms with j in $\Delta E_{i,j}$ are (from Equation (6.64))

$$p^2 \sum_{r=1}^{n-k-1} q^{r-1}[(q^k-1)d(j, j+r) + (1-q^k)d(i-r, j)]. \tag{6.69}$$

The terms with i in $\Delta E_{i+1,j-1}$ are (from Equation (6.65))

$$p^2 \sum_{t=1}^{k-1} q^{t-1}[(q^{n-k}-1)d(i+t, i) + (1-q^{n-k})d(i, j-t)], \tag{6.70}$$

and the terms with j in $\Delta E_{i+1,j-1}$ are (from Equation (6.65))

$$p^2 \sum_{t=1}^{k-1} q^{t-1}[(1-q^{n-k})d(i+t, j) + (q^{n-k}-1)d(j-t, j)]. \tag{6.71}$$

By subtracting (6.70) and (6.71) from (6.68) and (6.69) we obtain

$$\begin{aligned}\epsilon = p^2\{(1-q^k) &\sum_{r=1}^{n-k-1} q^{r-1}[d(i, j+r) - d(i-r, i) + d(j, j+r) - d(i-r, j)] \\ +(1-q^{n-k}) &\sum_{r=1}^{k-1} q^{r-1}[d(i+r, i) - d(i, j-r) - d(i+r, j) + d(j-r, j)]\}.\end{aligned} \tag{6.72}$$

Now, the two dimensional matrices of partial results A and B defined in the heterogeneous case by Equations (6.19) and (6.20), assume here the following form (as also given by Bertsimas [27])

$$A_{i,k} = \sum_{r=k}^{n-1} q^{r-1} d(i, i+r) \quad \text{and} \quad B_{i,k} = \sum_{r=k}^{n-1} q^{r-1} d(i-r, i), \quad 1 \leq k \leq n-1, \quad 1 \leq i \leq n. \tag{6.73}$$

By expressing Equation (6.72) in terms of A and B as defined above we obtain

$$\begin{aligned} \epsilon =& p^2[(q^{-k}-1)A_{i,k+1} + (q^k-1)(B_{i,1} - B_{i,n-k}) \\ &+ (q^k-1)(A_{j,1} - A_{j,n-k}) + (q^{-k}-1)B_{j,k+1} \\ &+ (1-q^{n-k})(A_{i,1} - A_{i,k}) + (1-q^{k-n})B_{i,n-k+1} \\ &+ (1-q^{k-n})A_{j,n-k+1} + (1-q^{n-k})(B_{j,1} - B_{j,k})]. \end{aligned} \tag{6.74}$$

Finally, the above expression, together with Equation (6.66) leads to the following recursive equation for the change in expected tour length

$$\begin{aligned} \Delta E_{i,j} = \Delta E_{i+1,j-1} &+ p^2[(q^{-k}-1)A_{i,k+1} + (q^k-1)(B_{i,1} - B_{i,n-k}) \\ &+ (q^k-1)(A_{j,1} - A_{j,n-k}) + (q^{-k}-1)B_{j,k+1} \\ &+ (1-q^{n-k})(A_{i,1} - A_{i,k}) + (1-q^{k-n})B_{i,n-k+1} \\ &+ (1-q^{k-n})A_{j,n-k+1} + (1-q^{n-k})(B_{j,1} - B_{j,k})]. \end{aligned} \tag{6.75}$$

This expression differs from the one proposed by Bertsimas in [24] in the sign of the last four terms inside the squared brackets, and from the expression proposed by Bertsimas-Howell in [27] also in the first term on the right side. Recall that Equation (6.75) is valid for $k \geq 2$. In the case $k = 1$ the expression for $\Delta E_{i,j}$, as published in [24] and [27], is correct, and it is the following

$$\Delta E_{i,i+1} = p^3[q^{-1}A_{i,2} - (B_{i,1} - B_{i,n-1}) - (A_{i+1,1} - A_{i+1,n-1}) + q^{-1}B_{i+1,2}]. \tag{6.76}$$

The computational complexity of evaluating the 2-p-opt neighborhood is $O(n^2)$, the motivation being the same as for the heterogeneous PTSP case (see the end of section 6.4.2.1). In practice, the homogeneous case will be faster, since one does not need to compute the matrices Q and $\overline{Q}$.

6.4.3.2 The cost of 1-shift moves

Consider, without loss of generality, a tour $\zeta = (1, 2, \ldots, i, i+1, \ldots, j, j+1, \ldots, n)$, and denote by $\zeta_{i,j}$ a tour obtained from ζ by moving node i to position j and shifting backwards one space the nodes $i+1, ..., j$, where $i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, n\}$, and $i \neq j$ (see Figure 6.2). Let $\Delta E_{i,j}$ denote the change in the expected length $E[L(\zeta_{i,j})] -$

$E[L(\zeta)]$. In the following, the correct recursive formula for $\Delta E_{i,j}$ is derived for the 1-shift move set. This derivation follows a line similar to the one for the 2-p-opt.

Let $j = i + k$ (we are using the notation expressed by (4.1)). Note that, for $k = 1$, the tour $\zeta_{i,j}$ obtained by 1-shift is the same as the one obtained by 2-p-opt, for which we believe [24] and [27] gives the correct expression (Equation (6.43)). Therefore, we focus on the case $k \geq 2$ (and $k \leq n - 2$).

Based on the node partitions $inside_{i,j}$ and $outside_{i,j}$ introduced for the 1-shift by Definition 13 and 14, it is not difficult to verify that, like in the heterogeneous case, the weights on arcs between outside nodes and arcs between inside nodes again do not change as a result of the shift. Therefore, the only contribution to $\Delta E_{i,j}$ is given by the change in weight placed on arcs between $inside_{i,j} \cup \{i\}$ nodes and $outside_{i,j}$ nodes, and on arcs between node $\{i\}$ and $inside_{i,j}$ nodes, that is

$$\Delta E_{i,j} = \Delta E_{i,j}\Big|_{(inside_{i,j}\cup\{i\})\leftrightarrow outside_{i,j}} + \Delta E_{i,j}\Big|_{\{i\}\leftrightarrow inside_{i,j}}. \tag{6.77}$$

In the following, we derive a recursive expression for each of the two components of $\Delta E_{i,j}$. Let

$$\Delta E_{i,j}\Big|_{(inside_{i,j}\cup\{i\})\leftrightarrow outside_{i,j}} = \Delta E_{i,j-1}\Big|_{(inside_{i,j-1}\cup\{i\})\leftrightarrow outside_{i,j-1}} + \delta, \tag{6.78}$$

and

$$\Delta E_{i,j}\Big|_{\{i\}\leftrightarrow inside_{i,j}} = \Delta E_{i,j-1}\Big|_{\{i\}\leftrightarrow inside_{i,j-1}} + \gamma. \tag{6.79}$$

Then, by Equation (6.77), we can write the following recursive expression

$$\Delta E_{i,j} = \Delta E_{i,j-1} + \delta + \gamma. \tag{6.80}$$

Now we complete the derivation by showing that it is possible to express the 'residual' terms δ and γ of Equation (6.80) in terms of the already defined matrices A, B (Equation (6.73)).

Let us now focus on the 'residual' term δ from Equation (6.78). The contribution to $\Delta' E_{i,j}$ due to arcs between $inside_{i,j} \cup \{i\}$ and $outside_{i,j}$ for ζ is the following

$$E[L(\zeta)]\Big|_{(inside_{i,j}\cup\{i\})\leftrightarrow outside_{i,j}} = p^2 \sum_{t=1}^{k+1} q^{t-1} \sum_{r=1}^{n-k-1} q^{r-1}[d(i-r, i+t-1)+d(j-t+1, j+r)]. \tag{6.81}$$

In the disturbed tour $\zeta_{i,j}$, the contribution of the same arcs is

$$\begin{aligned} E[L(\zeta_{i,j})]\Big|_{(inside_{i,j}\cup\{i\})\leftrightarrow outside_{i,j}} &= p^2 \sum_{r=1}^{n-k-1} q^{r-1}[q^k d(i-r, i) + d(i, j+r)] \\ &+ p^2 \sum_{t=1}^{k} q^{t-1} \sum_{r=1}^{n-k-1} q^{r-1}[qd(j-t+1, j+r) + d(i-r, i+t)]. \end{aligned} \tag{6.82}$$

By subtracting (6.81) from (6.82), we obtain

$$\Delta E_{i,j}\Big|_{(inside_{i,j}\cup\{i\})\leftrightarrow outside_{i,j}} = p^2 \sum_{r=1}^{n-k-1} q^{r-1}[(q^k-1)d(i-r,i)+(1-q^k)d(i,j+r)] \\ +p^2\sum_{t=1}^{k} q^{t-1} \sum_{r=1}^{n-k-1} q^{r-1}[(1-q)d(i-r,i+t)+(q-1)d(j-t+1,j+r)]. \tag{6.83}$$

Consider now the case where node i is shifted to position $j-1$, leading to a perturbed tour $\zeta_{i,j-1}$. Then the contribution to $\Delta' E_{i,j-1}$ due to arcs between $inside_{i,j-1}\cup\{i\}$ and $outside_{i,j-1}$ for $\zeta_{i,j-1}$ is

$$\Delta E_{i,j-1}\Big|_{(inside_{i,j-1}\cup\{i\})\leftrightarrow outside_{i,j-1}} = \\ p^2\sum_{r=1}^{n-k} q^{r-1}[(q^{k-1}-1)d(i-r,i)+(1-q^{k-1})d(i,j-1+r)] \\ +p^2\sum_{t=1}^{k-1} q^{t-1}\sum_{r=1}^{n-k} q^{r-1}[(1-q)d(i-r,i+t)+(q-1)d(j-t,j-1+r)]. \tag{6.84}$$

Now, here, like in the heterogeneous PTSP, the difference between $\Delta E_{i,j}\Big|_{(inside_{i,j}\cup\{i\})\leftrightarrow outside_{i,j}}$ and $\Delta E_{i,j-1}\Big|_{(inside_{i,j-1}\cup\{i\})\leftrightarrow outside_{i,j-1}}$ will only involve arcs which are connected to nodes i and j, that is,

$$\delta = \begin{array}{l}\text{terms with } i \text{ and } j \text{ in } \Delta E_{i,j}\Big|_{(inside_{i,j}\cup\{i\})\leftrightarrow outside_{i,j}} \\ -\text{ terms with } i \text{ and } j \text{ in } \Delta E_{i,j-1}\Big|_{(inside_{i,j-1}\cup\{i\})\leftrightarrow outside_{i,j-1}}.\end{array} \tag{6.85}$$

So, let us extract the terms which contain the appropriate arcs from (6.83) and (6.84). The terms with i and j in $\Delta E_{i,j}\Big|_{(inside_{i,j}\cup\{i\})\leftrightarrow outside_{i,j}}$ are (from (6.83)):

$$p^2\sum_{r=1}^{n-k-1} q^{r-1}[(q^k-1)d(i-r,i)+(1-q^k)d(i,j+r) \\ +(q-1)d(j,j+r)+(q^{k-1}-q^k)d(i-r,j)], \tag{6.86}$$

and the terms with i and j in $\Delta E_{i,j-1}\Big|_{(inside_{i,j-1}\cup\{i\})\leftrightarrow outside_{i,j-1}}$ are (from (6.84)):

$$p^2\sum_{t=1}^{k-1} q^{t-1}[q^{n-k-1}(1-q)d(j,i+t)+(q-1)d(j-t,j)] \\ +p^2\sum_{r=1}^{n-k} q^{r-1}[(q^{k-1}-1)d(i-r,i)+(1-q^{k-1})d(i,j+r-1)]. \tag{6.87}$$

Now, by subtracting (6.87) from (6.86) and by applying the definition of A and B (6.73) we obtain the following expression

$$\begin{aligned}\delta = p^2[&(q^k-1)(B_{i,1}-B_{i,n-k})+(q^{-k}-1)A_{i,k+1}\\ &+(1-q^{k-1})(B_{i,1}-B_{i,n-k+1})+(1-q^{-k+1})A_{i,k}\\ &+(q-1)(A_{j,1}-A_{j,n-k})+(q^{-1}-1)B_{j,k+1}\\ &+(1-q)(B_{j,1}-B_{j,k})+(1-q^{-1})A_{j,n-k+1}],\end{aligned} \tag{6.88}$$

which completes the recursive expression (6.78). Let us now focus on the 'residual' term γ from Equation (6.79). The contribution from arcs between $\{i\}$ and $inside_{i,j}$ to the expected tour length of the undisturbed tour ζ is

$$E[L(\zeta)]\big|_{\{i\}\leftrightarrow inside_{i,j}} = p^2\sum_{t=1}^{k}[q^{t-1}d(i,i+t)+q^{n-k-1}q^{t-1}d(j-t+1,i)]. \tag{6.89}$$

In the disturbed tour $\zeta_{i,j}$, the contribution of the same arcs is

$$E[L(\zeta_{i,j})]\big|_{\{i\}\leftrightarrow inside_{i,j}} = p^2\sum_{t=1}^{k}[q^{t-1}d(j+1-t,i)+q^{n-k-1}q^{t-1}d(i,i+t)]. \tag{6.90}$$

By subtracting (6.89) from (6.90), we obtain

$$\Delta E_{i,j}\big|_{\{i\}\leftrightarrow inside_{i,j}} = p^2(q^{n-k-1}-1)\sum_{t=1}^{k}q^{t-1}[d(i,i+t)-d(j+1-t,i)]. \tag{6.91}$$

Consider now the case where node i is shifted to position $j-1$, leading to a perturbed tour $\zeta_{i,j-1}$. Then the contribution to $\Delta' E_{i,j-1}$ due to arcs between $\{i\}$ and $inside_{i,j-1}$ is

$$\Delta E_{i,j-1}\big|_{\{i\}\leftrightarrow inside_{i,j-1}} = p^2(q^{n-k}-1)\sum_{t=1}^{k-1}q^{t-1}[d(i,i+t)-d(j-t,i)]. \tag{6.92}$$

Now, by subtracting (6.92) from (6.91) and by applying the definition of A and B (6.73) we obtain the following expression

$$\begin{aligned}\gamma = p^2[&(q^{n-k-1}-1)(A_{i,1}-A_{i,k+1})+(q^{k+1-n}-1)B_{i,n-k}\\ &+(1-q^{n-k})(A_{i,1}-A_{i,k})+(1-q^{k-n})B_{i,n-k+1}],\end{aligned} \tag{6.93}$$

which completes the recursive expression (6.79). By substituting the expression for δ (Equation (6.88)) and γ (Equation (6.93)) in Equation (6.80), we obtain the following recursive equation for the 1-shift local search for $j = i+k$ and $k \geq 2$:

$$\begin{aligned}\Delta E_{i,i+k} =& \Delta E_{i,i+k-1} + p^2[(q^{-k} - q^{n-k-1})A_{i,k+1} \\ &+(q^k - q^{k-1})B_{i,1} + (q^{k+1-n} - q^k)B_{i,n-k} \\ &+(q^{n-k-1} - q^{n-k})A_{i,1} + (q^{n-k} - q^{1-k})A_{i,k} \\ &+(q^{k-1} - q^{k-n})B_{i,n-k+1} \\ &+(q-1)(A_{i+k,1} - A_{i+k,n-k}) + (q^{-1} - 1)B_{i+k,k+1} \\ &+(1-q^{-1})A_{i+k,n-k+1} + (1-q)(B_{i+k,1} - B_{i+k,k})].\end{aligned} \tag{6.94}$$

This expression differs from the one proposed by Bertsimas in [24] and [27] by the first six terms on the right side inside the square brackets, and it can be further simplified to

$$\begin{aligned}\Delta E_{i,j} = \Delta E_{i,j-1} + p^2[&(q^{n-k} - q^{-k})(qA_{i,k} - A_{i,k+1}) \\ &+(q^{k-n} - q^{k-1})(qB_{i,n-k} - B_{i,n-k+1}) \\ &+(1-q^{-1})(q^k B_{i,1} - B_{j,k+1}) \\ &+(q^{-1} - 1)(q^{n-k}A_{i,1} - A_{j,n-k+1}) \\ &+(q-1)(A_{j,1} - A_{j,n-k}) \\ &+(1-q)(B_{j,1} - B_{j,k})].\end{aligned} \tag{6.95}$$

Recall that Equation (6.95) is valid for $k \geq 2$, and that in the case $k = 1$ the expression for $\Delta E_{i,j}$ as published in [24] and [27] (Equation (6.76)) is correct.

Again, the computational complexity of evaluating the 1-shift neighborhood is $O(n^2)$, the motivation being the same as for the heterogeneous PTSP case (see the end of section 6.4.2.2). In practice, the homogeneous case will be faster, since one does not need to compute the matrices Q and $\overline{Q}$.

6.4.4 Computational test

In this section we report the results of a computational test showing the improvement due to the use of the correct cost values with respect to using the incorrect expressions for the costs 2-p-opt and 1-shift moves published in [24] and in [27] for the homogeneous PTSP. In the following section 6.4.4.1 we explain the search strategies for the 2-p-opt and 1-shift local search, that we implemented following [24] and [27]. In section 6.4.4.2, we describe the experimental setup, and in section 6.4.4.3 we show and comment the experimental results.

6.4.4.1 Pseudocode of the 2-p-opt and 1-shift local search

2-p-opt The 2-p-opt local search proceeds in two phases. The first phase consists of only exploring the moves that swap two consecutive nodes $\lambda(i)$ and $\lambda(i{+}1)$ of the current tour λ, as represented in pseudocode by Algorithm 12, named Swapping_local_search. The costs $\Delta E_{i,i+1}$ are computed for every value of i (by means of Equation (6.76)),

and each time a negative $\Delta E_{i,i+1}$ is encountered, one immediately swaps the two nodes involved. Note that since the computation of $\Delta E_{i,i+1}$ only involves two rows of A and B, one can proceed to the next pair of nodes without recomputing each entire matrix. The local search has, as second argument, a tour λ_{BSF} which is the best-so-far tour known by the algorithm calling the Swapping_local_search function. At the end of the first phase of the local search, an a priori tour is reached for which every $\Delta E_{i,i+1}$ is positive, and the matrices A and B are complete and correct for that tour.

Algorithm 12 Swapping_local_search(λ, λ_{BSF})

```
for (i = 1, 2, ..., n) do
  compute rows i and i + 1 of matrices A and B relative to the current solution λ
  compute ΔE_{i,i+1} according to Equation (6.43)
  if (ΔE_{i,i+1} < 0) then
    λ := the tour obtained from λ by switching node λ(i) with node λ(i + 1)
    if E[L(λ)] < E[L(λ_BSF)] then
      λ_BSF := λ
    end if
  end if
end for
if (the starting solution has changed) then
  re-compute the full matrices A and B
end if
```

The second phase of the local search consists of computing $\Delta E_{i,j}$ with $j = i+k$ and $k \geq 2$ recursively by means of Equation (6.75). The 2-p-opt, represented in pseudocode by Algorithm 13, is implemented as a first-improvement local search, that is, when a neighbor better than the current solution is found, the current solution is immediately updated with the neighbor solution, and the search is restarted. When there are no improving solutions in the neighborhood, or when the time is over, the search stops.

1-shift The 1-shift algorithm follows the same lines as the 2-p-opt algorithm: all phase one computations, including the accumulation of matrices A and B, proceed in the same way (see Algorithm 12). The second phase of the local search consists of computing $\Delta E_{i,j}$ with $j = i + k$ and $k \geq 2$ recursively by means of Equation (6.95). Differently from the 2-p-opt, the 1-shift, represented in pseudocode by Algorithm 14, is implemented as a best-improvement local search, that is, the whole neighborhood is explored and the current solution is updated with the best (improving) neighbor solution. When there are not improving solutions in the neighborhood, or when the time is over, the search stops.

6.4.4.2 Experimental setup

We considered instances with $n = 100$ and with customers coordinates uniformly and independently distributed on the square $[0, 1]^2$. For generating random instances we

Algorithm 13 2-p-opt(λ, λ_{BSF})

(1) Swapping_local_search(λ, λ_{BSF})
$k := 2$
while ($k \leq n-2$ and time is not over) **do**
  $i := 1$
  while ($i \leq n$ and time is not over) **do**
    compute $\Delta E_{i,i+k}$ according to Equation (6.75)
    if ($\Delta E_{i,i+k} < 0$) **then**
      $\lambda :=$ tour obtained by inverting section $\lambda(i), \lambda(i+1), \ldots, \lambda(i+k)$ of λ
      if $E[L(\lambda)] < E[L(\lambda_{BSF})]$ **then**
        $\lambda_{BSF} := \lambda$
      end if
      go to (1)
    else
      $i := i + 1$
    end if
  end while
  $k := k + 1$
end while

Algorithm 14 1-shift(λ, λ_{BSF})

(1) Swapping_local_search(λ, λ_{BSF})
while (locally optimal tour not found and time is not over) **do**
  for ($i = 1, 2, \ldots, n$) **do**
    for ($k = 2, \ldots, n-2$) **do**
      compute $\Delta E_{i,i+k}$ according to Equation (6.95)
    end for
  end for
  if ($\arg\min_{i,k} \Delta E_{i,i+k} < 0$) **then**
    $\lambda :=$ tour obtained from λ by inserting $\lambda(i)$ after $\lambda(i+k)$
    if $E[L(\lambda)] < E[L(\lambda_{BSF})]$ **then**
      $\lambda_{BSF} := \lambda$
    end if
    go to (1)
  else
    return locally optimal tour λ
  end if
end while

used the Instance Generator Code of the 8^{th} DIMACS Implementation Challenge at `http://www.research.att.com/~dsj/chtsp/download.html`. Distances were computed with the Euclidean metric. We considered only homogeneous PTSP instances, and for each of the common demand probabilities $p = 0.1, 0.2, ..., 0.9$ we generated 10 problems. Experiments were run on a PowerPC G4 400MHz, the code has been written in C++ and it is available at `http://www.idsia.ch/~leo` .

6.4.4.3 Comparison between correct and incorrect local search

We tested the 2-p-opt local search algorithm with three delta objective evaluation expressions: the correct one (Equation (6.75)), the incorrect one published in [24], and the incorrect one published in [27]. We also tested two versions of the 1-shift local search algorithm: one with the correct delta objective evaluation (Equation (6.94)) and one with the incorrect delta objective evaluation published in [24] (which is the same as that published in [27]). In the following, we denote by the prefix 'C', the algorithms involving the correct delta objective evaluations (e.g., C-1shift), and by the prefix 'I' the ones implementing the incorrect delta objective evaluations as published in [27] (e.g., I-1shift). Since we obtained similar results with the incorrect expressions of [24] and [27], in the following we only show results obtained with the delta objective evaluation expression of [27].

For each C- and I-local search, we considered two types of starting solutions, generated with two different solution construction heuristics, namely the the Radial Sort and the Space Filling Curve heuristic. These solution construction heuristics have been extensively applied in previously published papers on the PTSP, and are described in Section 4.5.

The absolute solution quality of the tested algorithms was evaluated with respect to near-optimal solutions which were heuristically generated in the following way. Consider three solution construction heuristics, namely Radial Sort, Space Filling Curve and Random Search. Consider as local search heuristics the application of both C-1shift and C-2-p-opt one after the other. This results in two local search heuristics (first C-1shift and after C-2-p-opt and vice versa). By combining the three solution construction with the two local search heuristics one obtains six different heuristics. Return the best solution found by these six heuristic combinations.

Table 6.1 shows the results obtained with the correct algorithms, and the percent increase resulting from the use of Bertsimas' expressions. CPU running times in seconds are shown in parenthesis. In each line of the table the data show the average over 10 different instances. For each instance, each algorithm was run only once. Note, to evaluate the I-algorithms we checked the final solutions obtained using the full evaluation Equation (4.2). For the C-algorithms we confirm that this check was unnecessary because the algorithms exactly calculated the improvements obtained by the local search. In Figure 6.4 the relative difference between each algorithmic combination and the best near-optimal solution found is shown in a graphical form. It is clear from both Table 6.1 and Figure 6.4 that I-algorithms always give worse solution quality than to C-algorithms, as expected. From Table 6.1 we see that for small customer probability

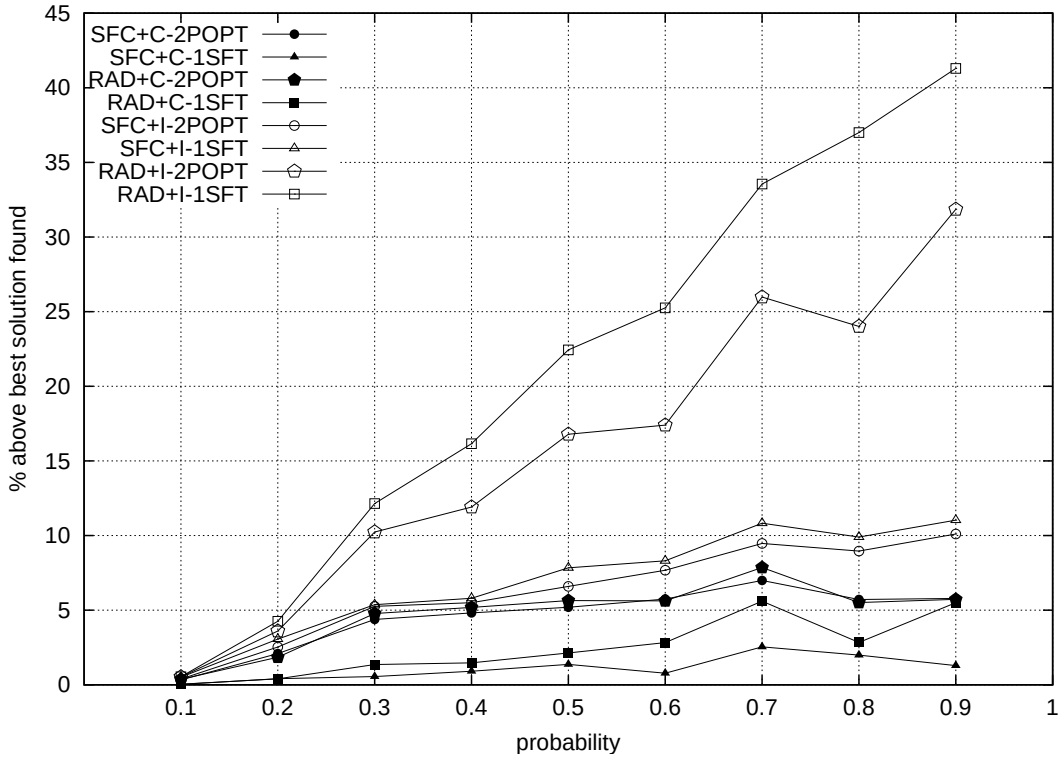


Figure 6.4: percent distance from the best solution found for C-(Correct) and I-(Incorrect) local search heuristics, combined with the Radial Sort and the Space Filling Curve solution construction heuristics. Black symbols refer to C-algorithms, and white symbols refer to I-algorithms. Each point is an average over 10 different Euclidean instances with 100 customers and customer coordinates in the unit square. The customer probability is on the horizontal axis.

(p=0.1) the error in Bertsimas' expressions results in a very small worsening effect, smaller than 1%, for all heuristic combinations. The table shows that the higher the probability, the worse are I-algorithms with respect to C-algorithms. This effect is bigger for the algorithms involving the Radial Sort, where I-algorithms obtain results more than 30% worse than C-algorithms.

The running times of all algorithmic combinations are under the second, and also obtaining the best near-optimal result only took a time of the order of the second. Running times of I-algorithms (not shown), are very similar to those of C-algorithms, except for some cases, where the solution quality of a I-algorithm is very bad. In those cases the I-local search would stop improving very early, because it starts cycling between two solutions, never stopping until the maximum allowed run time is exceeded. In order to understand why cycling may occur in the I-local search, consider for instance the I-2-p-opt local search. Given a solution λ, suppose that the section $\lambda(i), \lambda(i+1), \ldots, \lambda(i+k)$, with $k \geq 2$ is reversed, leading to a new solution $\lambda_{i,j}$, with $\lambda_{i,j}(i) = \lambda(i+k), \lambda_{i,j}(i+1) = \lambda(i+k-1), \ldots, \lambda_{i,j}(i+k) = \lambda(i)$. This means that $\Delta E_{i,i+k}(\lambda) < 0$. Now, due to the incorrectness of the expression for evaluating $\Delta E_{i,i+k}$ of [24, 27], it may happen that also $\Delta E_{i,i+k}(\lambda_{i,j}) < 0$, and that the section $\lambda_{i,j}(i), \lambda_{i,j}(i+1), \ldots, \lambda_{i,j}(i+k)$ is reversed, going back to solution λ. The same situation may occur in the I-1shift local search.

From Figure 6.4 we can see that C-1shift consistently gives the best results (among single heuristic combinations), with both the Space Filling Curve and the Radial Sort. The C-2-p-opt heuristic always gives worse results, but it is not clear if it works better with Space Filling Curve or with Radial Sort. When using I-algorithms, it seems that the starting solution is more important than the local search, since, as we see from Figure 6.4, the Space Filling Curve gives better results than the Radial Sort, no matter if it is combined with I-2-p-opt or with I-1shift. This is a side effect of the inaccuracy resulting from the use of incorrect local searches.

6.5 Approximated local search

In this section we describe three types of approximations for the move cost computation of PTSP local search operators. The first two types of approximations (section 6.5.1) are called 'ad hoc', since they are based on the particular solution structure of the PTSP, while the third type of approximation (section 6.5.2), which is based on Monte Carlo sampling, can be applied, in principle, to any SCOP. We restrict the description of move cost approximations to the 1-shift local search operator, since this is the one that appears to be more promising (according to our experiments described in Section 6.4.4). Nevertheless, our derivation might be quite easily extended to the 2-p-opt operator as well.

In the reminder of this Section, we will consider, without loss of generality, a tour $\zeta = (1, 2, ..., i, i+1, ..., j, j+1, ..., n)$ and a tour $\zeta_{i,j}$ obtained from ζ by moving node i to the position of node j and shifting backwards the nodes $(i+1, ..., j)$ (see Figure 6.2), like we did in Section 6.4.

	SFC+2POPT		SFC+1SFT		RAD+2POPT		RAD+1SFT		best
p	C	+I%	C	+I%	C	+I%	C	+I%	C
0.1	2.988 (0.20)	0.1	2.980 (0.29)	0.5	2.990 (0.14)	0.2	2.980 (0.21)	0.5	2.979 (1.28)
0.2	4.127 (0.15)	0.4	4.059 (0.33)	2.6	4.117 (0.22)	1.7	4.058 (0.44)	3.8	4.042 (1.06)
0.3	5.089 (0.11)	0.8	4.903 (0.33)	4.8	5.109 (0.33)	5.2	4.942 (0.49)	10.6	4.876 (1.11)
0.4	5.757 (0.10)	0.7	5.542 (0.28)	4.8	5.777 (0.32)	6.4	5.573 (0.44)	14.5	5.492 (1.25)
0.5	6.441 (0.12)	1.3	6.206 (0.31)	6.4	6.468 (0.39)	10.5	6.253 (0.52)	19.9	6.122 (1.34)
0.6	7.065 (0.11)	1.8	6.733 (0.31)	7.5	7.057 (0.39)	11.2	6.871 (0.50)	21.8	6.682 (1.28)
0.7	7.436 (0.12)	2.3	7.126 (0.32)	8.1	7.496 (0.46)	16.8	7.339 (0.57)	26.5	6.950 (1.51)
0.8	7.819 (0.11)	3.1	7.543 (0.27)	7.8	7.804 (0.48)	17.5	7.606 (0.55)	33.2	7.396 (1.62)
0.9	8.340 (0.12)	4.1	7.985 (0.30)	9.6	8.335 (0.49)	24.7	8.316 (0.58)	33.9	7.883 (0.47)

Table 6.1: Average solution values obtained with C-(Correct) local search algorithms, and percent increase resulting from the use of I-(Incorrect) local search algorithms. The column named 'best' shows the values of the best near-optimal solutions found heuristically, as described in section 6.4.4.3. For each probability p the average result over 10 different instances is reported (problems and experiments are the same as the ones used for obtaining Figure 6.4). CPU run times in seconds on a PowerPC G4 400MHz are shown in parentheses.

6.5.1 Ad hoc approximations for the 1-shift

Unlike the TSP, the expected cost of an a priori tour involves the arcs between all of the nodes, as it appears from Equation (4.2). The ordering of the nodes on the a priori tour simply affects the probability of an arc being used, and this probability determines the contribution this arc makes to the expected cost of the tour. The change in expected tour length, that we denote by $\Delta E_{i,j}$, resulting from shifting the node at position i to position j (see Figure 6.2) is thus based on the change in probability, or weight, placed on the arcs in the two tours ζ (the undisturbed tour) and $\zeta_{i,j}$ (the tour obtained by applying the 1-shift move to ζ). The exact value of $\Delta E_{i,j}$ thus consists of the sum of the weight change of *all* arcs.

The main idea behind the two ad hoc approximations that we are going to describe (1-shift-T and 1-shift-P), is to neglect in the computation of $\Delta E_{i,j}$ the arcs whose weight change is very small, or, equivalently to select a small number of arcs whose weight change is biggest. The two different ad hoc approximations correspond to two different ways in approximating the value of the weight change of selected arcs, while the selected arcs are the same in 1-shift-T and in 1-shift-P.

Let us first specify how we select arcs, and secondly how we approximate the weight change (respectively in Section 6.5.1.1 for 1-shift-T and in Section 6.5.1.2 for 1-shift-P). Arcs whose weight changes more from tour ζ to $\zeta_{i,j}$ are the ones satisfying the two following conditions

1. arcs whose weight change is not zero;

2. arcs that link nodes which are nearest neighbor either in ζ, or in $\zeta_{i,j}$ or in both tours.

The first condition is obvious, since an arc whose weight is the same in ζ and $\zeta_{i,j}$ cannot contribute to $\Delta E_{i,j}$. The second condition can be understood by considering that the weight of an arc corresponds to its probability of being actually traveled in the a posteriori tour. Thus, the more far away are two nodes (with respect to ζ and/or $\zeta_{i,j}$), the smaller is the probability of being actually traveled, and thus its weight in the expected cost expression. For example, arc $(i, i+4)$ in Figure 6.2 has a probability of actually being traveled in the a posteriori tour corresponding to ζ equal to $p_i p_{i+4} q_{i+1} q_{i+2} q_{i+3}$. This weight is a product over four numbers smaller than one, which is much smaller than one. Thus, the more the number of 'q' factors entering an arch weight, the smaller its weight.

There are only six arcs that satisfy both the two above mentioned conditions, and these are: $(j, j+1)$, (j, i), $(i, j+1)$, $(i-1, i+1)$, $(i-1, i)$, and $(i, i+1)$, as shown in Figure 6.5. Selecting only these six arcs from the set of all $2n(n-1)$ arcs for computing an approximation of $\Delta E_{i,j}$ constitutes a first level of approximation, which is common both to the 1-shift-T and the 1-shift-P approximation. The next level of approximation consists in approximating the difference between the weight of each of these arcs in ζ and $\zeta_{i,j}$, and the way this is done distinguishes the 1-shift-T from the 1-shift-P approximations. Before focusing on approximating weight differences, it is

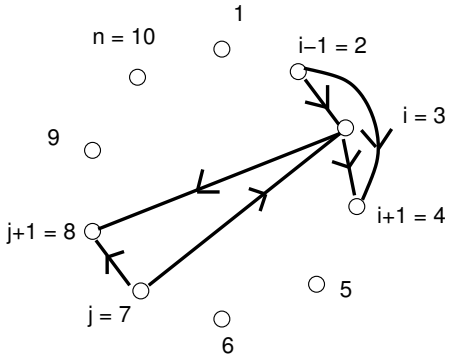


Figure 6.5: The set of arcs that are selected to approximate a 1-shift move cost $\Delta E_{i,j}$.

Arc	(1) Weight in $E[L(\zeta)]$	(2) Weight in $E[L(\zeta_{i,j})]$	(2)–(1) Weight in $\Delta E_{i,j}$
$(j, j+1)$	$p_j p_{j+1}$	$p_j p_{j+1} q_i$	$-p_j p_{j+1}(1-q_i)$
(j, i)	$p_j p_i \prod_{j+1}^{i-1} q_\zeta$	$p_j p_i$	$p_j p_i(1-\prod_{j+1}^{i-1} q_\zeta)$
$(i, j+1)$	$p_i p_{j+1} \prod_{i+1}^{j} q_\zeta$	$p_i p_{j+1}$	$p_i p_{j+1}(1-\prod_{i+1}^{j} q_\zeta)$
$(i-1, i+1)$	$p_{i-1} p_{i+1} q_i$	$p_{i-1} p_{i+1}$	$p_{i-1} p_{i+1}(1-q_i)$
$(i-1, i)$	$p_{i-1} p_i$	$p_{i-1} p_i \prod_{i+1}^{j} q_\zeta$	$-p_{i-1} p_i(1-\prod_{i+1}^{j} q_\zeta)$
$(i, i+1)$	$p_i p_{i+1}$	$p_i p_{i+1} \prod_{j+1}^{i-1} q_\zeta$	$-p_i p_{i+1}(1-\prod_{j+1}^{i-1} q_\zeta)$

Table 6.2: Exact probabilistic weights of the arcs selected for the 1-shift-T and the 1-shift-P ad hoc approximations. Tour ζ and $\zeta_{i,j}$ are shown in Figure 6.2, while for the definition of the product notation, see Equation (4.3).

useful to consider the exact weights of the selected arcs in ζ and $\zeta_{i,j}$. These are shown in Table 6.2. Note that some of the exact weight differences can require an $O(n)$ computation time, when the product of several 'q' factors must be performed.

6.5.1.1 1-shift-T approximation

In this very rough approximation, the weight difference of each selected arc (the right column of Table 6.2) is approximated either by 1 or by -1, depending on the fact whether its probability of being traveled a posteriori increases or decreases from ζ to $\zeta_{i,j}$. From an inspection of the exact weights of the six arcs that we selected, it is not difficult to verify that the weight increases for $(i-1, i+1)$, (j, i), and $(i, j+1)$, while it decreases for $(j, j+1)$, $(i-1, i)$, and $(i, i+1)$. Thus, we define the approximated move cost as

$$\Delta_T E_{i,j} = d(i-1,i+1)+d(j,i)+d(i,j+1)-d(j,j+1)-d(i-1,i)-d(i,i+1). \quad (6.96)$$

Note that $\Delta_T E_{i,j}$ also corresponds to the *length* difference between ζ and $\zeta_{i,j}$, as if, instead of dealing with a PTSP, we would be working with a TSP. This is the reason for the subscript T in $\Delta_T E_{i,j}$ and for the name of this ad hoc approximation. Obviously,

$\Delta_T E_{i,j}$ requires a constant-time computation.

6.5.1.2 1-shift-P approximation

Here, we approximate the weight difference of each selected arc (the right column of Table 6.2) by posing any product involving more than 3 'q' factors equal to zero. Let

$$ {}^3\prod_{i}^{j} q_\zeta = \begin{cases} \prod_i^j q_\zeta & \text{if } \; j-i \leq 3 \text{ or } \; j+n-i \leq 3, \\ 0 & \text{otherwise.} \end{cases} \tag{6.97} $$

Then, the approximated move cost in 1-shift-P is the following

$$ \begin{aligned} &\Delta_P E_{i,j} = \\ &d(i-1,i+1)p_{i-1}p_ip_{i+1} + d(j,i)p_jp_i(1 - {}^3\prod_{j+1}^{i-1} q_\zeta) + d(i,j+1)p_ip_{j+1}(1 - {}^3\prod_{i+1}^{j} q_\zeta) \\ &-d(j,j+1)p_jp_{j+1}p_i - d(i-1,i)p_{i-1}p_i(1 - {}^3\prod_{i+1}^{j} q_\zeta) - d(i,i+1)p_ip_{i+1}(1 - {}^3\prod_{j+1}^{i-1} q_\zeta). \end{aligned} \tag{6.98} $$

Note that, since each product appearing in Equation (6.98) can be computed in constant time, also the 1-shift-P approximation $\Delta_P E_{i,j}$ can be computed in constant time, for any node i and j.

6.5.2 Sampling approximation for the 1-shift: 1-shift-S

This approximation of $\Delta E_{i,j}$ is based on the Sampling approximation of the PTSP objective function, as defined in Section 5.2.4.1. More precisely, given a set of N samples of customers $\omega_1, \ldots, \omega_k, \ldots, \omega_N$, sampled independently according to probability $p(\omega_k)$ (see Equation (5.15)), the Sampling approximation for the 1-shift is defined as follows

$$ \Delta_{S_N} E_{i,j} = E_{S_N}[L(\zeta_{i,j})] - E_{S_N}[L(\zeta)], \tag{6.99} $$

where

$$ E_{S_N}[L(\zeta_{i,j})] = \frac{1}{N}\sum_{k=1}^{N} L(\zeta_{i,j|_{\omega_k}}) \;\text{ and }\; E_{S_N}[L(\zeta)] = \frac{1}{N}\sum_{k=1}^{N} L(\zeta_{|_{\omega_k}}). \tag{6.100} $$

Equation (6.99) can be rewritten as a sample average over length differences, each one corresponding to a sample ω_k of customers. For this purpose, let us introduce the following definitions

Definition 15 *Given a subset of customers $\omega \subseteq V$, and a node $i \in V$,*

$$ pred_\omega(i) = \begin{cases} i, & \text{if } i \in \omega, \\ \text{the first-met node } r \in \omega \text{ going backward from } i \text{ along } \zeta, & \text{otherwise.} \end{cases} \tag{6.101} $$

Definition 16 *Given a subset of customers $\omega \subseteq V$, and a node $i \in V$,*

$$succ_\omega(i) = \begin{cases} i, & \text{if } i \in \omega, \\ \text{the first-met node } r \in \omega \text{ going onward from } i \text{ along } \zeta, & \text{otherwise.} \end{cases} \tag{6.102}$$

Equipped with the above definitions, the length difference of the two a posteriori tours induced by ζ and $\zeta_{i,j}$ on a given subset of customers ω_k is the following

$$\Delta_{\omega_k} L_{i,j} = \begin{cases} 0, & \text{if } i \notin \omega_k, \\ \begin{array}{l} d(pred_{\omega_k}(i-1), succ_{\omega_k}(i+1)) \\ +d(pred_{\omega_k}(j), i) + d(i, succ_{\omega_k}(j+1)) \\ -d(pred_{\omega_k}(j), succ_{\omega_k}(j+1)) \\ -d(pred_{\omega_k}(i-1), i) - d(i, succ_{\omega_k}(i+1)), \end{array} & \text{otherwise,} \end{cases} \tag{6.103}$$

and the sampling approximation of 1-shift can be computed with the following expression

$$\Delta_{S_N} E_{i,j} = \frac{1}{N} \sum_{k=1}^{N} \Delta_{\omega_k} L_{i,j}. \tag{6.104}$$

Since the time required to compute a predecessor or a successor node is $O(n)$ in the worst case, the time complexity for computing $\Delta_{S_N} E_{i,j}$ is $O(Nn)$, which is much higher than the constant time complexity of 1-shift, 1-shift-T and 1-shift-P. Note, however, that there is the possibility to decrease at least by a constant factor the complexity of $\Delta_{S_N} E_{i,j}$, if the neighborhood is explored lexicographically, and if the same set of samples is kept fixed during the neighborhood exploration. Let us see in more detail how this is done.

Suppose that the 1-shift neighborhood is explored lexicographically, such that for each $i = 1, 2, \ldots, n$ and for each $j = 1, 2, \ldots, n$, the value of $\Delta_{S_N} E_{i,j}$ is computed in this order by means of Equation (6.104). Suppose also that before starting the double cycle over indexes i and j, N random samples of the subsets of customers are generated independently, and this set is kept fixed until all $\Delta_{S_N} E_{i,j}$ are evaluated. It is not difficult to see that in such a situation it is possible to compute some of the successor and predecessor nodes needed for the computation of $\Delta_{\omega_k} L_{i,j}$ (see Equation (6.103)) recursively. In fact, for any node $i \in V$ and any subset of customers ω, one can compute $succ_\omega(i)$ and $pred_\omega(i)$ in the following ways

$$succ_\omega(i) = \begin{cases} succ_\omega(i) & \text{(by Definition 16 in } O(n)\text{)}, & \text{if } i-1 \in \omega \\ succ_\omega(i-1) & \text{(recursively in } O(1)\text{)}, & \text{otherwise.} \end{cases} \tag{6.105}$$

$$pred_\omega(i) = \begin{cases} pred_\omega(i-1) & \text{(recursively in } O(1)\text{)}, & \text{if } i \notin \omega \\ i & \text{(in } O(1)\text{)}, & \text{otherwise.} \end{cases} \tag{6.106}$$

Thus, the probability of needing an $O(n)$ computation for $\Delta_{\omega_k} L_{i,j}$ is equal to the probability that customer $i-1 \in \omega$ in Equation (6.105), and this can be estimated by p, the average customers probability. In this way, the complexity of computing $\Delta_{S_N} E_{i,j}$ by Equation (6.104) is now $O(Npn)$ (and not $O(Nn)$ as it would be without any hypothesis on the neighborhood exploration). For PTSP instances with low customers probabilities, the time complexity of 1-shift-S may be comparable to that of 1-shift, 1-shift-T and 1-shift-P.

The interest in considering 1-shift-S despite its possibly high time complexity lies in its generality. In fact, the sampling approximation can be applied to any SCOP, even when there is no analytical or closed-form expression for the objective function, and so it is not possible to design any good ad hoc approximation.

6.5.3 Pseudocode of the approximated 1-shift local search

The three approximated move cost expressions $\Delta_T E_{i,j}$, $\Delta_P E_{i,j}$, and $\Delta_{S_N} E_{i,j}$ in principle can be used inside a local search algorithm without any restriction on the order in which the 1-shift neighbors are explored, and both in first-improvement or best-improvement mode. However, since our goal is to compare the effectiveness of these approximated move costs with the exact ones, we have developed 1-shift-T, 1-shift-P, and 1-shift-S by keeping the same exploration strategy (best-improvement with lexicographic neighborhood exploration) as the 1-shift algorithm based on the exact recursive move costs (described in Section 6.4.4.1). The pseudocode of the three approximated 1-shift local searches is very similar to the one of the exact recursive 1-shift, except for two differences: first, the exact move cost $\Delta E_{i,j}$ is substituted by, respectively, $\Delta_T E_{i,j}$, $\Delta_P E_{i,j}$, and $\Delta_{S_N} E_{i,j}$; second, there is no distinction between a first phase (where only single swap moves were checked) and a second phase. The pseudocode of 1-shift-T, 1-shift-P, and 1-shift-S are represented respectively in Algorithms 15, 16, and 17, and Table 6.5.3 summarizes the asymptotic time complexity for all the 1-shift variants that we have presented.

Local Search	Single move	All neighborhood
1-shift	$O(n)$, if $j = i + 1$; $O(1)$, otherwise	$O(n^2)$
1-shift-T	$O(1)$	$O(n^2)$
1-shift-P	$O(1)$	$O(n^2)$
1-shift-S (N samples)	$O(Npn)$	$O(Npn^3)$

Table 6.3: Time complexity of the neighborhood exploration for exact and approximated versions of 1-shift.

6.6 Overview of the results

There are several achievements obtained in this chapter, both from the methodological point of view common to all SCOPs, and from the point of view of specific results valid

Algorithm 15 1-shift-T(λ, λ_{BSF})

while (locally optimal tour not found and time is not over) **do**
  for ($i = 1, 2, \ldots, n$) **do**
    for ($k = 1, \ldots, n-2$) **do**
      compute $\Delta_T E_{i,i+k}$ using equation (6.96)
    end for
  end for
  if ($\min_{i,k} \Delta_T E_{i,i+k} < 0$) **then**
    $(i, k) := \arg\min_{i,k} \Delta_T E_{i,i+k}$
    $\lambda :=$ tour obtained from λ by inserting $\lambda(i)$ after $\lambda(i+k)$
    if $E[L(\lambda)] < E[L(\lambda_{BSF})]$ **then**
      $\lambda_{BSF} := \lambda$
    end if
  else
    return locally optimal tour λ
  end if
end while

Algorithm 16 1-shift-P(λ, λ_{BSF})

while (locally optimal tour not found and time is not over) **do**
  for ($i = 1, 2, \ldots, n$) **do**
    for ($k = 1, \ldots, n-2$) **do**
      compute $\Delta_P E_{i,i+k}$ using equation (6.98)
    end for
  end for
  if ($\min_{i,k} \Delta_P E_{i,i+k} < 0$) **then**
    $(i, k) := \arg\min_{i,k} \Delta_P E_{i,i+k}$
    $\lambda :=$ tour obtained from λ by inserting $\lambda(i)$ after $\lambda(i+k)$
    if $E[L(\lambda)] < E[L(\lambda_{BSF})]$ **then**
      $\lambda_{BSF} := \lambda$
    end if
  else
    return locally optimal tour λ
  end if
end while

Algorithm 17 1-shift-S(λ, λ_{BSF}) with fixed number of samples N

```
while (locally optimal tour not found and time is not over) do
  GenerateSamples(N)
  for (i = 1, 2, ..., n) do
    for (k = 1, ..., n − 2) do
      compute Δ_{S_N} E_{i,i+k} using equation (6.104)
    end for
  end for
  if (min_{i,k} Δ_{S_N} E_{i,i+k} < 0) then
    (i, k) := arg min_{i,k} Δ_{S_N} E_{i,i+k}
    λ := tour obtained from λ by inserting λ(i) after λ(i + k)
    GenerateSamples(N)
    Compute E_{S_N}[L(λ)] and (re-)compute E_{S_N}[L(λ_BSF)] using the last generated
    samples
    if E_{S_N}[L(λ)] < E_{S_N}[L(λ_BSF)] then
      λ_BSF := λ
    end if
  else
    return locally optimal tour λ
  end if
end while
```

and useful only for the PTSP.

- From the methodological point of view, we have presented, discussed, and exemplified in the case of the PTSP the three situations that one can face when designing a local search for a SCOP. Namely: using the full exact evaluation of the objective function for computing the move cost; finding exact and recursive expressions for the move cost, which are faster to be computed; or considering approximations of the move cost. In this last case, similarly to what happens for the evaluation of one single solution, ad hoc and sampling approximations may be considered.

- After verifying the burden of one single run of local search when the full exact evaluation of the objective function is used for computing the move cost, we can say that, unless faster alternatives are found, it is not convenient to consider such local search for integration in a metaheuristics. In fact, the metaheuristic itself would be very much slowed, and it is very unlikely that the already quite good results obtained without the local search would be improved in this situation.

- For the specific case of the PTSP, it has been possible to derive exact recursive and efficient move cost expressions for the 2-p-opt and 1-shift local search operators. This is a very good result, given that the recursive expressions allow the exploration of the neighborhood in the same asymptotic time complexity as the

(deterministic) TSP. In general, when efficient expressions for the move cost are obtained, it is very likely that, when combined with a metaheuristic, they will let greatly improve its performance. We will verify this aspect in the next chapter, dealing with hybrid ACO, that is, ACO integrated with local search.

- We proposed two ad hoc and one Sampling approximation of the move cost for the 1-shift local search operator. The main idea behind the two ad hoc approximations can be in principle exploited also in other SCOPs, since it is based on estimating and neglecting the terms in the objective function that are smaller, due to small probabilistic weights. The Sampling approximation that we propose is enhanced by some observations that allow to partially compute the move cost in a recursive way, thus saving a lot of computation time. However, the computational burden of the Sampling-based local search is bigger than ad hoc approximations.

Chapter 7

Integration of Ant Colony Optimization with local search

In this chapter, we focus on hybrid algorithms, that is, algorithms that solve an optimization problem by combining solution construction and local search techniques[1]. In particular, we consider hybrid algorithms obtained by integrating local search algorithms of Chapter 6 with the constructive ACO algorithms of Chapter 5 and with the simple heuristics presented in Chapter 4.

In the previous chapters, we have proposed several solution construction algorithms (5 simple constructive heuristics and a few ACO variants) and several variants of local search for the PTSP (2-p-opt and 1-shift with 4 different ways for computing the move cost). In principle, any combination of these algorithms can be used to obtain a new hybrid one that is hopefully better performing than the simple solution construction algorithm. However, it is impractical to test all the hybrid combinations (potentially more than 30) and we have to make a selection guided by some specific research question. The following are the main issues that we want to address by testing experimentally some hybrid combinations of the previously proposed algorithms:

Issue 1: the advantage of using a local search with ACO. We have seen that the design of a feasible local search algorithm for the PTSP has been more difficult than the design of an ACO algorithm. This situation can be quite often encountered in SCOPs, where metaheuristics are usually easy to design, but the same cannot be said for local search algorithms. Thus, we want to verify whether the effort for the design of a good local search is balanced by a significant improvement in the solution quality, when the local search is used to hybridize an easy-to-design metaheuristic (ACO in this context). Therefore, we want to quantify the improvement due to the use of local search with ACO, given a fixed

[1]Note, however, that the term 'hybrid' can also refer to other features of an algorithm. For instance, in the work about the vehicle routing problem with stochastic demands (VRPSD) reported in Appendix B, the term hybrid also refers to the fact of using inside the algorithm the objective function of other problems. Thus, in that context, the use of ad hoc objective function approximations taken from other problems is a type of hybridization.

amount of time is available for solving each PTSP problem.

Issue 2: exact versus approximated local search. In designing a local search for the PTSP (and for any SCOP in general), we have seen several possibilities for computing the move cost (exact recursive computation, ad hoc approximations, and the Sampling approximation). Each one of these alternatives needs more or less stringent conditions on the structure of the SCOP, such as a particular neighborhood structure, an explicitly known objective function, the possibility of performing simulations. Therefore, when one faces the problem of designing a local search for a particular SCOP, it may be useful to have an idea about which is the most promising choice among the different alternatives. For this purpose, we want to achieve a classification in terms of solution quality of the different local search variants, because this can constitute a sort of guideline for the practitioner who faces a SCOP different from the PTSP.

The remainder of this chapter is organized as follows. Section 7.1 presents the hybrid algorithms selected for the investigation of the above issues. Experiments are described in Section 7.2, and Section 7.3 summarizes the main results obtained.

7.1 Description of hybrid algorithms

In order to address the issues listed above, we have considered for hybridization only the 1-shift local search, since it has been shown to be more promising than the 2-p-opt (see the experimental results described in Section 6.4.4.3), and also because for 1-shift we have developed the different move cost sampling schemes. Our main interest is in analyzing the achievements of hybrid ACO, to which we have applied both exact and approximated versions of the 1-shift local search, as we are going to describe in Section 7.1.2. Nevertheless, we have also considered hybrid versions of the simple constructive heuristics using the exact recursive 1-shift local search, as described in Section 7.1.1. Similarly to the case of non-hybrid ACO, the performance of simple heuristics is considered a sort of minimum quality level, that any metaheuristic should be able to surpass.

7.1.1 Hybridization of simple constructive heuristics

We have considered hybridized versions of all the simple constructive heuristics introduced in Section 4.5, namely Space Filling Curve, Radial Sort, Farthest Insertion, Nearest Neighbor, and the Random Search heuristic. For all these heuristics, except Random Search, hybridization is done simply by applying just once the exact recursive 1-shift local search (Algorithm 14) to the single solution produced by the constructive heuristic. In Random Search hybridization is done differently. There, the exact recursive 1-shift algorithm is applied after any random solution, until the fixed available runtime is exhausted. Thus, in Random Search the local search is applied several times. At the end, the best solution with respect to the PTSP objective function is returned.

7.1.2 Hybridization of ACO

In order to address Issue 1 previously described, we considered an hybridized version of pACS with the exact recursive 1-shift local search (Algorithm 14). For addressing Issue 2, we considered different hybridizations, namely pACS with 1-shift-P, pACS with 1-shift-T, and pACS-S with 1-shift-S (for a description of the move cost approximation in 1-shift-P, -T, and -S variants, see Section 6.5).

The hybridization of pACS is done by applying the local search to each ant solution, before the global pheromone update is performed, as shown in Algorithm 18.

Algorithm 18 hybrid pACS (pACS + 1-shift, pACS + 1-shift-T, pACS + 1-shift-P)

```
1: Initialization [like in pACS]
2: for iteration k = 1, 2, ... do
3:    Initialize best ant solution λ_BA
4:    for ant a = 1, 2, ..., m do
5:       ConstructAntSolution [each ant constructs its solution λ_a]
6:       Apply either 1-shift(λ_a, λ_BSF), 1-shift-T(λ_a, λ_BSF), or 1-shift-P(λ_a, λ_BSF)
7:       if E[L(λ_a)] < E[L(λ_BA)] then
8:          set λ_BA = λ_a
9:       end if
10:   end for
11:   if E[L(λ_BA)] < E[L(λ_BSF)] then
12:      set λ_BSF = λ_BA
13:   end if
14:   GlobalPheromoneUpdate
15: end for
```

The hybridization of pACS-S has been done differently. In fact, in order to hybridize pACS-S with 1-shift-S, we had to take into account the fact that the move cost of 1-shift-S is much more computationally expensive than in 1-shift, 1-shift-T and 1-shift-P (see Table 6.5.3). Preliminary experiments lead us to consider an hybridization where local search is only applied to the best ant solution of each iteration, as shown in Algorithm 19.

7.2 Experimental analysis

The experimental environment is the same as for the non hybrid ACO versions described in Section 5.1.2.1. The parameters used for hybrid pACS and hybrid pACS-S are the same as those used, respectively, in pACS and pACS-S, as described in Section 5.1.2.2 and Section 5.2.5.1.

Algorithm 19 hybrid pACS-S (pACS-S + 1-shift-S)

Initialization [like in pACS]
for iteration $k = 1, 2, \ldots$ **do**
  Initialize best ant solution λ_{BA}
  Set N = NumberOfSamples(k) [apply Equation (5.20)]
  for ant $a = 1, 2, \ldots, m$ **do**
    ConstructAntSolution [each ant constructs its solution λ_a]
    GenerateSamples(N)
    Compute $E_{S_N}[L(\lambda_a)]$ and re-compute $E_{S_N}[L(\lambda_{BA})]$ using the last generated samples
    if $E_{S_N}[L(\lambda_a)] < E_{S_N}[L(\lambda_{BA})]$ **then**
      set $\lambda_{BA} = \lambda_a$
    end if
  end for
  GenerateSamples(N)
  Re-compute $E_{S_N}[L(\lambda_{BA})]$ and $E_{S_N}[L(\lambda_{BSF})]$ using the last generated samples
  if $E_{S_N}[L(\lambda_{BA})] < E_{S_N}[L(\lambda_{BSF})]$ **then**
    set $\lambda_{BSF} = \lambda_{BA}$
  end if
  1-shift-S($\lambda_{BA}, \lambda_{BSF}$)
  GlobalPheromoneUpdate [using $\Delta\tau_{ij} = (E_{S_N}[L(\lambda_{BSF})])^{-1}$]
end for
Compute $E[L(\lambda_{BSF})]$

7.2.1 Advantage of using a local search with ACO

If we consider average results over the entire PTSP benchmark (Table 7.1), it appears that pACS+1-shift goes nearer to the lower bound than pACS, but the amount of improvement is just 0.1% (see first row of the Table). The advantage of pACS+1-shift with respect to pACS is more relevant if we look at the average time for reaching the best solution within the allowed computation time: pACS+1-shift requires on average about 60% time less than pACS (again, see first row of Table 7.1). Table 7.1 also shows that the simple heuristics, even when hybridized with the 1-shift local search, do not reach the performance neither of pACS+1-shift, nor of pACS. The statistical significance of the difference among algorithms has been tested by means of a Friedman two-way analysis of variance by ranks [64]. The results of this type of test are reported in Figure A.1 of Appendix A.

As we have learned from the experimental analysis of pACS in Section 5.1.2 however, the algorithmic performance may be very different for different average customers probability. In particular, for pACS we arrived at the important conclusion that only PTSP instances with average customers probability up to 0.5 are worth solving by pACS, since for bigger customers probability the problem can be better treated like a TSP. Is the situation the same for pACS+1-shift? A look at the left part of Figure

	with 1-shift		without 1-shift	
algorithm	% above LB	time (seconds)	% above LB	time (seconds)
pACS	81.1%	1476.9	81.2%	2453.7
FI	83.2%	86.9	90.2%	0.1
SFC	93.1%	257.9	111.6%	0.1
NN	98.7%	293.1	111.1%	35.5
RAD	231.2%	2074.7	388.8%	0.1
RS	667.4%	131.7	1228.6%	2712.0

Table 7.1: Aggregated results showing average values over all instances of the PTSP benchmark.

7.1 tells us that the answer is positive. It is thus more fair to restrict the comparison between pACS+1-shift and other algorithms only to PTSP instances with average customers probability smaller than or equal to 0.5. Comparative results restricted to this subset of PTSP instances are reported in Table 7.2. From the Table and from a Friedman two-way analysis of variance by ranks [64], it is clear that pACS+1-shift finds statistically significant better results than all other algorithms (at a confidence level of 95%). In particular, the improvement with respect to pACS is 0.4%. Detailed results are reported by Table A.3 in Appendix A.

We have also analyzed how the influence of using 1-shift in pACS depends on the other factors characterizing PTSP instances (variance of the customers probability and number of customers). The number-of-customers factor, which is shown in the right plot of Figure 7.1 does not seem to be significant. The customers-probability-variance factor is more interesting, and we have showed it in the central plot of Figure 7.1. From the plot one can conclude that hybridization is more effective for instances with low customers probability variance, while for instances with high probability variance, it seems that pACS produces better results.

7.2.2 Exact versus approximated local search

The effect of hybridizing ACO with different approximated local search operators is summarized in Table 7.3. The ranking of the four algorithms has been tested by the Friedman two-way analysis of variance by ranks [64]. The statistical test tells us that differences are significant at a confidence level of 95%, except for pACS+1-shift-P and pACS+1-shift-T.

As expected, pACS+1-shift, which is based on the exact recursive local search, is the best one. This is an important confirmation, since it means that the big effort done in Section 6.4 for finding fast recursive expressions for the move cost has been worth.

Quite interestingly, the second classified hybrid ACO of Table 7.3 is the sampling-based algorithm pACS-S+1-shift-S. This fact is 'good news', as the sampling approximation can be applied to *any* SCOP, even when there is no analytical or closed-form expression for the objective function, and so it is not possible to design any good ad hoc approximation, or any fast recursive exact move cost expression. The performance

X	$\frac{\text{best(pACS+1-shift)}-\text{best(X)}}{\text{best(X)}}$ $(0.1 \leq p \leq 0.5)$
pACS	-0.4%
FI	-5.9%
NN	-16.1%
RAD	-46.0%
RS	-73.8%
SFC	-12.8%
FI+1-shift	-1.2%
NN+1-shift	-10.2%
RAD+1-shift	-14.9%
RS+1-shift	-25.0%
SFC+1-shift	-3.7%

Table 7.2: Average relative difference of the best solution found by pACS+1-shift with respect to other algorithms (listed in column named 'X'). Averages are restricted to significant PTSP instances, that is, those with customers probability less then or equal to 0.5.

of pACS-S+1-shift-S is not much worse than pACS+1-shift and pACS (taking into account Table 7.1), and it is superior to all the simple heuristics even when hybridized with the exact 1-shift (again, see Table 7.1), which means that the sampling approximation is indeed quite effective.

Table 7.3 shows another interesting result: the bad performance of the two ad hoc approximations, 1-shift-P and 1-shift-T. Their performance is bad also in comparison to the simple pACS and to the FI heuristic hybridized with the exact 1-shift (see Table 7.1). This means that the quality of the move cost approximations $\Delta_P E_{i,j}$ and $\Delta_T E_{i,j}$ is not sufficient, and the application of these approximated local searches to pACS is only a time consuming task. Computation time would be much better exploited by the solution construction mechanism of pACS.

The performance of the different hybrid ACO versions with respect to the value of the optimal TSP solution is shown in Figure 7.2. Similarly to what emerged from Figure 7.1, there are some groups of PTSP instances for which pACS-S+1-shift-S performs better than pACS+1-shift. This happens for probabilities higher than 0.4, and for probability variance higher than 60%. Detailed results obtained by pACS-S+1-shift-S, pACS+1-shift-T, and pACS+1-shift-P are reported, respectively, by Table A.4, A.5, and A.6 in Appendix A. Appendix A also reports, in Figure A.1, the results of the Friedman two-way analysis of variance by ranks done on all the hybrid and non-hybrid versions of pACS, together with the simple heuristics and the optimal TSP solutions.

7.3 Overview of the results

- Simple heuristics enhanced by local search do not reach the performance of the simple, non-hybrid pACS. This is an additional confirmation that, even without

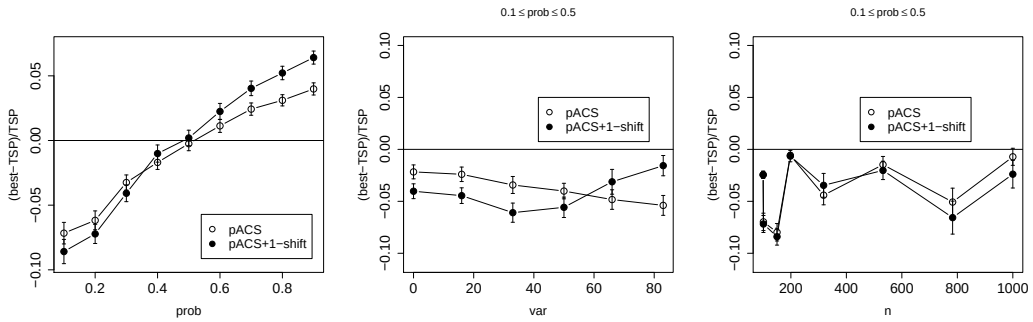


Figure 7.1: Relative gain of pACS (with and without the 1-shift local search) over the optimal TSP solution, versus, respectively, average customers probability (left), variance of the customers probability (center), and number of customers. Points are averages computed over the the complete PTSP benchmark (left) and over subset of PTSP instances with customers probability smaller than or equal to 0.5 (center and right). The length of error bars is equal to the standard deviation of the relative gain.

	% above LB	time(seconds)
pACS+1-shift	81.1%	1476.9
pACS-S+1-shift-S	81.9%	1965.2
pACS+1-shift-P	87.3%	1995.1
pACS+1-shift-T	88.6%	1859.2

Table 7.3: Average results over the complete PTSP benchmark of different ACO versions, hybridized with exact and approximated 1-shift local search algorithms.

local search, the ACO metaheuristic obtains quite good results for the PTSP.

- The hybridization of pACS with the exact recursive 1-shift local search leads to an improvement of less than 1%. This is not impressive, but it should be considered that the hybrid version converges faster to good solutions. In fact, the time required for finding the best solution in the hybrid pACS is about 60% less than in the non-hybrid version.

- Interestingly, the exact 1-shift local search does not improve, but worsens the solution quality of pACS, for PTSP instances characterized by very high variance of the customers probability.

- The sampling-based ACO, pACS-S and its hybrid version, is quite near to pACS hybridized with the exact recursive local search (within 1% on average). This is an interesting and good result, when we think that the Sampling approximation is very general, and can be applied to any SCOP, particularly to those for which an explicit expression for the objective function is not available. The result is also not obvious, since the behavior of a local search with approximated move cost,

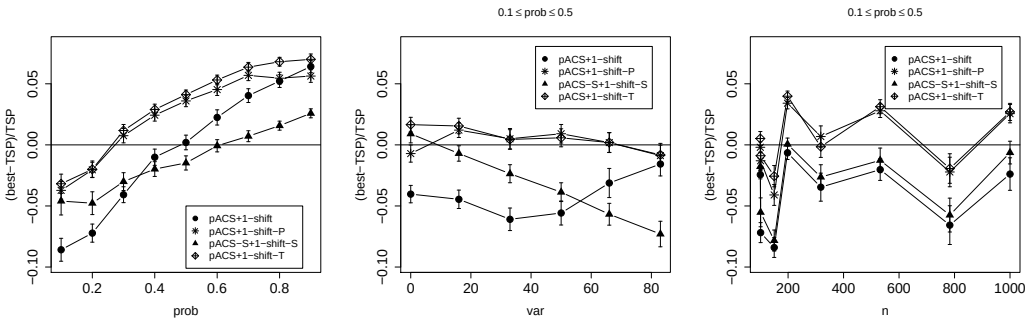


Figure 7.2: Relative gain of hybrid ACO algorithms over the optimal TSP solution, versus, respectively, average customers probability (left), variance of the customers probability (center), and number of customers (right). Points are averages computed over the the complete PTSP benchmark (left) and over subset of PTSP instances with customers probability smaller than or equal to 0.5 (center and right).

like 1-shift-S, could be in principle also quite bad.

- The best performance of the sampling-based ACO is in correspondence of PTSP instances with high variance of the customers probability, where pACS-S+1-shift-S outperforms both pACS and pACS+1-shift. This means that, for this class of PTSP instances, the Sampling approximation could be chosen even for those SCOPs, such as the PTSP, for which the exact objective function is available.

- When ACO is hybridized with local search based on ad hoc approximations, its performance can be bad, both with respect to non-hybrid ACO, and to simple heuristic hybridized with a good local search. Such bad performance of local search based on ad hoc approximations reveals that ad hoc approximations should be chosen very carefully, and that, whenever possible, it might be better to spend some effort in searching for exact efficient recursive move cost expressions. When this is impossible, it may still be better to put more effort in improving the performance of the non-hybrid metaheuristic, instead of considering the integration of local search with ACO.

Chapter 8

Conclusions

In this thesis we have focused on SCOPs (Stochastic Combinatorial Optimization Problems), a wide class of combinatorial optimization problems under uncertainty, where part of the information about the problem data is unknown at the planning stage, and some knowledge about its probability distribution is assumed. Note, however, that the SCOP modeling approach is one among many other possible modeling approaches to optimization under uncertainty, such as Pure Online, Robust, and Fuzzy approaches, that have been briefly described in the introduction, but that we have otherwise neglected.

The first part of the thesis has been devoted to the introduction and survey of the applications to SCOPs of metaheuristics for which there is a significant amount of literature. These include, Ant Colony Optimization, Evolutionary Computation, Simulated Annealing, Tabu Search and Stochastic Partitioning Methods. From the survey, two properties of metaheuristics emerge clearly: they are a valid alternative to exact classical methods for addressing real-sized SCOPs, and they are flexible, since they can be quite easily adapted to solve different SCOPs formulations, both static and dynamic. In fact, there exist applications of metaheuristics to problems formalized as Stochastic, Chance Constraint, Two-stage and Multi-stage Integer Programs, and Markov Decision Processes.

On the base of the reviewed literature, we have identified mainly three key issues in solving SCOPs via metaheuristics: (1) the design and integration of ad hoc, fast and effective objective function approximations inside the optimization algorithm; (2) the estimation of the objective function by sampling when no closed-form expression for the objective function is available, and the ways to deal with the time complexity and noise induced by this type of estimation; (3) the characterization of the efficiency of metaheuristic variants with respect to different levels of stochasticity in the problem instances. In particular, the following observations can be done by looking at the application of metaheuristics to SCOPs.

(1) The design of ad hoc approximations is strongly problem dependent, and no general rule exists for finding efficient approximations of the objective function. Examples of ad hoc approximations in the literature include: the use of the objective function

of a deterministic combinatorial optimization problem similar in some respects to the SCOP considered; the use of truncated expressions for the expected values, by neglecting terms that are estimated to be small; the use of scenarios, instead of considering the true probabilistic model. Ad hoc approximations, if on one side accelerate the evaluation and comparison among solutions, on the other side introduce a systematic error in the computation of objective values. Usually, the systematic error cannot be reduced unless a different, more precise ad hoc approximation is designed, and it can only be evaluated by comparison with the exact objective value. Thus, metaheuristics typically alternate exact and approximated evaluations during the optimization process.

(2) When the estimation of the objective function by sampling is used, the decision whether a solution is better than another can only be done by statistical sampling, obtaining a correct comparison result only with a certain probability. The way sampling approximation is used in metaheuristics largely depends on the way solutions are compared and the best solution among a set of other solutions is selected ('selection-of-the-best' method). The selection-of-the-best method that a metaheuristic uses for performing sample averages and for comparing solutions can have a great impact on the effectiveness of the algorithm, but it is still hard to say which method is the most effective in relation to the metaheuristic where it is employed.

(3) It has been recognized that completely different algorithms may be needed for small and for large search spaces. Also the "degree of randomness", that is, the size of noise compared to the undisturbed objective function values, is an important factor. It cannot be expected that a metaheuristic variant working well for solution spaces with a small amount of noise will also perform optimally for solution spaces with a large amount of noise, and vice versa. It appears that a characterization of metaheuristic variants for SCOPs with respect to their appropriate domains of problem instance types still waits for being elaborated.

The second part of the thesis describes a case study where we have investigated the three key issues introduced above, by focusing in particular on a SCOP belonging to the class of vehicle routing problems: the PTSP (Probabilistic Traveling Salesman Problem). This problem is NP-hard, and can also be formalized as a Stochastic Integer Program. Since no commonly used and satisfying benchmark existed for the PTSP, we have generated our own. It consists of 432 PTSP instances carefully designed, in particular to allow the analysis of the behavior of optimization algorithms for different levels of stochasticity. In order to facilitate future comparisons with our results, we have also used a known lower bound from the literature to evaluate the lower bound of the optimal solution values for the instances of our PTSP benchmark. The PTSP, besides having a practical interest, has many features that make it is useful as a test problem for algorithms addressing SCOPs, such as: simple formulation, analogy with the well known (deterministic) Traveling Salesman Problem, and most importantly, a closed form expression for the exact objective function. Thus, it is likely it will be considered again in the literature, and for this reason we think that the creation of a benchmark of instances for the PTSP is a useful instrument for future research.

For the PTSP, we have first concentrated on the design of the ACO (Ant Colony Optimization) metaheuristic, second on the design of efficient local search algorithms, and

third, on the integration of local search with ACO and with other heuristics (hybridization). In the following, we summarize the main results and highlight their importance and limitations.

- Our simplest implementation of ACO, which we have called pACS, obtains, without the use of local search, results that can be considered good. In fact, pACS is better than the Farthest Insertion and the Space Filling Curve heuristics both enhanced by local search, which have been considered for long time as the best available algorithms for the PTSP.

- We have evaluated how the performance of pACS depends on the level of stochasticity of an instance. In our benchmark two factors can be considered as expression of the level of stochasticity: the average customers probability and the variance of the customers probability of requiring a visit. When there is no stochasticity (that is, when customers probability is 1 and variance is 0), the PTSP is not different from the deterministic classical Traveling Salesman Problem (TSP), thus, we have compared pACS results with the results obtained by an algorithm for the TSP which solves the TSP to the optimum. We have found that there is a *critical probability*, under which PTSP instances are really worth solving by pACS, but above the critical probability the problem can be better treated like a TSP. The average critical probability is 0.5, but in general, the higher the number of customers, the lower the critical probability. Moreover, the higher the variance of customers probability, the more important is the gain of pACS with respect to the optimal TSP solution.

- We have also considered two versions of pACS that exploit two ad hoc approximations for the PTSP objective function. One of these two approximations is the objective function of the corresponding TSP, and another one is based on neglecting some small terms of the exact objective function. We have shown that the performance of pACS using an ad hoc approximation is strongly correlated with the absolute error of the approximation, and less related with the linear correlation between exact and approximated evaluation. This fact has important consequences, since, when designing ad hoc approximations for a SCOP, one could be tempted to choose one which from preliminary experiments seems well correlated with the exact objective function. But this choice could be, as we have seen, quite bad, and we would suggest to consider, instead, the absolute error of the approximation. However, it is important to stress the fact that the goal of our analysis of ad hoc approximations applied to ACO was not to enhance its performance, since this result has already been achieved by others ([51, 52]), showing that in some cases an ad hoc approximation can accelerate convergence without significantly worsening the solution quality.

- The sampling approximation of the objective function has also been considered in a version of pACS, with the goal to see how worse is the performance with respect to the original pACS algorithm. We found results constantly worse than pACS

of roughly 2%, but the qualitative behavior with respect to factors determining the stochasticity of PTSP instances is the same as the pACS algorithm based on the exact objective.

- One of the nicest results of this thesis is the design of powerful local search algorithms for the PTSP, that do not rely on any objective function approximation. In fact, it has been possible to derive exact recursive and efficient cost expressions for two local search operators, the 2-p-opt and 1-shift. This is a very good result, given that the recursive expressions allow the exploration of the neighborhood of a solution in the same asymptotic time complexity as the (deterministic) TSP, and that, without recursion, the same task would require two orders of magnitude more time. A limitation of this result is that it useful only for the PTSP, since it strongly depends on the structure of its objective function. The experimental evaluation local search has been restricted to the most promising 1-shift local search. The integration of pACS with the exact recursive 1-shift local search leads an average improvement of less than 1%. This is not impressive, but it should be considered that the hybrid version converges faster to good solutions. In fact, the time required for finding the best solution in pACS with local search is about 60% less than without local search.

- In considering the effect of ad hoc approximation on local search, we have observed that, when pACS is hybridized with local search based on two different ad hoc approximations, its performance can be quite bad, both with respect to non-hybrid ACO, and to simple heuristic hybridized with a good local search. Such bad performance of local search based on ad hoc approximations reveals that ad hoc approximations should be chosen very carefully, and that, whenever possible, it is better to spend some effort in searching for exact efficient recursive move cost expressions. When this is impossible, it may still be better to put more effort in improving the performance of the non-hybrid metaheuristic, instead of considering the integration of local search with ACO.

- We have also considered an hybrid version of pACS which only uses evaluations of the objective function and of the local search moves based on the sampling approximation of the objective function. Such algorithm is quite good, since it achieves on average results within 1% of the best hybrid pACS algorithm using the exact objective function. Interestingly, for PTSP instances with high customers probability variance the sampling-based hybrid pACS is the best one. This means that, for this class of PTSP instances, the sampling approximation could be chosen on purpose, even if the exact objective function is available.

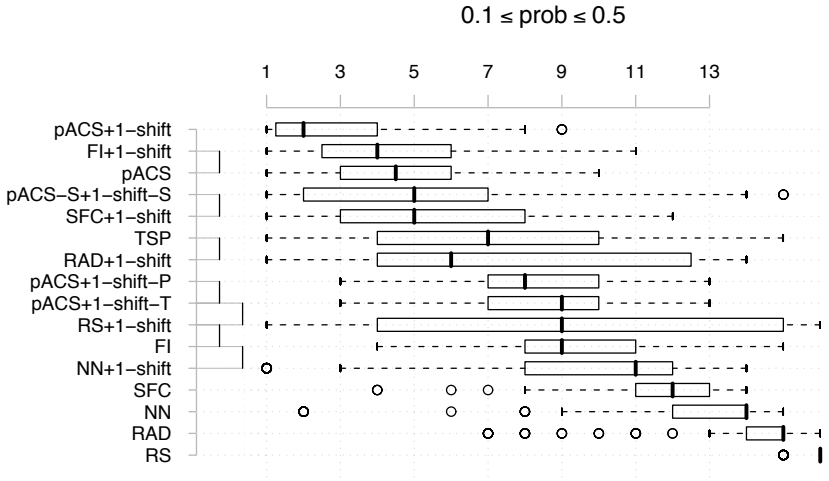
Appendix A

Detailed computational results of ACO algorithms for the PTSP

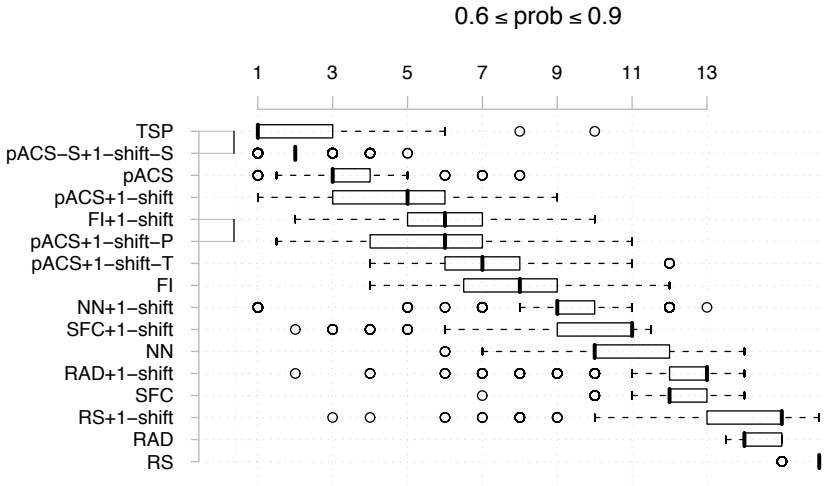
This appendix first presents, in Figure A.1, a complete ranking scheme of most of the algorithms analyzed in this thesis. Secondly, six tables of detailed computational results are reported. The six tables report computational results of two ACO algorithms described in Chapter 5 (pACS and pACS-S) and four hybrid ACO algorithms described in Chapter 7 (pACS+1-shift, pACS-S+1-shift-S, pACS+1-shift-T, and pACS+1-shift-P). Hybrid ACO algorithms combine pACS and pACS-S with three versions of the 1-shift local search presented in Chapter 6.

Experiments have been run on a machine with two processors Intel(R) Xeon(TM) CPU 1.70GHz, running the GNU/Linux Debian 2.4.27 operating system. All algorithms have been coded in C++ under the same development framework. Each algorithm has been run once on each PTSP instance for a computation time equal to $n^2/100$ CPU seconds, where n is the number of customers of the PTSP instance.

The format of the tables is as follows. There are three times six columns. The first of the six columns contains the name of the PTSP instance, from which it is possible to extract informations such as: the corresponding TSP instance with customers coordinates, the number of customers, the average and variance of the customers probability. For detailed informations on the instances of the PTSP benchmark, see Section 4.3. Since, as we have seen in Section 5, above the critical probability the PTSP is better treated by TSP-specific algorithms, our results are interesting particularly for instances with average probability smaller than the critical one. For this reason, we report only results of instances with average probability up to 0.5 (which corresponds to the average critical probability). The second of the six columns reports the PTSP objective function value of the best found solution. The third and the fourth of the six columns contain, respectively, the iteration k_b and the time t_b at which the best solution was found. Finally, the fifth and the sixth of the six columns contain, respectively, the total number of iterations performed k_{tot} and the total time t_{tot} allowed to the algorithm.



(a) Ranking on PTSP instances with low customers probability.



(b) Ranking on PTSP instances with high customers probability. Note that in this case, no algorithm generates significantly better solutions than optimal TSP solutions.

Figure A.1: Ranking of algorithms on the PTSP benchmark. According to the Friedman two-way analysis of variance by ranks [64], the algorithms have statistically significant different ranks, at a confidence level of 95%, when they are not linked by a vertical line on the left of the plot.

pACS

Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}	Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}	Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}
kroA100.p10.v16	9104.9	4736	99.97	4738	100	ch150.p50.v16	4993.8	4000	197.49	4559	" "	att532.p30.v16	56571.0	4454	2663.9	4730	" "
kroA100.p10.v33	8280.7	4776	99.99	4777	" "	ch150.p50.v33	5107.8	3022	148.99	4630	" "	att532.p30.v33	55323.6	4699	2804.26	4742	" "
kroA100.p10.v50	7318.1	4338	89.93	4826	" "	ch150.p50.v50	4915.2	4528	215.76	4714	" "	att532.p30.v50	52790.9	4651	2828.39	4654	" "
kroA100.p10.v66	9507.4	4711	98.98	4764	" "	ch150.p50.v66	4992.4	3960	185.81	4751	" "	att532.p30.v66	55349.7	3648	2175.14	4756	" "
kroA100.p10.v83	9700.7	1872	31.75	5131	" "	ch150.p50.v83	4395.6	4219	204.73	4641	" "	att532.p30.v83	54761.4	4661	2748.36	4798	" "
kroA100.p20.v16	11003.2	4502	95.53	4711	" "	ch150.p10	2493.6	5902	181.51	7279	" "	att532.p40.v16	63021.1	4529	2682.75	4775	" "
kroA100.p20.v33	12008.7	4494	92.74	4855	" "	ch150.p20	3444.2	6794	218.85	6986	" "	att532.p40.v33	63909.0	4482	2683.06	4725	" "
kroA100.p20.v50	11416.7	5160	99.96	5164	" "	ch150.p30	4076.3	5807	186.21	7020	" "	att532.p40.v50	62651.1	4598	2764.17	4708	" "
kroA100.p20.v66	10369.2	4186	86.93	4828	" "	ch150.p40	4636.7	7045	224.04	7075	" "	att532.p40.v66	64815.9	4650	2713.06	4859	" "
kroA100.p20.v83	11402.6	3699	76.13	4837	" "	ch150.p50	5164.3	4945	155.82	7159	" "	att532.p40.v83	60648.0	4841	2813.22	4871	" "
kroA100.p30.v16	14116.2	4271	90.98	4698	" "	d198.p10.v16	8166.3	4407	362.39	4768	" "	att532.p50.v16	72227.3	4593	2781.96	4672	" "
kroA100.p30.v33	12973.2	4210	87.51	4804	" "	d198.p10.v33	7891.4	4874	390.52	4893	" "	att532.p50.v33	67775.5	4554	2720.79	4738	" "
kroA100.p30.v50	13894.6	4578	96.93	4724	" "	d198.p10.v50	7481.7	4524	370.17	4789	" "	att532.p50.v50	67844.2	3114	2579.61	3414	" "
kroA100.p30.v66	12916.7	3949	82.01	4830	" "	d198.p10.v66	6473.9	4139	341.1	4763	" "	att532.p50.v66	68433.3	4860	2824.87	4869	" "
kroA100.p30.v83	12462.4	4724	97.42	4853	" "	d198.p10.v83	6496.9	4116	314.11	5107	" "	att532.p50.v83	72637.3	4883	2827.52	4888	" "
kroA100.p40.v16	15338.9	3611	72.78	4922	" "	d198.p20.v16	9138.0	4533	365.37	4857	" "	att532.p10	35179.7	7296	2793.28	7393	" "
kroA100.p40.v33	14870.9	4544	96.7	4700	" "	d198.p20.v33	9709.2	4618	378.53	4785	" "	att532.p20	47531.4	7386	2819.1	7415	" "
kroA100.p40.v50	16112.4	4747	99.43	4776	" "	d198.p20.v50	8287.4	1349	104.95	5008	" "	att532.p30	55865.3	7115	2788.28	7237	" "
kroA100.p40.v66	14871.3	5104	99.69	5120	" "	d198.p20.v66	9123.9	4511	366.91	4821	" "	att532.p40	63308.0	7204	2817.19	7237	" "
kroA100.p40.v83	13901.5	3207	67.09	4771	" "	d198.p20.v83	9525.9	1227	99.45	4982	" "	att532.p50	69671.2	7210	2822.81	7229	" "
kroA100.p50.v16	16454.6	2861	61.43	4672	" "	d198.p30.v16	11100.2	3004	246.01	4869	" "	rat783.p10.v16	3382.4	4851	6117.61	4861	6131
kroA100.p50.v33	16851.8	3299	69.48	4749	" "	d198.p30.v33	10305.1	4387	356.29	4829	" "	rat783.p10.v33	3320.1	4801	6047.41	4877	" "
kroA100.p50.v50	16825.6	3918	83.5	4688	" "	d198.p30.v50	10463.7	4800	390.26	4822	" "	rat783.p10.v50	3167.7	1425	1781.86	4879	" "
kroA100.p50.v66	16200.8	4590	97.19	4726	" "	d198.p30.v66	12356.7	4894	389.08	4931	" "	rat783.p10.v66	3236.0	4757	6006.06	4857	" "
kroA100.p50.v83	15771.9	2189	45.8	4780	" "	d198.p30.v83	10497.6	3510	278.95	4938	" "	rat783.p10.v83	2732.1	4767	5873.92	4968	" "
kroA100.p10	9039.4	7867	99.64	7897	" "	d198.p40.v16	11817.7	4678	376.14	4873	" "	rat783.p20.v16	4938.1	4855	6122.6	4862	" "
kroA100.p20	11720.6	6721	82.45	8182	" "	d198.p40.v33	13040.6	4707	384.76	4795	" "	rat783.p20.v33	4720.8	4817	6008.55	4914	" "
kroA100.p30	13711.4	6688	83.34	8011	" "	d198.p40.v50	10753.2	4733	388.07	4782	" "	rat783.p20.v50	4680.7	4804	6115.72	4824	" "
kroA100.p40	15253.8	7451	93.79	7945	" "	d198.p40.v66	12649.6	4925	384.74	5019	" "	rat783.p20.v66	4391.3	4662	5704.86	5008	" "
kroA100.p50	16605.4	7712	95.11	8112	" "	d198.p40.v83	11260.9	4729	371.65	4987	" "	rat783.p20.v83	5020.8	4788	5832.12	5030	" "
eil101.p10.v16	184.9	4468	94.88	4808	102	d198.p50.v16	13174.1	4516	369.64	4793	1011	rat783.p30.v16	5768.9	4705	5926.64	4867	" "
eil101.p10.v33	173.7	2340	48.66	4901	" "	d198.p50.v33	12969.6	3903	316.83	4841	" "	rat783.p30.v33	5752.8	4883	6107.47	4901	" "
eil101.p10.v50	149.2	4713	100.4	4791	" "	d198.p50.v50	11945.2	4709	373.04	4944	" "	rat783.p30.v50	5845.3	1732	2183.8	4836	" "
eil101.p10.v66	176.2	3494	72.7	4893	" "	d198.p50.v66	11614.1	4471	353.88	4997	" "	rat783.p30.v66	5641.2	3205	6096.46	3224	" "
eil101.p10.v83	220.9	2488	51.33	4906	" "	d198.p50.v83	13293.1	4957	388.26	5004	" "	rat783.p30.v83	5803.9	4006	4860.76	5039	" "
eil101.p20.v16	276.0	4111	87.52	4790	" "	d198.p10	7556.1	6953	367.45	7419	" "	rat783.p40.v16	6647.8	4733	5903.59	4913	" "
eil101.p20.v33	299.7	4865	100.48	4939	" "	d198.p20	9489.2	7032	373.32	7384	" "	rat783.p40.v33	6714.0	4856	6122.22	4863	" "
eil101.p20.v50	271.4	4814	101.02	4861	" "	d198.p30	10951.9	1044	55.33	7406	" "	rat783.p40.v50	6682.2	3093	5973.12	3174	" "
eil101.p20.v66	277.5	4646	98.4	4817	" "	d198.p40	12047.9	337	17.49	7576	" "	rat783.p40.v66	6559.2	4772	6110.96	4788	" "
eil101.p20.v83	267.0	2852	60.87	4789	" "	d198.p50	12745.5	7209	382.14	7397	" "	rat783.p40.v83	6026.6	4918	6116.8	4929	" "
eil101.p30.v16	346.5	4615	96.67	4871	" "	lin318.p10.v16	18511.7	4691	999.05	4747	" "	rat783.p50.v16	7285.5	4865	6083.56	4903	" "
eil101.p30.v33	366.8	4652	98.65	4812	" "	lin318.p10.v33	18831.3	4673	1007.09	4692	" "	rat783.p50.v33	7222.1	3654	6093.83	3677	" "
eil101.p30.v50	353.8	4829	100.92	4881	" "	lin318.p10.v50	17446.5	4567	1006.69	4588	" "	rat783.p50.v50	7275.9	4917	6105.37	4938	" "
eil101.p30.v66	338.5	3475	73.86	4801	" "	lin318.p10.v66	16941.7	4680	998.93	4736	" "	rat783.p50.v66	7178.3	5036	6125.26	5041	" "
eil101.p30.v83	361.5	4954	95.15	5275	" "	lin318.p10.v83	18288.8	4661	1002.98	4698	" "	rat783.p50.v83	7188.3	4821	5939.14	4976	" "
eil101.p40.v16	399.4	4808	102	4809	" "	lin318.p20.v16	24781.9	4035	892.12	4583	" "	rat783.p10	3368.9	7448	6081.61	7508	" "
eil101.p40.v33	426.5	4169	88.49	4800	" "	lin318.p20.v33	24070.8	4532	1000.97	4579	" "	rat783.p20	4781.2	7638	6105.53	7669	" "
eil101.p40.v50	409.1	4725	100.26	4810	" "	lin318.p20.v50	24228.3	4099	897.68	4614	" "	rat783.p30	5794.0	7396	6055.79	7488	" "
eil101.p40.v66	433.6	4431	84.8	5252	" "	lin318.p20.v66	22356.3	4574	989.16	4676	" "	rat783.p40	6643.6	7351	6004.38	7505	" "
eil101.p40.v83	384.6	4350	91.21	4864	" "	lin318.p20.v83	22020.2	4512	960.26	4744	" "	rat783.p50	7334.1	7331	6000.03	7491	" "
eil101.p50.v16	466.6	3733	79.88	4767	" "	lin318.p30.v16	27840.1	4578	999.69	4630	" "	dsj1000.p10.v16	7877145.0	4122	9928.24	4152	10000
eil101.p50.v33	469.9	3768	78.66	4877	" "	lin318.p30.v33	28665.3	3560	768.75	4673	" "	dsj1000.p10.v33	7360107.9	3945	9448.79	4173	" "
eil101.p50.v50	463.1	4702	100.2	4787	" "	lin318.p30.v50	27947.6	4615	1007.24	4633	" "	dsj1000.p10.v50	7622986.4	4102	9992.69	4106	" "
eil101.p50.v66	431.2	4929	101.84	4938	" "	lin318.p30.v66	28331.7	3941	853.22	4672	" "	dsj1000.p10.v66	8122145.6	3712	8950.18	4159	" "
eil101.p50.v83	392.2	4641	95.45	4963	" "	lin318.p30.v83	26640.7	4416	926.11	4810	" "	dsj1000.p10.v83	7494223.3	2128	4909.98	4322	" "
eil101.p10	199.7	7640	95.11	8190	" "	lin318.p40.v16	31072.0	4742	1010.29	4746	" "	dsj1000.p20.v16	10463471.8	4157	9996.07	4159	" "
eil101.p20	286.7	7461	95.67	7959	" "	lin318.p40.v33	31633.4	4497	962.5	4720	" "	dsj1000.p20.v33	10444565.1	4138	9951.73	4159	" "
eil101.p30	353.5	7646	96.4	8090	" "	lin318.p40.v50	31780.5	4608	1010.57	4611	" "	dsj1000.p20.v50	10647711.3	3552	8647.21	4118	" "
eil101.p40	410.9	6950	87.56	8100	" "	lin318.p40.v66	30802.8	4583	975.41	4749	" "	dsj1000.p20.v66	10329885.8	4261	9965.86	4276	" "
eil101.p50	470.7	7660	99.04	7891	" "	lin318.p40.v83	31151.6	3532	740.23	4781	" "	dsj1000.p20.v83	10594892.2	3788	8822.87	4281	" "
ch150.p10.v16	2465.3	4470	220.53	4562	225	lin318.p50.v16	34790.1	4199	907.26	4676	2830	dsj1000.p30.v16	12656292.2	3942	9496.84	4152	" "
ch150.p10.v33	2301.9	3827	191.21	4505	" "	lin318.p50.v33	33756.3	4205	917.69	4625	" "	dsj1000.p30.v33	12639273.8	4100	9988.05	4105	" "
ch150.p10.v50	2426.2	4512	224.98	4513	" "	lin318.p50.v50	32835.5	4755	1009.01	4766	" "	dsj1000.p30.v50	12121381.1	3954	9496.62	4172	" "
ch150.p10.v66	2462.0	4466	219.09	4600	" "	lin318.p50.v66	34607.4	4704	1009.49	4712	" "	dsj1000.p30.v66	12982963.5	3850	8865.51	4342	" "
ch150.p10.v83	2501.4	2453	121.63	4559	" "	lin318.p50.v83	33085.1	4768	1004.71	4798	" "	dsj1000.p30.v83	11962145.3	4101	9547.72	4286	" "
ch150.p20.v16	3553.0	4232	212.14	4492	" "	lin318.p10	17583.4	7016	994.06	7136	" "	dsj1000.p40.v16	14362558.1	3968	9663.42	4106	" "
ch150.p20.v33	3441.5	4531	223.04	4572	" "	lin318.p20	24413.7	6479	921.46	7108	" "	dsj1000.p40.v33	14248703.2	4095	9965.81	4110	" "
ch150.p20.v50	3301.9	4507	224.88	4511	" "	lin318.p30	28654.4	7107	1009.2	7121	" "	dsj1000.p40.v50	13938082.7	3025	9903.35	3055	" "
ch150.p20.v66	3113.0	4766	222.7	4830	" "	lin318.p40	32035.3	6761	936.93	7294	" "	dsj1000.p40.v66	14314148.8	3862	9196.83	4203	" "
ch150.p20.v83	3513.6	4543	220.99	4626	" "	lin318.p50	34797.1	7088	1005.88	7125	" "	dsj1000.p40.v83	14185609.1	2327	5413.97	4289	" "
ch150.p30.v16	3973.1	4229	211.3	4503	" "	att532.p10.v16	37742.4	4657	2797.11	4711	" "	dsj1000.p50.v16	15599104.5	2894	9987.1	2898	" "
ch150.p30.v33	4096.4	4546	223.42	4579	" "	att532.p10.v33	35825.3	4630	2808.23	4672	" "	dsj1000.p50.v33	15446401.0	3107	9962.8	3120	" "
ch150.p30.v50	4144.8	4742	224.13	4760	" "	att532.p10.v50	32788.1	3588	2191.98	4635	" "	dsj1000.p50.v50	15579476.9	3318	9797.33	3387	" "
ch150.p30.v66	3894.0	4588	222.97	4630	" "	att532.p10.v66	33202.3	4146	2494.98	4704	" "	dsj1000.p50.v66	15141669.2	4266	9949.5	4288	" "
ch150.p30.v83	4019.7	4521	222.53	4572	" "	att532.p10.v83	31511.7	4795	2802.34	4843	" "	dsj1000.p50.v83	14900006.2	4047	9271.85	4366	" "
ch150.p40.v16	4701.1	3770	186.56	4545	" "	att532.p20.v16	47508.7	4632	2797.67	4688	" "	dsj1000.p10	7772825.8	6615	9948.81	6650	" "
ch150.p40.v33	4771.8	4305	210.96	4594	" "	att532.p20.v33	48727.8	1090	662.94	4657	" "	dsj1000.p20	10674105.7	6620	9998.22	6622	" "
ch150.p40.v50	4651.5	3530	176.16	4508	" "	att532.p20.v50	45935.8	4628	2804.55	4671	" "	dsj1000.p30	12564114.6	6727	9944.57	6765	" "
ch150.p40.v66	4410.3	4719	221.2	4799	" "	att532.p20.v66	44800.1	4556	2662.58	4844	" "	dsj1000.p40	14313456.5	6509	9632.96	6758	" "
ch150.p40.v83	4498.8	2261	109.25	4660	" "	att532.p20.v83	45257.9	4620	2693.74	4853	" "	dsj1000.p50	15563658.3	6594	9762.4	6758	" "

Table A.1: Detailed results of pACS for PTSP instances with $0.1 \leq p \leq 0.5$.

pACS-S

Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}	Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}	Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}
kroA100.p10.v16	9182.38	2247	99.82	2251	100	ch150.p50.v16	4983.58	1843	224.01	1848	" "	att532.p30.v16	56961.37	4454	2816.56	1014	" "
kroA100.p10.v33	8293.02	2259	98.34	2280	" "	ch150.p50.v33	5090.03	1849	224.95	1850	" "	att532.p30.v33	56354.99	4699	2784.6	1020	" "
kroA100.p10.v50	7330.31	2288	99.94	2290	" "	ch150.p50.v50	4914.16	1840	223.54	1847	" "	att532.p30.v50	55153.30	4651	2829.36	1025	" "
kroA100.p10.v66	9515.77	2276	99.68	2282	" "	ch150.p50.v66	4922.89	1859	224.66	1862	" "	att532.p30.v66	55373.59	3648	2764.24	1027	" "
kroA100.p10.v83	9700.78	2278	99.68	2283	" "	ch150.p50.v83	4309.73	1832	211.89	1892	" "	att532.p30.v83	56697.74	4661	291.76	1032	" "
kroA100.p20.v16	11152.90	2278	99.98	2279	" "	ch150.p10	2533.73	1903	224.22	1908	" "	att532.p40.v16	64376.03	4529	2793.78	1000	" "
kroA100.p20.v33	12046.54	2318	99.98	2320	" "	ch150.p20	3486.08	1878	224.9	1880	" "	att532.p40.v33	64866.41	4482	2588.81	1006	" "
kroA100.p20.v50	11423.25	2297	99.98	2298	" "	ch150.p30	4290.10	1859	224.89	1861	" "	att532.p40.v50	62777.79	4598	2799.67	1011	" "
kroA100.p20.v66	10386.02	2274	99.86	2276	" "	ch150.p40	4803.82	1864	224.92	1865	" "	att532.p40.v66	65533.29	4650	2810.4	1014	" "
kroA100.p20.v83	11406.31	2281	99.94	2283	" "	ch150.p50	5318.64	1847	224.99	1848	" "	att532.p40.v83	65135.12	4841	2793.47	1017	" "
kroA100.p30.v16	14275.90	2234	99.95	2236	" "	d198.p10.v16	8298.36	4407	374.19	1692	392	att532.p50.v16	72217.84	4593	2821.12	1004	" "
kroA100.p30.v33	12900.53	2241	99.73	2245	" "	d198.p10.v33	7900.93	4874	380.3	1699	" "	att532.p50.v33	68567.59	4554	2805.45	1004	" "
kroA100.p30.v50	14164.21	2232	99.73	2236	" "	d198.p10.v50	7481.83	4524	381.73	1708	" "	att532.p50.v50	70003.69	3114	2815.8	1009	" "
kroA100.p30.v66	12932.81	2236	97.93	2263	" "	d198.p10.v66	6598.56	4139	376.36	1691	" "	att532.p50.v66	69915.05	4860	2764.21	1019	" "
kroA100.p30.v83	12535.02	2274	99.85	2277	" "	d198.p10.v83	6560.45	4116	257.45	1705	" "	att532.p50.v83	72235.53	4883	2388.22	1024	" "
kroA100.p40.v16	15436.64	2213	99.8	2216	" "	d198.p20.v16	9192.41	4533	384.47	1666	" "	att532.p10	36946.82	7296	2828.43	1051	" "
kroA100.p40.v33	14928.29	2231	99.95	2233	" "	d198.p20.v33	9984.28	4618	389.03	1670	" "	att532.p20	48392.53	7386	2828.71	1023	" "
kroA100.p40.v50	16188.03	2230	99.89	2232	" "	d198.p20.v50	8231.83	1349	374.28	1690	" "	att532.p30	57177.77	7115	2829.96	1003	" "
kroA100.p40.v66	14916.15	2242	100	2243	" "	d198.p20.v66	9057.44	4511	88.85	1687	" "	att532.p40	64655.98	7204	2825.99	991	" "
kroA100.p40.v83	14009.87	2239	99.86	2241	" "	d198.p20.v83	9592.74	1227	388.75	1687	" "	att532.p50	70963.86	7210	2827.3	992	" "
kroA100.p50.v16	16572.49	2232	99.91	2234	" "	d198.p30.v16	11044.31	3004	158.51	1643	" "	rat783.p10.v16	3520.95	518	2210.97	883	6131
kroA100.p50.v33	16877.11	2288	99.85	2291	" "	d198.p30.v33	10458.84	4387	358.22	1653	" "	rat783.p10.v33	3475.39	830	5373.94	889	" "
kroA100.p50.v50	16915.17	2225	99.79	2228	" "	d198.p30.v50	10899.03	4800	367.71	1638	" "	rat783.p10.v50	3206.70	855	5633.16	894	" "
kroA100.p50.v66	16278.17	2116	90.71	2234	" "	d198.p30.v66	12340.85	4894	356.51	1671	" "	rat783.p10.v66	3386.69	349	1030.22	892	" "
kroA100.p50.v83	15842.17	2269	99.79	2273	" "	d198.p30.v83	10474.07	3510	45.12	1658	" "	rat783.p10.v83	2820.73	879	5898.91	897	" "
kroA100.p10	9098.05	2284	99.88	2286	" "	d198.p40.v16	12249.24	4678	387.92	1629	" "	rat783.p20.v16	5001.62	849	6061.2	855	" "
kroA100.p20	11819.46	2282	99.89	2284	" "	d198.p40.v33	13179.72	4707	389.94	1637	" "	rat783.p20.v33	4826.23	862	6097.73	865	" "
kroA100.p30	13831.31	2241	99.91	2243	" "	d198.p40.v50	10940.37	4733	379.25	1631	" "	rat783.p20.v50	4833.63	865	6078.9	869	" "
kroA100.p40	15299.55	2225	99.96	2226	" "	d198.p40.v66	13222.28	4925	386.88	1643	" "	rat783.p20.v66	4473.33	871	6057.55	878	" "
kroA100.p50	16793.61	2245	100.01	2246	" "	d198.p40.v83	11220.15	4729	386.31	1640	" "	rat783.p20.v83	5126.43	439	1652.18	874	" "
eil101.p10.v16	185.76	2267	101.89	2270	102	d198.p50.v16	13281.17	4516	384.99	1623	" "	rat783.p30.v16	5904.17	832	6097.41	836	" "
eil101.p10.v33	174.85	2292	101.72	2297	" "	d198.p50.v33	13087.54	3903	369.73	1632	" "	rat783.p30.v33	5870.49	843	6101.92	846	" "
eil101.p10.v50	149.42	2235	97.7	2291	" "	d198.p50.v50	12053.34	4709	381.49	1638	" "	rat783.p30.v50	5919.19	848	6113.06	850	" "
eil101.p10.v66	176.72	2278	101.49	2286	" "	d198.p50.v66	11637.35	4471	391.16	1631	" "	rat783.p30.v66	5950.67	832	5791.09	857	" "
eil101.p10.v83	219.51	2269	101.7	2274	" "	d198.p50.v83	13750.07	4957	348.12	1637	" "	rat783.p30.v83	5772.29	540	2518.15	859	" "
eil101.p20.v16	276.99	2252	101.46	2260	" "	d198.p10	7584.43	6953	388.21	1702	" "	rat783.p40.v16	6704.20	816	6050.99	822	" "
eil101.p20.v33	296.73	2308	101.27	2318	" "	d198.p20	9607.01	7032	391.53	1663	" "	rat783.p40.v33	6839.40	828	6127.4	829	" "
eil101.p20.v50	275.54	2249	100.05	2274	" "	d198.p30	10879.23	1044	384.89	1639	" "	rat783.p40.v50	6804.40	663	3895.72	839	" "
eil101.p20.v66	283.22	2261	99.1	2298	" "	d198.p40	11912.64	337	386.88	1628	" "	rat783.p40.v66	6690.64	805	5618.55	843	" "
eil101.p20.v83	265.52	2282	102	2283	" "	d198.p50	13109.46	7209	363.9	1624	" "	rat783.p40.v83	6189.75	773	5149.35	847	" "
eil101.p30.v16	352.81	2255	101.94	2257	" "	lin318.p10.v16	19171.90	4691	1008.89	1351	1011	rat783.p50.v16	7327.53	805	5932.52	819	" "
eil101.p30.v33	367.66	2220	101.93	2222	" "	lin318.p10.v33	18298.50	4673	988.18	1360	" "	rat783.p50.v33	7335.12	819	6033.69	826	" "
eil101.p30.v50	362.76	2242	101.82	2245	" "	lin318.p10.v50	17374.96	4567	994.48	1353	" "	rat783.p50.v50	7540.76	832	6093.82	835	" "
eil101.p30.v66	341.47	2233	101.77	2237	" "	lin318.p10.v66	18123.88	4680	954.57	1366	" "	rat783.p50.v66	7388.08	841	6121.39	842	" "
eil101.p30.v83	349.61	2246	101.63	2252	" "	lin318.p10.v83	18684.73	4661	749.14	1356	" "	rat783.p50.v83	7401.50	832	5970.7	845	" "
eil101.p40.v16	405.59	2206	101.99	2207	" "	lin318.p20.v16	25199.26	4035	1003.92	1322	" "	rat783.p10	3472.58	869	6100.19	872	" "
eil101.p40.v33	432.45	2228	101.73	2232	" "	lin318.p20.v33	24510.45	4532	904.26	1326	" "	rat783.p20	4863.61	844	6120.03	846	" "
eil101.p40.v50	410.60	2289	101.92	2291	" "	lin318.p20.v50	25001.38	4099	307.14	1327	" "	rat783.p30	5888.89	815	6016.13	824	" "
eil101.p40.v66	434.75	2199	99.53	2230	" "	lin318.p20.v66	22609.73	4574	902.38	1336	" "	rat783.p40	6707.93	807	6107.34	809	" "
eil101.p40.v83	369.67	2240	101.67	2245	" "	lin318.p20.v83	22061.35	4512	1002.61	1338	" "	rat783.p50	7423.20	806	6103.95	808	" "
eil101.p50.v16	476.90	2232	101.68	2237	" "	lin318.p30.v16	28093.94	4578	1010.16	1306	" "	dsj1000.p10.v16	8066789.86	762	9963.63	764	10000
eil101.p50.v33	479.17	2238	101.9	2240	" "	lin318.p30.v33	28674.76	3560	1008.25	1304	" "	dsj1000.p10.v33	7673655.47	758	9638.42	773	" "
eil101.p50.v50	476.94	2246	101.97	2247	" "	lin318.p30.v50	28512.86	4615	1007.64	1303	" "	dsj1000.p10.v50	7961863.87	761	9693.95	774	" "
eil101.p50.v66	431.68	2222	101.19	2233	" "	lin318.p30.v66	29255.38	3941	1010.26	1308	" "	dsj1000.p10.v66	8186052.70	624	6514.5	780	" "
eil101.p50.v83	417.87	2250	101.68	2255	" "	lin318.p30.v83	26434.57	4416	988.57	1318	" "	dsj1000.p10.v83	7332409.55	780	9991.78	782	" "
eil101.p10	201.68	2285	101.92	2287	" "	lin318.p40.v16	32049.58	4742	1009.88	1290	" "	dsj1000.p20.v16	10799931.97	729	9935.55	733	" "
eil101.p20	287.97	2272	101.92	2274	" "	lin318.p40.v33	32387.46	4497	1010.53	1290	" "	dsj1000.p20.v33	10770686.78	737	9914.45	741	" "
eil101.p30	357.49	2240	101.94	2241	" "	lin318.p40.v50	32484.74	4608	972.08	1294	" "	dsj1000.p20.v50	10813920.38	744	9881.3	749	" "
eil101.p40	421.07	2227	101.85	2230	" "	lin318.p40.v66	31596.57	4583	993.87	1292	" "	dsj1000.p20.v66	10292914.12	681	8155.47	757	" "
eil101.p50	476.03	2243	102.01	2244	" "	lin318.p40.v83	31575.20	3532	1010.03	1302	" "	dsj1000.p20.v83	10936611.85	760	10000.11	761	" "
ch150.p10.v16	2524.34	1899	223.62	1907	225	lin318.p50.v16	35203.54	4199	973.61	1294	" "	dsj1000.p30.v16	12916320.79	699	9793.78	708	" "
ch150.p10.v33	2289.85	1899	223.71	1906	" "	lin318.p50.v33	34422.99	4205	1007.98	1287	" "	dsj1000.p30.v33	13072731.61	700	9567.12	717	" "
ch150.p10.v50	2430.43	1915	224.79	1917	" "	lin318.p50.v50	33420.74	4755	1009.4	1296	" "	dsj1000.p30.v50	12381820.80	731	9961.54	733	" "
ch150.p10.v66	2469.26	1751	190.24	1919	" "	lin318.p50.v66	34481.07	4704	1002.48	1301	" "	dsj1000.p30.v66	12802724.67	715	9492.48	735	" "
ch150.p10.v83	2502.02	1924	222.86	1935	" "	lin318.p50.v83	33916.41	4768	1006.95	1292	" "	dsj1000.p30.v83	12323164.62	435	3551.38	745	" "
ch150.p20.v16	3518.86	1874	224.91	1875	" "	lin318.p10	18130.60	7016	993.68	1347	" "	dsj1000.p40.v16	14507685.54	679	9796.68	688	" "
ch150.p20.v33	3521.56	1885	225.01	1886	" "	lin318.p20	24924.10	6479	1010.77	1319	" "	dsj1000.p40.v33	14494379.47	687	9684.42	700	" "
ch150.p20.v50	3347.31	1885	223.05	1895	" "	lin318.p30	29922.82	7107	1009.12	1302	" "	dsj1000.p40.v50	14431883.24	712	9992.95	713	" "
ch150.p20.v66	3178.82	1747	192.05	1905	" "	lin318.p40	32737.70	6761	1010.6	1290	" "	dsj1000.p40.v66	14700442.81	498	4910.97	721	" "
ch150.p20.v83	3375.21	1895	223.34	1903	" "	lin318.p50	35559.28	7088	1008.33	1287	" "	dsj1000.p40.v83	14183887.09	727	9904.61	731	" "
ch150.p30.v16	4057.14	1849	224.61	1852	" "	att532.p10.v16	38841.24	4657	2812.02	1050	2830	dsj1000.p50.v16	15682511.91	679	9982.77	680	" "
ch150.p30.v33	4407.36	1830	218.91	1859	" "	att532.p10.v33	36114.86	4630	2670.12	1069	" "	dsj1000.p50.v33	15749031.67	686	9854.33	692	" "
ch150.p30.v50	4025.01	1862	223.99	1868	" "	att532.p10.v50	35380.47	3588	2793.17	1078	" "	dsj1000.p50.v50	15870168.49	695	9736.25	706	" "
ch150.p30.v66	4089.49	1780	207.78	1860	" "	att532.p10.v66	34038.21	4146	2641.78	1077	" "	dsj1000.p50.v66	15503076.08	710	9752.6	720	" "
ch150.p30.v83	4021.93	1099	84.64	1884	" "	att532.p10.v83	32677.09	4795	2738.61	1083	" "	dsj1000.p50.v83	15135976.98	141	465.43	728	" "
ch150.p40.v16	4695.92	1838	224.87	1840	" "	att532.p20.v16	48940.30	4632	2822.02	1035	" "	dsj1000.p10	7947090.75	756	9909.45	760	" "
ch150.p40.v33	4779.80	1844	224.18	1848	" "	att532.p20.v33	47846.35	1090	2681.44	1042	" "	dsj1000.p20	10855790.85	720	9918.87	724	" "
ch150.p40.v50	4646.98	1845	224.55	1848	" "	att532.p20.v50	46725.65	4628	2781.22	1052	" "	dsj1000.p30	12772805.12	696	9946.66	699	" "
ch150.p40.v66	4574.77	1850	223.85	1856	" "	att532.p20.v66	47116.98	4556	2804.57	1041	" "	dsj1000.p40	14353182.94	679	9997.57	680	" "
ch150.p40.v83	4544.59	1880	224.13	1884	" "	att532.p20.v83	48522.20	4620	1415.78	1049	" "	dsj1000.p50	15741284.59	664	9854.02	670	" "

Table A.2: Detailed results of pACS-S for PTSP instances with $0.1 \leq p \leq 0.5$.

pACS+1-shift

Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}	Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}	Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}
kroA100.p10.v16	8279.7	1	47.16	17	100	ch150.p50.v16	4979.5	16	202.43	29	" "	att532.p30.v16	52469.3	1	439.84	4	" "
kroA100.p10.v33	7318.1	1	38.43	15	" "	ch150.p50.v33	4763.4	25	209.6	28	" "	att532.p30.v33	51282.7	1	317.82	3	" "
kroA100.p10.v50	9507.0	1	60.73	17	" "	ch150.p50.v50	4953.4	1	211.17	16	" "	att532.p30.v50	53201.8	1	99.39	3	" "
kroA100.p10.v66	9700.7	1	7	19	" "	ch150.p50.v66	4588.7	18	218.24	20	" "	att532.p30.v66	59588.5	1	1214.73	3	" "
kroA100.p10.v83	10969.9	1	0.21	19	" "	ch150.p50.v83	5344.3	83	215.75	87	" "	att532.p30.v83	61735.2	38	1357.8	81	" "
kroA100.p20.v16	11997.1	1	4.74	33	" "	ch150.p10	3418.3	1	143.55	19	" "	att532.p40.v16	62660.0	1	504.31	4	" "
kroA100.p20.v33	11416.5	1	15.14	41	" "	ch150.p20	4053.5	1	223.42	34	" "	att532.p40.v33	61540.0	1	568.95	4	" "
kroA100.p20.v50	10360.8	1	29.66	34	" "	ch150.p30	4581.4	18	98.85	77	" "	att532.p40.v50	67870.7	1	532.68	4	" "
kroA100.p20.v66	11414.8	1	37.8	16	" "	ch150.p40	5030.2	30	107.42	105	" "	att532.p40.v66	64386.0	74	2554.54	86	" "
kroA100.p20.v83	14081.7	68	75.82	114	" "	ch150.p50	5467.1	28	192.42	81	" "	att532.p40.v83	69773.5	52	1951.11	77	" "
kroA100.p30.v16	12828.8	1	33.8	48	" "	d198.p10.v16	7809.8	1	23.18	4	392	att532.p50.v16	66369.5	1	56.15	4	" "
kroA100.p30.v33	13880.7	25	72.78	35	" "	d198.p10.v33	7373.7	1	284.48	5	" "	att532.p50.v33	67862.0	1	479.28	4	" "
kroA100.p30.v50	12916.7	1	40.63	38	" "	d198.p10.v50	6451.1	1	9.92	6	" "	att532.p50.v50	70631.0	1	328.25	2	" "
kroA100.p30.v66	12630.9	1	0.87	22	" "	d198.p10.v66	6589.3	1	85.26	7	" "	att532.p50.v66	73302.4	82	2795.44	84	" "
kroA100.p30.v83	15302.8	100	97.36	105	" "	d198.p10.v83	8927.8	99	343.4	113	" "	att532.p50.v83	74189.3	70	2692.83	80	" "
kroA100.p40.v16	14868.8	18	51.12	99	" "	d198.p20.v16	9504.5	1	55.86	5	" "	att532.p10	45755.5	1	1095.67	3	" "
kroA100.p40.v33	16093.9	1	30.88	54	" "	d198.p20.v33	7898.0	1	55.04	5	" "	att532.p20	55197.4	1	424.27	4	" "
kroA100.p40.v50	15115.5	1	64.46	55	" "	d198.p20.v50	9578.7	1	279.22	5	" "	att532.p30	62285.2	1	469.29	5	" "
kroA100.p40.v66	14040.9	30	30.52	122	" "	d198.p20.v66	9676.4	33	128.99	99	" "	att532.p40	69346.5	1	2314.42	6	" "
kroA100.p40.v83	16454.1	78	86.22	101	" "	d198.p20.v83	10716.9	96	320.12	117	" "	att532.p50	74465.2	1	207.02	7	" "
kroA100.p50.v16	16850.3	65	97.89	67	" "	d198.p30.v16	9977.5	1	96.52	7	" "	rat783.p10.v16	3065.1	1	2016.47	3	6131
kroA100.p50.v33	16564.6	24	51.24	97	" "	d198.p30.v33	10181.7	1	273	7	" "	rat783.p10.v33	2946.5	1	2098.74	2	" "
kroA100.p50.v50	16186.7	1	60.29	36	" "	d198.p30.v50	12146.1	1	275.55	7	" "	rat783.p10.v50	2978.5	1	3665.17	2	" "
kroA100.p50.v66	15845.5	1	29.55	40	" "	d198.p30.v66	10858.9	1	195.01	6	" "	rat783.p10.v66	3003.2	1	3415.33	3	" "
kroA100.p50.v83	17499.0	36	39.47	91	" "	d198.p30.v83	11801.6	72	310.32	92	" "	rat783.p10.v83	4722.8	51	3553.29	92	" "
kroA100.p10	11715.1	1	16.25	50	" "	d198.p40.v16	12871.4	1	35.84	8	" "	rat783.p20.v16	4442.9	1	3680.9	3	" "
kroA100.p20	13679.3	1	38.52	105	" "	d198.p40.v33	10890.1	1	347.05	10	" "	rat783.p20.v33	4392.6	1	4202.93	3	" "
kroA100.p30	15253.6	1	28.69	105	" "	d198.p40.v50	13773.8	1	372.08	8	" "	rat783.p20.v50	4529.9	1	1180.19	3	" "
kroA100.p40	16569.7	24	27.87	178	" "	d198.p40.v66	11918.3	3	240.14	114	" "	rat783.p20.v66	5275.4	1	881.96	74	" "
kroA100.p50	17723.7	1	56.09	183	" "	d198.p40.v83	13100.5	83	390.6	85	" "	rat783.p20.v83	5546.6	67	5444.93	76	" "
eil101.p10.v16	172.6	1	55.28	13	102	d198.p50.v16	12467.9	1	358.03	9	" "	rat783.p30.v16	5367.4	1	1906.88	3	" "
eil101.p10.v33	149.1	1	88.18	12	" "	d198.p50.v33	11782.2	1	3.95	10	" "	rat783.p30.v33	5461.3	1	2321.88	3	" "
eil101.p10.v50	176.2	1	5.25	13	" "	d198.p50.v50	12496.4	1	298.53	9	" "	rat783.p30.v50	5927.4	1	921.67	3	" "
eil101.p10.v66	219.4	1	1.62	18	" "	d198.p50.v66	14104.3	84	365.17	99	" "	rat783.p30.v66	5767.0	40	4532.02	58	" "
eil101.p10.v83	270.0	1	0.85	25	" "	d198.p50.v83	13333.3	85	380.6	88	" "	rat783.p30.v83	6390.1	17	1414.47	78	" "
eil101.p20.v16	293.8	1	47.99	15	" "	d198.p10	9312.1	1	210.39	9	" "	rat783.p40.v16	6446.7	1	252.23	3	" "
eil101.p20.v33	267.9	1	88.05	14	" "	d198.p20	10645.9	1	59.69	9	" "	rat783.p40.v33	6747.1	1	763.8	3	" "
eil101.p20.v50	277.5	1	89.83	16	" "	d198.p30	11631.5	1	232.58	11	" "	rat783.p40.v50	6829.9	81	5341.53	95	" "
eil101.p20.v66	277.7	1	43.19	16	" "	d198.p40	12666.5	1	284.23	16	" "	rat783.p40.v66	6178.2	18	1464.7	81	" "
eil101.p20.v83	344.7	124	99.85	127	" "	d198.p50	13539.7	1	308.65	16	" "	rat783.p40.v83	7088.5	1	1009.71	80	" "
eil101.p30.v16	353.4	1	66.79	20	" "	lin318.p10.v16	17556.2	1	980.88	4	1011	rat783.p50.v16	6973.7	1	172.59	3	" "
eil101.p30.v33	349.7	1	85.49	19	" "	lin318.p10.v33	16788.8	1	188.88	4	" "	rat783.p50.v33	7475.4	1	3365	3	" "
eil101.p30.v50	334.7	1	79.54	20	" "	lin318.p10.v50	19755.4	1	232.65	4	" "	rat783.p50.v50	7421.6	82	5705.81	89	" "
eil101.p30.v66	385.0	1	39.32	15	" "	lin318.p10.v66	19158.1	1	43.5	103	" "	rat783.p50.v66	7386.9	81	6030.79	83	" "
eil101.p30.v83	393.0	71	63.77	113	" "	lin318.p10.v83	24587.6	116	1004.73	117	" "	rat783.p50.v83	7987.6	57	4547.67	77	" "
eil101.p40.v16	418.6	1	90.54	40	" "	lin318.p20.v16	22957.6	1	130.36	5	" "	rat783.p10	4619.2	1	1234.98	3	" "
eil101.p40.v33	383.7	1	95.87	24	" "	lin318.p20.v33	23822.4	1	744.53	5	" "	rat783.p20	5672.8	1	1555.04	3	" "
eil101.p40.v50	452.6	6	95.2	30	" "	lin318.p20.v50	21881.9	1	198.98	4	" "	rat783.p30	6516.2	1	320.25	4	" "
eil101.p40.v66	395.3	71	64.02	113	" "	lin318.p20.v66	22589.5	1	231.6	4	" "	rat783.p40	7199.0	1	221.2	4	" "
eil101.p40.v83	454.7	88	96.54	93	" "	lin318.p20.v83	27411.2	92	983.32	95	" "	rat783.p50	7785.5	1	682.03	5	" "
eil101.p50.v16	456.4	32	97.04	35	" "	lin318.p30.v16	28360.4	1	173.58	6	" "	dsj1000.p10.v16	7072023.4	1	1405.43	2	10000
eil101.p50.v33	450.0	25	91.03	29	" "	lin318.p30.v33	27552.9	1	850.98	6	" "	dsj1000.p10.v33	7241096.9	1	9110.1	2	" "
eil101.p50.v50	464.2	1	91.69	22	" "	lin318.p30.v50	27541.5	1	234.72	5	" "	dsj1000.p10.v50	6875873.6	1	2985.8	2	" "
eil101.p50.v66	434.5	93	93.69	102	" "	lin318.p30.v66	30091.0	1	304.01	5	" "	dsj1000.p10.v66	7603393.8	1	6701.61	2	" "
eil101.p50.v83	506.0	63	66.16	97	" "	lin318.p30.v83	31184.7	51	633.75	87	" "	dsj1000.p10.v83	9992975.8	18	2152.05	83	" "
eil101.p10	283.6	1	1.21	30	" "	lin318.p40.v16	31734.9	1	768.93	7	" "	dsj1000.p20.v16	9840719.2	1	9999.42	2	" "
eil101.p20	349.2	1	73.25	32	" "	lin318.p40.v33	31656.1	1	915.68	7	" "	dsj1000.p20.v33	9953525.6	1	7854.33	2	" "
eil101.p30	404.7	9	47.26	72	" "	lin318.p40.v50	34028.0	1	957.72	7	" "	dsj1000.p20.v50	10536279.5	1	9960.55	2	" "
eil101.p40	454.5	14	60.31	84	" "	lin318.p40.v66	34738.0	71	686.46	106	" "	dsj1000.p20.v66	10941566.1	1	1363.86	60	" "
eil101.p50	500.6	12	78.63	110	" "	lin318.p40.v83	34505.6	32	388.99	83	" "	dsj1000.p20.v83	12157464.8	1	6275.69	72	" "
ch150.p10.v16	2278.3	1	92.32	7	225	lin318.p50.v16	32922.7	1	858.91	9	" "	dsj1000.p30.v16	11933542.3	1	4873.86	3	" "
ch150.p10.v33	2423.3	1	150.51	7	" "	lin318.p50.v33	32604.1	1	851.87	8	" "	dsj1000.p30.v33	11499618.5	1	6369.15	2	" "
ch150.p10.v50	2460.4	1	203.41	8	" "	lin318.p50.v50	35782.0	1	1010.57	7	" "	dsj1000.p30.v50	13260366.4	1	2813.75	3	" "
ch150.p10.v66	2501.4	1	109.16	10	" "	lin318.p50.v66	35763.7	76	985.57	94	" "	dsj1000.p30.v66	12750737.3	31	5838.84	58	" "
ch150.p10.v83	3444.1	98	199.62	111	" "	lin318.p50.v83	36885.9	19	299.62	87	" "	dsj1000.p30.v83	13815793.3	21	3577.4	63	" "
ch150.p20.v16	3390.7	1	184.53	12	" "	lin318.p10	23719.9	1	869.14	6	" "	dsj1000.p40.v16	13820422.7	1	1033.16	3	" "
ch150.p20.v33	3284.0	1	221.01	11	" "	lin318.p20	28945.8	1	605.64	8	" "	dsj1000.p40.v33	14968529.9	1	2343	3	" "
ch150.p20.v50	3107.0	1	163.72	10	" "	lin318.p30	31676.6	1	657.24	9	" "	dsj1000.p40.v50	14737781.8	44	7652.21	61	" "
ch150.p20.v66	3511.5	1	72.6	8	" "	lin318.p40	34575.3	1	1000.21	16	" "	dsj1000.p40.v66	14269100.2	1	1100.66	65	" "
ch150.p20.v83	3941.2	20	42.64	103	" "	lin318.p50	36774.7	13	956.23	22	" "	dsj1000.p40.v83	15717063.4	1	56.12	63	" "
ch150.p30.v16	4035.8	1	208.23	13	" "	att532.p10.v16	34145.9	1	360.52	3	2830	dsj1000.p50.v16	15976411.2	96	8906.4	109	" "
ch150.p30.v33	3960.2	1	156.98	12	" "	att532.p10.v33	33662.9	1	2094.49	3	" "	dsj1000.p50.v33	15916478.1	27	3735.58	83	" "
ch150.p30.v50	3892.6	1	139.73	14	" "	att532.p10.v50	31982.0	1	1646.95	3	" "	dsj1000.p50.v50	15792848.0	11	4120.13	77	" "
ch150.p30.v66	4247.8	1	183.42	10	" "	att532.p10.v66	32594.8	1	725.65	3	" "	dsj1000.p50.v66	15201985.7	1	2365.17	70	" "
ch150.p30.v83	4540.5	86	213.64	91	" "	att532.p10.v83	46120.8	86	2630.3	93	" "	dsj1000.p50.v83	17093520.2	1	9.04	71	" "
ch150.p40.v16	4649.8	24	219.88	25	" "	att532.p20.v16	45123.6	1	355.95	4	" "	dsj1000.p10	10055741.1	1	1861.05	3	" "
ch150.p40.v33	4626.5	1	206.98	18	" "	att532.p20.v33	44572.6	1	302.66	3	" "	dsj1000.p20	12076064.1	1	1224.55	3	" "
ch150.p40.v50	4557.8	1	213.13	16	" "	att532.p20.v50	48693.1	1	1153.64	3	" "	dsj1000.p30	13756464.9	1	5784.11	3	" "
ch150.p40.v66	4843.6	86	192.3	115	" "	att532.p20.v66	49473.9	1	570.46	82	" "	dsj1000.p40	15209693.8	1	346.11	3	" "
ch150.p40.v83	4943.4	51	133.78	86	" "	att532.p20.v83	54615.5	81	2820.9	82	" "	dsj1000.p50	17021037.7	1	292.43	4	" "

Table A.3: Detailed results of pACS+1-shift for PTSP instances with $0.1 \leq p \leq 0.5$.

pACS-S+1-shift-S

Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}	Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}	Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}
kroA100.p10.v16	9230.3	610	99.78	614	100	ch150.p50.v16	4917.1	273	223.6	275	" "	att532.p30.v16	56022.5	152	2673.7	187	" "
kroA100.p10.v33	8294.0	1339	99.73	1341	" "	ch150.p50.v33	4953.2	265	223.57	271	" "	att532.p30.v33	55023.4	11	2081.18	183	" "
kroA100.p10.v50	7354.7	1440	99.89	1442	" "	ch150.p50.v50	4678.7	265	222.85	275	" "	att532.p30.v50	53958.5	1	1470.19	161	" "
kroA100.p10.v66	9518.9	1487	98.96	1497	" "	ch150.p50.v66	4846.5	364	223.22	370	" "	att532.p30.v66	50676.7	1	2156.79	153	" "
kroA100.p10.v83	9701.0	1663	99.94	1665	" "	ch150.p50.v83	4279.9	245	224.62	308	" "	att532.p30.v83	51855.9	1	1894.72	131	" "
kroA100.p20.v16	11312.0	341	95.36	361	" "	ch150.p10	2814.2	293	218.77	315	" "	att532.p40.v16	62975.9	202	2827.03	203	" "
kroA100.p20.v33	12577.8	288	98.79	347	" "	ch150.p20	3539.1	181	169.35	272	" "	att532.p40.v33	64045.9	38	595.97	186	" "
kroA100.p20.v50	11476.3	408	97.65	421	" "	ch150.p30	4118.5	275	223.99	277	" "	att532.p40.v50	63235.9	160	2757.45	165	" "
kroA100.p20.v66	10398.1	1179	95	1229	" "	ch150.p40	4574.2	275	224.9	276	" "	att532.p40.v66	61581.3	68	1720.68	157	" "
kroA100.p20.v83	11417.4	1126	99.52	1133	" "	ch150.p50	5051.3	271	222.25	275	" "	att532.p40.v83	62493.8	1	2371.25	126	" "
kroA100.p30.v16	14477.0	190	66.95	291	" "	d198.p10.v16	8372.7	287	391.25	288	392	att532.p50.v16	69857.5	184	2709.58	192	" "
kroA100.p30.v33	12887.6	310	99.19	314	" "	d198.p10.v33	8216.5	105	376.13	276	" "	att532.p50.v33	66431.1	159	2507.8	187	" "
kroA100.p30.v50	13952.7	353	92.25	382	" "	d198.p10.v50	7553.0	139	315.07	276	" "	att532.p50.v50	68192.3	1	1664.71	161	" "
kroA100.p30.v66	12922.9	577	99.66	589	" "	d198.p10.v66	6513.8	139	373.65	290	" "	att532.p50.v66	66841.3	1	2667.27	147	" "
kroA100.p30.v83	12467.3	1148	99.86	1152	" "	d198.p10.v83	6487.5	5	258.26	361	" "	att532.p50.v83	67911.4	133	2793.06	156	" "
kroA100.p40.v16	15411.1	317	99.53	319	" "	d198.p20.v16	9537.2	83	374.83	275	" "	att532.p10	40662.6	1	2196.46	195	" "
kroA100.p40.v33	14933.0	318	99.87	319	" "	d198.p20.v33	10131.4	21	99.61	257	" "	att532.p20	48748.1	1	2521.97	197	" "
kroA100.p40.v50	16101.1	293	98.08	299	" "	d198.p20.v50	8241.3	33	190.3	258	" "	att532.p30	57785.7	1	50.42	195	" "
kroA100.p40.v66	14956.4	357	99.98	368	" "	d198.p20.v66	9115.5	14	385.18	235	" "	att532.p40	64084.5	198	2807.4	200	" "
kroA100.p40.v83	13901.4	1015	99.12	1025	" "	d198.p20.v83	9380.8	11	316.2	265	" "	att532.p50	69650.1	201	2830.05	204	" "
kroA100.p50.v16	16476.6	313	99.33	316	" "	d198.p30.v16	11131.0	46	371.94	256	" "	rat783.p10.v16	3495.0	1	61.4	165	6131
kroA100.p50.v33	16863.1	308	100	309	" "	d198.p30.v33	10347.9	18	225.48	245	" "	rat783.p10.v33	3460.7	1	2010.97	148	" "
kroA100.p50.v50	16637.7	317	99.49	320	" "	d198.p30.v50	10172.3	51	316.28	244	" "	rat783.p10.v50	3106.2	1	945.98	155	" "
kroA100.p50.v66	16247.8	338	99.86	342	" "	d198.p30.v66	12076.4	24	357.85	216	" "	rat783.p10.v66	3201.8	1	2091.15	126	" "
kroA100.p50.v83	15771.9	1331	94.01	1415	" "	d198.p30.v83	10133.3	2	183.33	212	" "	rat783.p10.v83	2581.3	1	5038.71	115	" "
kroA100.p10	9161.6	681	99.67	685	" "	d198.p40.v16	11933.0	184	275.7	259	" "	rat783.p20.v16	5014.1	10	1106.83	170	" "
kroA100.p20	12241.7	250	85.31	326	" "	d198.p40.v33	13147.0	28	187.92	246	" "	rat783.p20.v33	4721.2	1	5437.24	151	" "
kroA100.p30	13787.0	308	99.91	311	" "	d198.p40.v50	11045.9	5	377.86	246	" "	rat783.p20.v50	4743.9	1	2559.21	136	" "
kroA100.p40	15286.8	324	99.96	325	" "	d198.p40.v66	12682.9	2	134.02	232	" "	rat783.p20.v66	4320.9	1	1805.49	137	" "
kroA100.p50	16679.3	283	96.98	298	" "	d198.p40.v83	11417.1	7	372.15	231	" "	rat783.p20.v83	4451.3	1	5655.17	99	" "
eil101.p10.v16	197.6	495	96.19	523	102	d198.p50.v16	12988.3	156	235.75	256	" "	rat783.p30.v16	5888.0	17	3964.24	155	" "
eil101.p10.v33	180.9	525	96.37	554	" "	d198.p50.v33	12733.0	8	245.4	251	" "	rat783.p30.v33	5711.8	1	3379	144	" "
eil101.p10.v50	149.3	776	100	780	" "	d198.p50.v50	11494.0	38	349.38	244	" "	rat783.p30.v50	5749.7	60	3089	151	" "
eil101.p10.v66	177.6	1193	97.26	1251	" "	d198.p50.v66	11625.5	19	370.74	236	" "	rat783.p30.v66	5570.0	1	5893.04	121	" "
eil101.p10.v83	219.8	946	100	948	" "	d198.p50.v83	13223.7	2	292.98	189	" "	rat783.p30.v83	5338.1	1	931.08	104	" "
eil101.p20.v16	291.9	162	100	292	" "	d198.p10	8028.3	281	385.78	286	" "	rat783.p40.v16	6539.0	164	5711.75	176	" "
eil101.p20.v33	303.5	288	100	289	" "	d198.p20	10071.9	218	310.32	272	" "	rat783.p40.v33	6708.5	8	4226.63	153	" "
eil101.p20.v50	284.9	74	98.38	285	" "	d198.p30	11290.9	22	381.58	263	" "	rat783.p40.v50	6668.9	1	3580.08	132	" "
eil101.p20.v66	280.8	200	79.66	359	" "	d198.p40	11978.5	230	342.86	262	" "	rat783.p40.v66	6485.3	1	3125.67	122	" "
eil101.p20.v83	267.4	536	100	587	" "	d198.p50	12613.3	128	388.13	255	" "	rat783.p40.v83	5969.7	1	3279.12	107	" "
eil101.p30.v16	368.9	229	100.11	284	" "	lin318.p10.v16	19454.4	42	169.28	231	1011	rat783.p50.v16	7126.1	152	6066.69	164	" "
eil101.p30.v33	368.4	224	97.19	280	" "	lin318.p10.v33	18857.1	1	541.93	214	" "	rat783.p50.v33	7239.7	1	5648.78	148	" "
eil101.p30.v50	368.3	180	101.37	277	" "	lin318.p10.v50	17678.2	4	927.92	216	" "	rat783.p50.v50	7343.6	1	5748.98	133	" "
eil101.p30.v66	343.6	191	100.94	291	" "	lin318.p10.v66	17211.9	1	236.28	197	" "	rat783.p50.v66	7165.0	1	4077.72	120	" "
eil101.p30.v83	348.8	593	101.34	597	" "	lin318.p10.v83	18044.3	1	827.7	173	" "	rat783.p50.v83	7014.6	1	5229.56	105	" "
eil101.p40.v16	402.2	275	100.17	281	" "	lin318.p20.v16	26036.7	9	954.25	224	" "	rat783.p10	3514.5	1	31.54	172	" "
eil101.p40.v33	430.8	254	97.81	279	" "	lin318.p20.v33	25083.2	23	601.76	211	" "	rat783.p20	4943.5	101	3769.11	162	" "
eil101.p40.v50	390.1	176	99	272	" "	lin318.p20.v50	24357.1	1	694.16	204	" "	rat783.p30	5855.0	133	4659.63	173	" "
eil101.p40.v66	434.9	251	101.71	272	" "	lin318.p20.v66	22197.5	2	682.63	197	" "	rat783.p40	6622.7	167	5795.25	176	" "
eil101.p40.v83	356.4	531	100.4	547	" "	lin318.p20.v83	21627.9	1	447.03	168	" "	rat783.p50	7261.4	176	6111.74	177	" "
eil101.p50.v16	460.2	271	100.16	280	" "	lin318.p30.v16	29657.2	1	163.9	219	" "	dsj1000.p10.v16	8181221.0	1	4665.53	141	10000
eil101.p50.v33	462.2	271	101.16	274	" "	lin318.p30.v33	29344.1	1	714.7	211	" "	dsj1000.p10.v33	7824935.1	1	3399.16	130	" "
eil101.p50.v50	455.5	231	91.06	270	" "	lin318.p30.v50	28710.8	1	365.57	196	" "	dsj1000.p10.v50	7659801.0	1	7355.16	123	" "
eil101.p50.v66	429.9	307	101.31	311	" "	lin318.p30.v66	27845.2	1	905.47	184	" "	dsj1000.p10.v66	7277949.9	1	9323.73	108	" "
eil101.p50.v83	387.8	614	97.47	644	" "	lin318.p30.v83	27227.6	1	732.27	162	" "	dsj1000.p10.v83	6940676.8	1	9752.77	93	" "
eil101.p10	236.1	434	101.88	436	" "	lin318.p40.v16	31156.8	107	950.29	219	" "	dsj1000.p20.v16	11010588.1	1	47.19	140	" "
eil101.p20	313.7	261	101.32	290	" "	lin318.p40.v33	32134.2	30	652.14	208	" "	dsj1000.p20.v33	10757083.9	1	1172.85	123	" "
eil101.p30	364.0	237	97.94	283	" "	lin318.p40.v50	32152.0	2	235.83	200	" "	dsj1000.p20.v50	10757457.2	1	6229.08	124	" "
eil101.p40	413.6	280	101.59	282	" "	lin318.p40.v66	32272.0	1	905.49	177	" "	dsj1000.p20.v66	9950239.4	1	4431.27	108	" "
eil101.p50	460.7	268	101.93	280	" "	lin318.p40.v83	30572.7	1	810.29	167	" "	dsj1000.p20.v83	10312122.1	1	1524.21	95	" "
ch150.p10.v16	2668.2	285	215.23	299	225	lin318.p50.v16	34119.5	222	1006.71	224	" "	dsj1000.p30.v16	12869179.9	1	7253.26	135	" "
ch150.p10.v33	2471.9	287	219.47	342	" "	lin318.p50.v33	33610.0	218	986.67	224	" "	dsj1000.p30.v33	12879954.8	1	3352.98	125	" "
ch150.p10.v50	2458.4	88	215.35	361	" "	lin318.p50.v50	32728.5	52	748.47	206	" "	dsj1000.p30.v50	12233534.5	1	9368.57	118	" "
ch150.p10.v66	2487.9	308	218.55	330	" "	lin318.p50.v66	34724.6	1	947.3	196	" "	dsj1000.p30.v66	12670807.3	1	7714.14	109	" "
ch150.p10.v83	2514.6	802	216.81	825	" "	lin318.p50.v83	33013.2	1	922.8	166	" "	dsj1000.p30.v83	12081075.5	1	1649.63	84	" "
ch150.p20.v16	3577.2	57	205.55	269	" "	lin318.p10	20075.7	14	909.41	240	" "	dsj1000.p40.v16	14267996.3	119	9886.1	133	" "
ch150.p20.v33	3570.6	102	163.03	268	" "	lin318.p20	26765.1	12	699.73	228	" "	dsj1000.p40.v33	14100084.6	16	9551.9	135	" "
ch150.p20.v50	3371.1	226	207.16	305	" "	lin318.p30	29958.2	1	900.57	226	" "	dsj1000.p40.v50	14243574.9	1	9623.66	119	" "
ch150.p20.v66	3144.1	113	213.42	349	" "	lin318.p40	32122.0	148	658.13	226	" "	dsj1000.p40.v66	14304389.9	1	1879.36	110	" "
ch150.p20.v83	3410.3	99	177.75	271	" "	lin318.p50	34338.2	231	1005.81	233	" "	dsj1000.p40.v83	13791984.3	1	8272.07	90	" "
ch150.p30.v16	4046.1	69	223.99	261	" "	att532.p10.v16	40632.8	3	2054.18	183	2830	dsj1000.p50.v16	15676078.7	134	9216.52	144	" "
ch150.p30.v33	4113.3	2	224.27	261	" "	att532.p10.v33	37971.3	1	2199.85	186	" "	dsj1000.p50.v33	15484736.5	124	9623.54	129	" "
ch150.p30.v50	3986.3	32	180.89	253	" "	att532.p10.v50	34184.7	1	1945.84	166	" "	dsj1000.p50.v50	15564876.7	44	5802.32	116	" "
ch150.p30.v66	3819.4	291	223.8	303	" "	att532.p10.v66	32732.6	1	1740.44	158	" "	dsj1000.p50.v66	14968905.1	29	8977.94	112	" "
ch150.p30.v83	4001.2	1	224.79	346	" "	att532.p10.v83	31207.0	1	937.23	143	" "	dsj1000.p50.v83	14671854.7	1	9401.83	95	" "
ch150.p40.v16	4590.7	264	220.7	269	" "	att532.p20.v16	47825.1	1	1456.4	176	" "	dsj1000.p10	8116079.6	40	2342.36	147	" "
ch150.p40.v33	4603.4	221	194.61	271	" "	att532.p20.v33	47879.5	1	1125.87	182	" "	dsj1000.p20	10925284.4	1	973.42	150	" "
ch150.p40.v50	4463.8	262	221.91	266	" "	att532.p20.v50	45897.9	1	651.84	175	" "	dsj1000.p30	12946617.0	1	46.87	145	" "
ch150.p40.v66	4419.9	200	224.58	249	" "	att532.p20.v66	45449.7	1	1941.89	151	" "	dsj1000.p40	14358797.0	142	9886.96	145	" "
ch150.p40.v83	4460.0	28	116.53	268	" "	att532.p20.v83	43596.9	1	1050.87	131	" "	dsj1000.p50	15373069.3	120	9731.58	145	" "

Table A.4: Detailed results of pACS-S+1-shift-S for PTSP instances with $0.1 \leq p \leq 0.5$.

pACS+1-shift-T

Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}	Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}	Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}
kroA100.p10.v16	9348.0	1	96.87	33	100	ch150.p50.v16	5422.4	28	195.63	33	" "	att532.p30.v16	57908.2	17	1544.22	32	" "
kroA100.p10.v33	8331.0	29	86.38	34	" "	ch150.p50.v33	5499.8	33	220.13	35	" "	att532.p30.v33	58734.3	27	2512.5	31	" "
kroA100.p10.v50	7327.5	32	82.44	40	" "	ch150.p50.v50	5156.8	28	194.21	34	" "	att532.p30.v50	56852.1	17	1546.67	32	" "
kroA100.p10.v66	9512.3	41	99.87	49	" "	ch150.p50.v66	5432.5	27	176.06	35	" "	att532.p30.v66	57661.7	12	936.64	36	" "
kroA100.p10.v83	9700.7	50	85.73	59	" "	ch150.p50.v83	4770.0	36	200.7	42	" "	att532.p30.v83	57620.2	32	2467.12	37	" "
kroA100.p20.v16	11394.6	30	91.63	34	" "	ch150.p10	2577.8	32	209.2	36	" "	att532.p40.v16	64715.6	20	1801.47	32	" "
kroA100.p20.v33	12058.6	27	82.04	34	" "	ch150.p20	3753.0	33	212.7	36	" "	att532.p40.v33	65357.6	30	2764.96	32	" "
kroA100.p20.v50	12133.9	33	98.06	35	" "	ch150.p30	4358.9	29	184.58	36	" "	att532.p40.v50	65774.5	27	2746.18	32	" "
kroA100.p20.v66	10381.0	36	86.66	43	" "	ch150.p40	4893.7	28	212.95	36	" "	att532.p40.v66	65977.7	28	2352.64	34	" "
kroA100.p20.v83	11419.5	54	96.58	59	" "	ch150.p50	5447.5	24	152.6	36	" "	att532.p40.v83	64480.5	32	2683.3	34	" "
kroA100.p30.v16	14599.6	33	99.84	34	" "	d198.p10.v16	8389.9	22	256.15	34	392	att532.p50.v16	72994.9	18	1630.35	32	" "
kroA100.p30.v33	13520.8	23	98.93	33	" "	d198.p10.v33	8030.4	32	352.68	36	" "	att532.p50.v33	69544.9	1	197.06	32	" "
kroA100.p30.v50	14613.7	33	99.88	34	" "	d198.p10.v50	7801.8	29	265	42	" "	att532.p50.v50	71411.7	21	1971.07	30	" "
kroA100.p30.v66	13408.7	29	97.46	34	" "	d198.p10.v66	6743.7	40	376.71	43	" "	att532.p50.v66	71518.3	29	2820.5	32	" "
kroA100.p30.v83	13277.4	42	99.96	43	" "	d198.p10.v83	6929.2	13	73.47	63	" "	att532.p50.v83	73585.1	29	2568.87	33	" "
kroA100.p40.v16	15998.5	25	95.88	34	" "	d198.p20.v16	9311.6	27	315.66	34	" "	att532.p10	37364.4	28	2469.98	33	" "
kroA100.p40.v33	15242.8	31	92.08	34	" "	d198.p20.v33	10371.0	21	244.63	34	" "	att532.p20	49824.7	17	1484.69	32	" "
kroA100.p40.v50	17053.9	30	98.34	33	" "	d198.p20.v50	8576.5	35	391.09	36	" "	att532.p30	57812.4	27	2466.91	32	" "
kroA100.p40.v66	16065.1	12	99.03	33	" "	d198.p20.v66	9424.4	28	299.07	41	" "	att532.p40	66226.2	21	1837.09	32	" "
kroA100.p40.v83	14270.9	46	96.87	53	" "	d198.p20.v83	9863.9	56	387.36	57	" "	att532.p50	71613.6	29	2588.05	32	" "
kroA100.p50.v16	18201.7	30	92.04	34	" "	d198.p30.v16	11366.1	29	345.58	34	" "	rat783.p10.v16	3525.9	21	4231.01	30	6131
kroA100.p50.v33	17318.5	26	93.84	33	" "	d198.p30.v33	10735.2	32	384.79	34	" "	rat783.p10.v33	3493.5	29	5582.51	32	" "
kroA100.p50.v50	17037.5	32	96.46	34	" "	d198.p30.v50	11323.1	27	312.45	34	" "	rat783.p10.v50	3216.1	33	5342.81	39	" "
kroA100.p50.v66	16791.1	21	94.86	34	" "	d198.p30.v66	12744.4	31	343.01	36	" "	rat783.p10.v66	3428.2	22	2877.12	44	" "
kroA100.p50.v83	15845.6	28	74.81	43	" "	d198.p30.v83	11015.8	26	278.55	37	" "	rat783.p10.v83	3029.5	54	5998.12	56	" "
kroA100.p10	9229.9	25	86.89	35	" "	d198.p40.v16	12122.4	17	366.73	34	" "	rat783.p20.v16	5024.5	26	5297.28	30	" "
kroA100.p20	12115.8	33	89.57	37	" "	d198.p40.v33	13526.4	28	332.59	34	" "	rat783.p20.v33	4874.4	18	5734.15	31	" "
kroA100.p30	14042.3	34	97.85	35	" "	d198.p40.v50	11635.0	2	15.83	34	" "	rat783.p20.v50	4944.0	25	4577.05	33	" "
kroA100.p40	15653.9	29	93.52	35	" "	d198.p40.v66	13777.8	24	265.42	36	" "	rat783.p20.v66	4556.9	36	6046.02	38	" "
kroA100.p50	16839.5	34	98.3	35	" "	d198.p40.v83	12050.0	37	389.63	38	" "	rat783.p20.v83	5305.1	37	4723.23	49	" "
eil101.p10.v16	196.1	22	64.67	35	102	d198.p50.v16	13793.4	27	322.03	34	" "	rat783.p30.v16	5921.3	28	5861.3	30	" "
eil101.p10.v33	179.3	31	87.44	37	" "	d198.p50.v33	13281.5	28	328.1	34	" "	rat783.p30.v33	5942.7	23	4980.95	31	" "
eil101.p10.v50	150.4	39	87.42	46	" "	d198.p50.v50	12361.1	30	353.66	34	" "	rat783.p30.v50	5963.9	30	6101.2	31	" "
eil101.p10.v66	181.9	48	88.7	56	" "	d198.p50.v66	12830.2	6	63.19	34	" "	rat783.p30.v66	5939.1	25	5004.66	31	" "
eil101.p10.v83	220.9	45	85.23	54	" "	d198.p50.v83	14009.9	27	318.41	35	" "	rat783.p30.v83	5796.8	37	5713.86	40	" "
eil101.p20.v16	297.4	30	94.03	34	" "	d198.p10	7747.8	32	357.81	36	" "	rat783.p40.v16	6730.1	27	5515.75	30	" "
eil101.p20.v33	318.2	33	100.76	34	" "	d198.p20	10031.1	19	210.6	35	" "	rat783.p40.v33	6846.5	16	3175.08	30	" "
eil101.p20.v50	296.4	30	95.55	37	" "	d198.p30	11157.3	22	239.4	36	" "	rat783.p40.v50	6842.6	4	767.31	29	" "
eil101.p20.v66	289.6	30	100.05	44	" "	d198.p40	12698.4	16	173.4	36	" "	rat783.p40.v66	6862.5	25	4955.71	31	" "
eil101.p20.v83	281.3	21	47.14	48	" "	d198.p50	13261.4	33	370.36	35	" "	rat783.p40.v83	6230.8	30	4807.95	39	" "
eil101.p30.v16	366.3	32	97.43	34	" "	lin318.p10.v16	19371.4	30	948.77	33	1011	rat783.p50.v16	7479.0	28	5755.2	30	" "
eil101.p30.v33	378.0	33	100.03	34	" "	lin318.p10.v33	19469.2	28	1000.68	33	" "	rat783.p50.v33	7377.6	28	6007.38	29	" "
eil101.p30.v50	382.1	32	95.19	35	" "	lin318.p10.v50	18096.6	34	994.63	36	" "	rat783.p50.v50	7616.4	15	2965.04	30	" "
eil101.p30.v66	385.4	30	84.14	37	" "	lin318.p10.v66	18686.4	24	502.18	48	" "	rat783.p50.v66	7471.4	29	5795.24	31	" "
eil101.p30.v83	394.0	21	85.26	48	" "	lin318.p10.v83	19078.3	44	822.52	53	" "	rat783.p50.v83	7421.3	7	1079.08	34	" "
eil101.p40.v16	433.8	33	99.69	35	" "	lin318.p20.v16	25663.2	29	914.4	33	" "	rat783.p10	3488.3	30	5913.63	32	" "
eil101.p40.v33	459.2	25	76	34	" "	lin318.p20.v33	25551.1	30	955.51	33	" "	rat783.p20	4886.7	16	3026.56	32	" "
eil101.p40.v50	432.0	30	91.43	35	" "	lin318.p20.v50	24910.4	28	837.48	34	" "	rat783.p30	5912.7	22	4373.63	32	" "
eil101.p40.v66	473.5	32	94.8	35	" "	lin318.p20.v66	23651.3	34	832.32	41	" "	rat783.p40	6736.3	8	1415.85	32	" "
eil101.p40.v83	401.0	36	87.61	42	" "	lin318.p20.v83	22667.0	50	963.43	53	" "	rat783.p50	7427.4	31	6127.99	32	" "
eil101.p50.v16	491.4	25	76.03	34	" "	lin318.p30.v16	28740.3	29	914.08	33	" "	dsj1000.p10.v16	8344886.8	25	9015.93	28	10000
eil101.p50.v33	506.0	33	101.34	34	" "	lin318.p30.v33	29180.9	29	967	33	" "	dsj1000.p10.v33	7895146.6	25	8634.33	29	" "
eil101.p50.v50	499.2	30	91.31	34	" "	lin318.p30.v50	29372.8	32	1010.71	33	" "	dsj1000.p10.v50	7939788.7	31	8747.46	36	" "
eil101.p50.v66	467.6	29	74.87	39	" "	lin318.p30.v66	29097.7	38	1002.2	39	" "	dsj1000.p10.v66	8496195.3	31	7103.56	42	" "
eil101.p50.v83	445.9	22	100.63	39	" "	lin318.p30.v83	29724.9	36	885.98	42	" "	dsj1000.p10.v83	7858166.8	6	1016.28	52	" "
eil101.p10	209.8	34	100.91	35	" "	lin318.p40.v16	32505.6	29	926.83	33	" "	dsj1000.p20.v16	11027340.2	21	7453.91	29	" "
eil101.p20	308.1	30	95.91	35	" "	lin318.p40.v33	33523.0	11	332.78	32	" "	dsj1000.p20.v33	10947425.7	25	8846.33	29	" "
eil101.p30	386.4	35	100.79	36	" "	lin318.p40.v50	32605.8	31	968.79	33	" "	dsj1000.p20.v50	10968878.7	10	3234.45	30	" "
eil101.p40	441.9	28	80.95	36	" "	lin318.p40.v66	33333.6	32	951.06	35	" "	dsj1000.p20.v66	10518728.9	33	9867.85	35	" "
eil101.p50	491.7	27	80.5	35	" "	lin318.p40.v83	33625.5	36	974.37	38	" "	dsj1000.p20.v83	10997852.7	34	8468.7	44	" "
ch150.p10.v16	2628.2	26	176.33	33	225	lin318.p50.v16	35273.7	31	978.35	33	" "	dsj1000.p30.v16	13014260.7	14	4789.86	29	" "
ch150.p10.v33	2363.6	26	162.29	36	" "	lin318.p50.v33	34698.5	30	946.85	33	" "	dsj1000.p30.v33	13159190.9	7	7242.54	28	" "
ch150.p10.v50	2605.1	40	210.51	43	" "	lin318.p50.v50	34667.0	30	938.31	33	" "	dsj1000.p30.v50	12475469.4	1	189.59	29	" "
ch150.p10.v66	2509.2	32	133.26	56	" "	lin318.p50.v66	35137.2	32	1010.95	33	" "	dsj1000.p30.v66	13360667.7	27	8907.79	31	" "
ch150.p10.v83	2511.8	31	128.1	54	" "	lin318.p50.v83	35156.1	37	1000.48	38	" "	dsj1000.p30.v83	12498224.5	32	8996.26	36	" "
ch150.p20.v16	3644.4	31	211.92	33	" "	lin318.p10	18714.7	23	700.97	34	" "	dsj1000.p40.v16	14626192.1	25	9290.11	28	" "
ch150.p20.v33	3725.9	11	72.9	34	" "	lin318.p20	24880.3	27	810.61	34	" "	dsj1000.p40.v33	14619775.6	25	8999.73	28	" "
ch150.p20.v50	3722.8	30	187.92	37	" "	lin318.p30	29583.3	31	920.06	34	" "	dsj1000.p40.v50	14600198.7	20	7610.9	27	" "
ch150.p20.v66	3420.2	34	209.07	37	" "	lin318.p40	32906.9	27	823.45	34	" "	dsj1000.p40.v66	14801227.0	23	8223.57	29	" "
ch150.p20.v83	3543.7	29	120.52	53	" "	lin318.p50	35558.5	30	899.76	34	" "	dsj1000.p40.v83	14310237.1	30	8017.7	38	" "
ch150.p30.v16	4198.0	22	151.43	34	" "	att532.p10.v16	39690.6	30	2730.77	32	2830	dsj1000.p50.v16	15911032.9	22	8298.73	27	" "
ch150.p30.v33	4276.5	33	221.86	34	" "	att532.p10.v33	38117.1	37	2684.31	40	" "	dsj1000.p50.v33	15829449.3	6	1980.95	27	" "
ch150.p30.v50	4208.6	29	195.82	34	" "	att532.p10.v50	36298.6	45	2640.98	49	" "	dsj1000.p50.v50	15995734.4	10	3454.64	28	" "
ch150.p30.v66	4274.9	20	123.56	39	" "	att532.p10.v66	34110.9	42	2753.45	44	" "	dsj1000.p50.v66	15915362.2	27	9645.22	29	" "
ch150.p30.v83	4151.0	35	209.3	38	" "	att532.p10.v83	35996.3	32	1552.58	58	" "	dsj1000.p50.v83	15234347.6	31	9179.5	34	" "
ch150.p40.v16	4879.9	33	223.38	34	" "	att532.p20.v16	49769.3	15	1319.57	32	" "	dsj1000.p10	8141856.8	27	9332.35	30	" "
ch150.p40.v33	5126.4	31	208.86	34	" "	att532.p20.v33	48588.3	6	1512.41	31	" "	dsj1000.p20	10928532.6	14	4712.81	30	" "
ch150.p40.v50	4902.1	25	174.31	34	" "	att532.p20.v50	46622.8	32	2781.3	33	" "	dsj1000.p30	12904321.0	24	8081.83	30	" "
ch150.p40.v66	4805.9	33	210.88	36	" "	att532.p20.v66	48480.0	38	2799.28	39	" "	dsj1000.p40	14596440.1	27	9238.32	30	" "
ch150.p40.v83	4752.7	34	187.52	41	" "	att532.p20.v83	49350.4	47	2763.82	49	" "	dsj1000.p50	15844384.8	3	905.79	30	" "

Table A.5: Detailed results of pACS+1-shift-T for PTSP instances with $0.1 \leq p \leq 0.5$.

pACS+1-shift-P

Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}	Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}	Instance name	$best$	k_b	t_b	k_{tot}	t_{tot}
kroA100.p10.v16	9231.28	40	99.37	41	100	ch150.p50.v16	5257.95	32	218.23	33	"	att532.p30.v16	57234.84	15	1073.41	40	"
kroA100.p10.v33	8311.57	47	92.16	52	"	ch150.p50.v33	5364.55	41	217.65	43	"	att532.p30.v33	57562.08	41	2802.9	44	"
kroA100.p10.v50	7328.27	40	92.25	44	"	ch150.p50.v50	5127.69	51	222.38	53	"	att532.p30.v50	56744.81	27	1753.15	44	"
kroA100.p10.v66	9529.83	47	99.65	48	"	ch150.p50.v66	5440.89	27	115.16	52	"	att532.p30.v66	57350.33	50	2820.94	51	"
kroA100.p10.v83	9701.82	58	87.36	66	"	ch150.p50.v83	4420.91	52	216.69	55	"	att532.p30.v83	58715.05	33	1845.79	52	"
kroA100.p20.v16	11717.29	41	93.12	45	"	ch150.p10	2540.74	33	215.82	35	"	att532.p40.v16	64138.12	25	2394.97	30	"
kroA100.p20.v33	12553.35	45	88.81	51	"	ch150.p20	3446.41	35	224.83	36	"	att532.p40.v33	65607.63	41	2673.8	44	"
kroA100.p20.v50	11984.50	44	96.79	46	"	ch150.p30	4110.99	31	196.56	37	"	att532.p40.v50	68409.80	44	2555.34	50	"
kroA100.p20.v66	10415.38	41	78.85	53	"	ch150.p40	4629.25	28	216.36	35	"	att532.p40.v66	65621.06	49	2801.86	50	"
kroA100.p20.v83	11509.82	39	84.41	47	"	ch150.p50	5099.03	36	224.89	37	"	att532.p40.v83	66303.11	25	1465.78	49	"
kroA100.p30.v16	14345.75	41	99.06	42	"	d198.p10.v16	8268.71	42	390.66	43	392	att532.p50.v16	72897.09	28	2800.83	31	"
kroA100.p30.v33	13033.73	44	96.3	47	"	d198.p10.v33	8145.76	15	111.91	52	"	att532.p50.v33	70223.34	41	2782.62	42	"
kroA100.p30.v50	14552.30	47	99.62	48	"	d198.p10.v50	7600.31	46	378.69	48	"	att532.p50.v50	70726.72	40	2523.42	45	"
kroA100.p30.v66	13482.30	46	83.37	56	"	d198.p10.v66	6634.63	42	358.09	47	"	att532.p50.v66	72258.37	39	2151.36	52	"
kroA100.p30.v83	12595.77	32	61.26	54	"	d198.p10.v83	6647.58	17	168.29	42	"	att532.p50.v83	73392.64	43	2487.39	50	"
kroA100.p40.v16	15624.58	37	96.01	40	"	d198.p20.v16	9336.76	37	344.32	43	"	att532.p10	36007.74	1	2788.38	27	"
kroA100.p40.v33	15318.28	38	91.8	43	"	d198.p20.v33	10359.02	18	135.93	52	"	att532.p20	48548.83	26	2627.09	29	"
kroA100.p40.v50	17539.00	40	84.96	48	"	d198.p20.v50	8569.92	48	359.74	53	"	att532.p30	57075.65	28	2767.84	32	"
kroA100.p40.v66	15078.76	20	38.66	52	"	d198.p20.v66	9706.05	43	318.31	54	"	att532.p40	64432.50	24	2114.99	34	"
kroA100.p40.v83	14438.70	45	83.65	55	"	d198.p20.v83	9830.37	42	338.82	49	"	att532.p50	70544.94	21	2822.07	30	"
kroA100.p50.v16	16882.97	28	93.75	31	"	d198.p30.v16	11701.53	10	133.01	28	"	rat783.p10.v16	3524.52	22	2790.46	48	6131
kroA100.p50.v33	17470.58	20	88.65	23	"	d198.p30.v33	10714.47	48	388.99	49	"	rat783.p10.v33	3499.64	45	5529.84	50	"
kroA100.p50.v50	17990.17	34	75.38	46	"	d198.p30.v50	11414.01	50	386.73	51	"	rat783.p10.v50	3217.89	46	5962.09	48	"
kroA100.p50.v66	16547.76	51	96.61	53	"	d198.p30.v66	12755.91	48	382.89	50	"	rat783.p10.v66	3443.11	40	5976.19	42	"
kroA100.p50.v83	15846.55	11	19.43	58	"	d198.p30.v83	10989.88	16	113.56	53	"	rat783.p10.v83	2961.02	36	5151.16	44	"
kroA100.p10	9084.80	36	97.84	37	"	d198.p40.v16	12538.54	32	378.7	34	"	rat783.p20.v16	5021.14	35	4712.38	46	"
kroA100.p20	11778.00	28	89.83	40	"	d198.p40.v33	13259.88	38	328.9	46	"	rat783.p20.v33	4879.48	42	5383.5	49	"
kroA100.p30	13795.36	38	97.68	40	"	d198.p40.v50	11088.61	44	319.83	54	"	rat783.p20.v50	4944.03	45	5494.68	51	"
kroA100.p40	15381.14	1	73.77	29	"	d198.p40.v66	13317.83	50	386.57	51	"	rat783.p20.v66	4543.35	49	6031.97	51	"
kroA100.p50	16681.61	22	97	31	"	d198.p40.v83	11909.48	21	142.42	56	"	rat783.p20.v83	5278.60	39	5343.46	46	"
eil101.p10.v16	194.51	31	70.88	45	102	d198.p50.v16	13768.76	32	340.35	37	"	rat783.p30.v16	5916.00	28	4506.42	38	"
eil101.p10.v33	180.15	29	61.96	49	"	d198.p50.v33	13747.73	39	347.13	44	"	rat783.p30.v33	5925.74	45	6081.33	46	"
eil101.p10.v50	151.86	30	91.38	48	"	d198.p50.v50	12375.01	48	383.12	50	"	rat783.p30.v50	5969.87	38	5036.68	47	"
eil101.p10.v66	181.71	47	100.49	49	"	d198.p50.v66	12385.83	42	318.32	52	"	rat783.p30.v66	5955.31	46	6025.3	47	"
eil101.p10.v83	220.36	45	101.37	46	"	d198.p50.v83	13849.37	54	390.69	55	"	rat783.p30.v83	5842.49	33	4288.22	47	"
eil101.p20.v16	296.69	43	89.85	49	"	d198.p10	7650.94	10	383.74	35	"	rat783.p40.v16	6683.81	23	6082.74	30	"
eil101.p20.v33	319.11	47	98.25	50	"	d198.p20	9919.90	5	372.93	34	"	rat783.p40.v33	6846.09	4	687.02	41	"
eil101.p20.v50	298.27	38	68.32	56	"	d198.p30	11117.27	29	312.89	38	"	rat783.p40.v50	6843.67	42	5622.41	46	"
eil101.p20.v66	285.47	49	100.19	51	"	d198.p40	12187.15	3	25.67	38	"	rat783.p40.v66	6880.52	39	4842.51	50	"
eil101.p20.v83	308.84	23	54.41	47	"	d198.p50	13502.58	31	381.25	39	"	rat783.p40.v83	6239.47	8	977.54	51	"
eil101.p30.v16	378.51	46	99.83	48	"	lin318.p10.v16	19629.48	45	911.89	51	1011	rat783.p50.v16	7337.81	23	5323.79	27	"
eil101.p30.v33	385.33	49	91.87	55	"	lin318.p10.v33	19357.58	39	795.46	51	"	rat783.p50.v33	7378.40	40	5924.48	42	"
eil101.p30.v50	385.72	14	27.12	52	"	lin318.p10.v50	18532.24	49	966.67	52	"	rat783.p50.v50	7602.76	32	4686.82	43	"
eil101.p30.v66	390.74	48	94.87	52	"	lin318.p10.v66	18545.92	13	303.72	43	"	rat783.p50.v66	7453.93	46	5903.61	48	"
eil101.p30.v83	383.55	37	79.27	49	"	lin318.p10.v83	20357.57	24	527.25	45	"	rat783.p50.v83	7416.97	18	2208.26	49	"
eil101.p40.v16	433.43	26	66	39	"	lin318.p20.v16	26162.23	39	897.05	46	"	rat783.p10	3440.87	24	5968.88	26	"
eil101.p40.v33	469.71	43	95.87	46	"	lin318.p20.v33	25489.35	48	1004.72	49	"	rat783.p20	4857.52	7	5849.77	28	"
eil101.p40.v50	424.77	45	95.15	49	"	lin318.p20.v50	25203.28	51	1003.2	52	"	rat783.p30	5893.63	26	5924.7	27	"
eil101.p40.v66	475.40	9	16.58	55	"	lin318.p20.v66	23982.01	52	968.84	55	"	rat783.p40	6666.42	1	5705.13	31	"
eil101.p40.v83	392.49	44	86.03	53	"	lin318.p20.v83	22659.58	36	767.78	48	"	rat783.p50	7346.53	29	5951.31	32	"
eil101.p50.v16	493.19	22	61.43	37	"	lin318.p30.v16	28519.06	12	945.23	39	"	dsj1000.p10.v16	8348796.60	29	7033.6	41	10000
eil101.p50.v33	509.28	37	75.09	51	"	lin318.p30.v33	30516.39	46	984.4	48	"	dsj1000.p10.v33	7697726.28	35	7513.74	47	"
eil101.p50.v50	505.83	35	82.97	44	"	lin318.p30.v50	30317.57	25	484.17	53	"	dsj1000.p10.v50	7986595.96	43	9229.47	47	"
eil101.p50.v66	468.49	51	101.49	52	"	lin318.p30.v66	30239.58	36	663.55	55	"	dsj1000.p10.v66	8321199.24	34	8414.45	40	"
eil101.p50.v83	465.03	22	39.18	57	"	lin318.p30.v83	28867.86	41	782.67	54	"	dsj1000.p10.v83	7741932.44	22	5562.1	40	"
eil101.p10	201.61	18	96.67	30	"	lin318.p40.v16	32625.38	13	958.13	33	"	dsj1000.p20.v16	11002618.72	36	9467.29	38	"
eil101.p20	288.27	22	96.24	35	"	lin318.p40.v33	33232.49	43	1008.6	44	"	dsj1000.p20.v33	10775932.04	36	8555.22	42	"
eil101.p30	355.95	29	97.18	36	"	lin318.p40.v50	33408.37	25	481.85	53	"	dsj1000.p20.v50	11099510.72	44	9835.26	45	"
eil101.p40	414.84	29	90.32	34	"	lin318.p40.v66	33937.32	43	966.23	54	"	dsj1000.p20.v66	10464092.81	26	5693.44	46	"
eil101.p50	465.96	30	90.33	34	"	lin318.p40.v83	34088.45	23	443.12	54	"	dsj1000.p20.v83	11016257.59	38	9610.05	41	"
ch150.p10.v16	2641.00	43	211.75	46	225	lin318.p50.v16	35388.85	28	930.67	31	"	dsj1000.p30.v16	13018989.62	13	3906.66	32	"
ch150.p10.v33	2358.79	51	211.74	54	"	lin318.p50.v33	34950.54	34	912.65	38	"	dsj1000.p30.v33	13237395.32	36	8121.27	45	"
ch150.p10.v50	2536.87	10	44.35	50	"	lin318.p50.v50	34501.05	40	874.58	47	"	dsj1000.p30.v50	12474427.16	35	7553.77	47	"
ch150.p10.v66	2499.40	42	210.2	45	"	lin318.p50.v66	35841.49	48	957.67	52	"	dsj1000.p30.v66	13363286.41	43	9243.32	47	"
ch150.p10.v83	2584.48	39	219.89	41	"	lin318.p50.v83	35602.99	51	969.28	54	"	dsj1000.p30.v83	12465217.77	39	8306.3	47	"
ch150.p20.v16	3739.19	48	221.58	50	"	lin318.p10	18012.75	13	883.33	28	"	dsj1000.p40.v16	14698262.25	23	7675.73	32	"
ch150.p20.v33	3612.21	38	163.47	53	"	lin318.p20	25393.27	13	747.61	31	"	dsj1000.p40.v33	14781265.97	38	9782.92	40	"
ch150.p20.v50	3652.77	49	202.77	55	"	lin318.p30	29250.48	26	957.45	34	"	dsj1000.p40.v50	14600198.67	19	4365.39	44	"
ch150.p20.v66	3491.84	40	168.43	53	"	lin318.p40	32798.12	33	991.58	35	"	dsj1000.p40.v66	15124600.00	35	8343.21	42	"
ch150.p20.v83	3581.89	44	211.21	47	"	lin318.p50	35010.69	29	990.85	34	"	dsj1000.p40.v83	14341220.65	34	7251.32	47	"
ch150.p30.v16	4172.97	36	209.8	39	"	att532.p10.v16	39506.27	45	2544.86	51	2830	dsj1000.p50.v16	15842221.27	5	9036.06	31	"
ch150.p30.v33	4235.27	50	219.55	52	"	att532.p10.v33	37525.77	49	2724.85	52	"	dsj1000.p50.v33	15829449.32	14	3927.15	33	"
ch150.p30.v50	4206.13	48	219.12	53	"	att532.p10.v50	35792.71	42	2373.37	50	"	dsj1000.p50.v50	15995734.38	40	9200.55	44	"
ch150.p30.v66	4069.44	54	219.78	56	"	att532.p10.v66	35366.58	25	1577.11	45	"	dsj1000.p50.v66	15917561.02	27	5893.02	46	"
ch150.p30.v83	4137.79	46	187.63	56	"	att532.p10.v83	35489.79	26	1704.97	43	"	dsj1000.p50.v83	15223680.09	44	9446.47	47	"
ch150.p40.v16	4654.73	30	212.05	38	"	att532.p20.v16	48497.67	41	2523.11	47	"	dsj1000.p10	8170111.35	1	8630.59	23	"
ch150.p40.v33	5108.15	37	195.24	43	"	att532.p20.v33	48736.35	1	122.23	48	"	dsj1000.p20	10826795.74	1	9148.94	28	"
ch150.p40.v50	5009.68	46	211.45	50	"	att532.p20.v50	47561.67	38	2058.16	52	"	dsj1000.p30	12901577.35	25	8704.97	30	"
ch150.p40.v66	4939.47	38	164.5	52	"	att532.p20.v66	48693.10	40	2399.24	48	"	dsj1000.p40	14438800.04	12	9900.4	30	"
ch150.p40.v83	4759.95	41	174.58	54	"	att532.p20.v83	48858.97	44	2670.55	47	"	dsj1000.p50	15809154.06	1	9528.45	30	"

Table A.6: Detailed results of pACS+1-shift-P for PTSP instances with $0.1 \leq p \leq 0.5$.

Appendix B

Hybrid Metaheuristics for the Vehicle Routing Problem with Stochastic Demands

This section analyzes the performance of metaheuristics on the vehicle routing problem with stochastic demands (VRPSD). The problem is known to have a computationally demanding objective function, and for this reason the optimization of large instances could be an infeasible task. Fast approximations of the objective function are therefore appealing because they would allow for an extended exploration of the search space. We explore the *hybridization* of the metaheuristic by means of two objective functions which are approximate measures of the exact solution quality. Particularly helpful for some metaheuristics is the objective function derived from the TSP, a closely related problem. In the light of this observation, we analyze possible extensions of the metaheuristics which take the hybridized solution approach VRPSD-TSP even further and report about experimental results on different types of instances. We show that, for the instances tested, two hybridized versions of iterated local search and evolutionary algorithm attain better solutions than state-of-the-art algorithms.

B.1 Introduction

Vehicle routing problems (VRPs) concern the transport of items between depots and customers by means of a fleet of vehicles. Solving a VRP means finding the best set of routes servicing all customers and respecting the operational constraints, such as vehicles capacity, time windows, driver's maximum working time. VRPs are a key issue in supply-chain and distribution systems today, and they are becoming increasingly complex. For this reason there is an ever increasing interest in routing models that are dynamic, stochastic, rich of constraints, and thus have more and more complex objective functions.

Models that focus particularly on the *stochasticity* of information are mainly known in the literature as Stochastic VRPs (SVRPs), and the problem we are addressing in

this section belongs to this class. In SVRPs elements of the problem such as the set of customers visited, the customers demands, or the travel times, are modeled as stochastic variables with known probability distributions, and the objective function is usually the expected cost of the planned routes. Note, however, that in SVRPs one needs to specify not only the concept of 'planned routes', but also the way planned routes are to be modified in response to the realization of the stochastic information.

A common feature of SVRPs is that they all have at least one deterministic counterpart, which is the VRP that one obtains by considering zero-variance probability distributions for the stochastic elements of the problem. SVRPs are thus NP-hard problems, like most VRPs. An important point that increases the difficulty of SVRPs is that they have an objective function (expected cost) which is much more computationally expensive than their deterministic counterparts. For this reason, a key issue in solving SVRPs by heuristics and metaheuristics is the use of fast and effective objective function approximations that may accelerate the search process. Due to the analogy between stochastic and deterministic VRPs, a reasonable choice for the objective function approximation of a given SVRP is the objective of the corresponding deterministic problems.

In this section, we investigate the use of objective function approximations derived from deterministic problems in the context of the VRPSD. This is an NP-hard problem, and despite the fact that it has a quite simple formulation, it arises in practice in many real world situations. One example is garbage collection, where it is indeed impossible to know a priori how much garbage has to be collected at each place. Another example where the demand is uncertain is the delivery of petrol to petrol stations. In fact, when a customer issues the order it is still unknown how much he will sell in the time between the order and the delivery.

In the VRPSD, a vehicle of finite capacity is leaving from a depot with full load, and has to serve a set of customers whose exact demand is only known on arrival at the each customer location. A planned route in this context is very simple: a tour starting from the depot and visiting all customers exactly once; this is also called an a priori tour, and it will be addressed as such in the remainder of the section. The a priori tour is a sort of skeleton that fixes the order in which customers will be served, but the actual route the vehicle would travel would include return trips to the depot for replenishments when needed. The points at which return trips are performed are, in general, stochastic. The objective function to be minimized is the expected cost of the a priori tour.

Due to the nature of the a priori tour, a feasible solution for the VRPSD may also be seen as a feasible solution for a traveling salesman problem (TSP) on the set of customers (depot included). Moreover, if the vehicle has infinite capacity, the consequent VRPSD is, in fact, a TSP. Due to these analogies, a natural approximation of the VRPSD objective function is the length of the a priori tour.

In this section we consider basic implementations of five metaheuristics: simulated annealing [129], tabu search [91], iterated local search [137], ant colony optimization [72] and evolutionary algorithms [11]. Our main goal is to test the impact on metaheuristics of interleaving the exact VRPSD objective function with the a priori tour

length as an approximation of it. This mixture changes the *search landscape* during the search for good quality solutions and can be seen as an innovative type of hybridization of a metaheuristic's search process that has not been yet explored in the literature. In particular, we investigate two types of hybridization: first, we consider a local search algorithm (OrOpt) for which a quite good approximation for the exact VRPSD objective function is available, and we compare metaheuristics using this underlined local search by applying both VRPSD approximation and TSP approximation. Second, we further exploit the TSP analogy, by choosing the 3-opt local search operator, which is very good for the TSP, but for which there is no immediate VRPSD approximation.

The remainder of this section is organized as follows. In Section B.2 we give the formal description of the VRPSD, we describe in detail the objective function, the state of the art about the VRPSD, and the relevant aspects of generating a benchmark of instances for this problem, taking into account the existing literature. Section B.3 describes at high level the metaheuristics and the other algorithms analyzed. Section B.4 reports details about tested instances, parameters used for the metaheuristics, computation times allowed. Sections B.5 and B.6 describe the computational experiments respectively on the first type of hybridization (the use of the TSP objective function in OrOpt) and on the second type of hybridization (the use of the 3-opt local search with the TSP objective function). Section B.7 summarizes the main conclusions that can be drawn from the experimental results.

B.2 The Vehicle Routing Problem with Stochastic Demands

The VRPSD is defined on a complete graph $G = (V, A, D)$, where $V = \{0, 1, ..., n\}$ is a set of nodes (customers) with node 0 denoting the depot, $A = \{(i, j) : i, j \in V, i \neq j\}$ is the set of arcs joining the nodes, and $D = \{d_{ij} : i, j \in V, i \neq j\}$ are the travel costs (distances) between nodes. The cost matrix D is symmetric and satisfies the triangular inequality. One vehicle with capacity Q has to deliver goods to the customers according to their demands, minimizing the total expected distance traveled, and given that the following assumptions are made. Customers' demands are stochastic variables ξ_i, $i = 1, ..., n$ independently distributed with known distributions. The actual demand of each customer is only known when the vehicle arrives at the customer location. It is also assumed that ξ_i does not exceed the vehicle's capacity Q, and follows a discrete probability distribution $p_{ik} = \text{Prob}(\xi_i = k)$, $k = 0, 1, 2, ..., K \leq Q$. A feasible solution to the VRPSD is a permutation of the customers $s = (s(1), s(2), \ldots, s(n))$ starting at the depot (that is, $s(1) = 0$), and it is called an a priori tour. The vehicle visits the customers in the order given by the a priori tour, and it has to choose, according to the actual customer's demand, whether to proceed to the next customer or to go to depot for restocking. Sometimes the choice of restocking is the best one, even if the vehicle is not empty, or if its capacity is bigger than the expected demand of the next scheduled customer; this action is called 'preventive restocking'. The goal of preventive restocking is to avoid the risk of having a vehicle without enough load to serve a customer and

thus having to perform a back-and-forth trip to the depot for completing the delivery at the customer.

The expected distance traveled by the vehicle (that is, the objective function), is computed as follows. Let $s = (0, 1, \ldots, n)$ be an a priori tour. 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. With this notation, the expected cost of the a priori tour is $f_0(Q)$. If L_j represents the set of all possible loads that a vehicle can have after service completion at customer j, then, $f_j(q)$ for $q \in L_j$ satisfies

$$f_j(q) = \text{Minimum}\{f_j^p(q), f_j^r(q)\}, \tag{B.1}$$

where

$$\begin{aligned} f_j^p(q) = d_{j,j+1} &+ \sum_{k:k\le q} f_{j+1}(q-k)p_{j+1,k} \\ &+ \sum_{k:k>q} [2d_{j+1,0} + f_{j+1}(q+Q-k)]p_{j+1,k}, \end{aligned} \tag{B.2}$$

$$f_j^r(q) = d_{j,0} + d_{0,j+1} + \sum_{k=1}^{K} f_{j+1}(Q-k)p_{j+1,k}, \tag{B.3}$$

with the boundary condition $f_n(q) = d_{n,0},\ q \in L_n$. In (B.2-B.3), $f_j^p(q)$ is the expected cost corresponding to the choice of proceeding directly to the next customer, while $f_j^r(q)$ is the expected cost in case preventive restocking is chosen. As shown by Yang et al. in [183], the optimal choice is of threshold type: given the a priori tour, for each customer j there is a load threshold h_j such that, if the residual load after serving j is greater than or equal to h_j, then it is better to proceed to the next planned customer, otherwise it is better to go back to the depot for preventive restocking. This property of the VRPSD is illustrated by Figure B.1. The computation of $f_0(Q)$ 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. Algorithm 20 is an implementation of the recursion (B.2-B.3) for the computation of $f_0(Q)$ and of the thresholds.

According to the above definition, the VRSPD is a single-vehicle routing problem. Note that there would be no advantage in considering a multiple-vehicle problem by allowing multiple a priori tours, since, as proved by Yang et al. [183], the optimal solution is always a single tour.

The literature about VRPSD and SVRPs in general is quite rich. Formulations of SVRPs include the Traveling Salesman Problem with Stochastic Customers (TSPSC), the Traveling Salesman Problem with Stochastic Travel Times (TSPST), the Vehicle Routing Problem with Stochastic Customers (VRPSC), the Vehicle Routing Problem with Stochastic Customers and Demands (VRPSCD). For a survey on the early approaches to these problems, see [87] and [29]; for a more recent survey, especially on mathematical programming approaches used for SVRPs, see [128]. In the following we summarize the main contributions to solve VRPSD and similar problems, relevant to

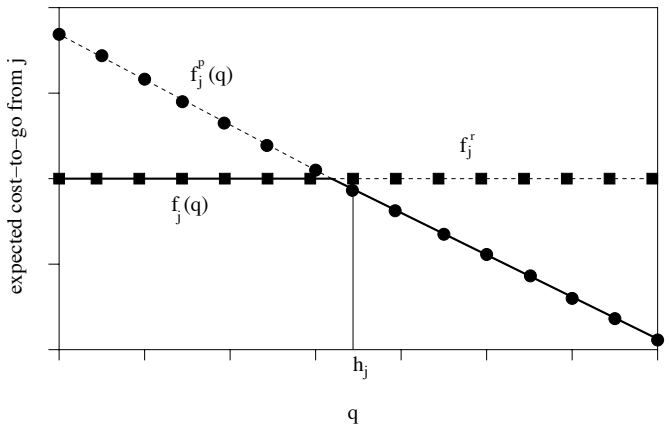


Figure B.1: The bold line is the cost function $f_j(q)$, that is, the total expected cost from node j onward. The quantity q is the residual capacity of the vehicle just after having serviced customer j. The function $f_j^p(q)$ would be the expected cost in case the vehicle proceeded directly to customer $j+1$ without first going to de depot for replenishment. The function (constant in q) $f_j^r(q)$ would be the expected cost 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 20 Computation of the VRPSD objective function $\mathbf{f_0(Q)}$

for $(q = Q, Q-1, ..., 0)$ **do**
  $f_n(q) = d_{n,0}$
  for $(j = n-1, n-3, ..., 1)$ **do**
    compute f_j^r using $f_{j+1}(\cdot)$ (by means of Equation (B.3))
    for $(q = Q, Q-1, ..., 0)$ **do**
      compute $f_j^p(q)$ (by means of Equation (B.2))
      compare f_j^r and $f_j^p(q)$ for finding the threshold h_j
      compute $f_j(q)$ using $f_{j+1}(\cdot)$ (by means of Equation (B.1))
    end for
  end for
end for
compute $f_0(Q)$
return $f_0(Q)$

this section. Jaillet [118, 119] and Jaillet-Odoni [120] derive analytic expressions for the computation of the expected length of a solution for the Probabilistic Traveling Salesman Problem and variations of it;

Bertsimas [25] proposes the cyclic heuristic for the VRPSD, by adapting to a stochastic framework one of the heuristics presented by Haimovitch and Rinnooy Kan [106] in a deterministic context; later, Bertsimas et al. [26] improve this heuristic by applying dynamic programming, to supplement the a priori tour with rules for selecting returns trips to the depot, similarly to the preventive restocking strategy; their computational experience suggests that the two versions of the cyclic heuristic provide good quality solutions when customers are randomly distributed on a square region;

Gendreau, Laporte and Séguin [86] present an exact stochastic integer programming method for VRPSCD (the same method can be applied to VRPSD as well); by means of the integer L-shaped method [131] they solve instances with up to 46 and 70 customers and 2 vehicles, for the VRPSCD and VRPSD, respectively; in [88], they also develop a tabu search algorithm called TABUSTOCH for the same problem; this algorithm is to be employed when instances become too large to be solved exactly by the L-shaped method;

Teodorović and Pavković [175] propose a Simulated Annealing algorithm to the multi-vehicle VRPSD, with the assumption that no more than one route failure is allowed during the service of each vehicle.

Gutjahr [100] applies S-ACO to the Traveling Salesman Problem with Time Windows, in case of stochastic service times. S-ACO is a simulation-based Ant Colony optimization algorithm, that computes the expected cost of a solution (the objective function), by Monte Carlo sampling.

Secomandi [162, 163] applies Neuro Dynamic Programming techniques to the VRPSD; he addresses the VRSPD with a re-optimization approach, where after each new exact information about customers demand is updated, the a priori tour planned on the not yet served customers is completely re-planned; this approach may find solutions with a lower expected value with respect to the preventive restocking strategy, but it is much more computationally expensive; moreover, the a priori tour may be completely different from the actual tour followed by the vehicle, and this situation is often seen as a disadvantage by companies;

Yang et al. [183] investigate the single- and multi-vehicle VRPSD; the latter is obtained by imposing that the expected distance traveled by each vehicle does not exceed a given value; the authors 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 for small size instances, as shown by comparisons with branch-and-bound solutions to instances with up to 15 customers; The authors also adapt to the stochastic case (both single- and multi-vehicle VRPSD) the OrOpt local search due to Or [147], by proposing a fast approximation computation for the change of the objective function value of a solution modified with a local search move.

The OrOpt local search and objective function approximation are used here as

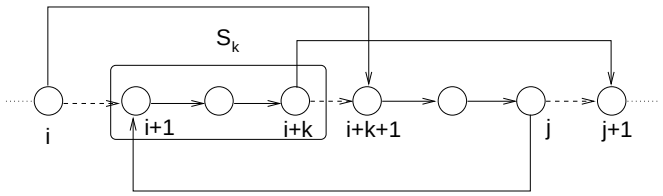


Figure B.2: How an a priori tour is modified after performing an OrOpt move, where the set of consecutive customers S_k (here, $k = 3$) is moved forward in the tour.

building blocks of our metaheuristics, therefore we will describe them in detail in the following subsection.

B.2.1 The OrOpt local search

A basic move of the OrOpt local search, as suggested by Yang et al. in [183], works as follows. Given a starting a priori tour, sets S_k of k consecutive customers with $k \in \{1, 2, 3\}$ are moved from one position to another in the tour, like in Figure B.2. In the following we describe the two types of approximation schemes used for the computation of the move cost. The first one, that we call *VRPSD*, is the one proposed in [183], and the second one, that we call *TSP*, only computes the length change of the a priori tour.

B.2.1.1 VRPSD approximation scheme

The move cost is computed in two stages: i) compute the saving from extracting the set of customers from the tour; ii) compute the cost of inserting it back somewhere else in the tour. Let i and $i+k+1$ be the nodes immediately preceding, respectively following, S_k in the tour, and let j be the node immediately after which S_k is to be inserted, as shown in Figure B.2. Here, we assume that j is *after* i in the a priori tour. Let $f_i(q)$ and $f_{i+k+1}(q)$ be the expected cost-to-go from nodes i, respectively $i + k + 1$ onward before the extraction of S_k. Apply one dynamic programming recursion step starting with cost vector $f_{i+k+1}(\cdot)$ at node $i + k + 1$ back to node i, without considering the sequence S_k. Let $f'_i(\cdot)$ be the resulting cost vector at node i, that is, after extracting S_k from the tour. Then, define the approximate extraction saving as a simple average over q of $f_i(q) - f'_i(q)$. The computation of the approximate insertion cost of S_k between nodes j and $j+1$ in the tour, is done analogously, if we assume that the insertion point (node j) is after the extraction point (node i). Let $f_j(q)$ be the cost-to-go at node j before inserting S_k, and $f''_j(q)$ be the cost-to-go at node j after inserting the S_k. The total approximate cost of an OrOpt move is computed by subtracting the approximate extraction saving from the approximate insertion cost, as follows

$$\Delta_{\text{VRPSD}} = \frac{\sum_{q=0}^{Q}[(f''_j(q) - f_j(q)) - (f_i(q) - f'_i(q))]}{Q+1} \,. \tag{B.4}$$

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 Equation (B.4). The only computations that must be done here are the evaluation of cost vectors $f'_{i+1}(\cdot)$ and $f''_j(\cdot)$, requiring $O(KQ)$ time, and the average of Equation (B.4), requiring $O(Q)$ time. Therefore, with the proposed *VRPSD* approximation, the cost of an OrOpt move can be computed in $O(KQ)$ time. Although it is possible that tours which are worsening with respect to the evaluation function are accepted because recognized as improving by the approximate evaluation, in practice this approximation scheme behave quite well. For a deeper discussion on the issues related with this scheme we refer the reader to the original paper [183].

B.2.1.2 TSP approximation scheme

In the *TSP* approximation scheme the cost of an OrOpt move coincides with the difference between the length of the tour before the move and after the move:

$$\Delta_{\text{TSP}} = d_{i,i+k+1} + d_{j,i+1} + d_{i+k,j+1} - d_{i,i+1} - d_{i+k,i+k+1} - d_{j,j+1}, \tag{B.5}$$

where, as before, i and j are the extraction, respectively insertion point of a string of k consecutive customers (see Figure B.2). Clearly, Δ_{TSP} is computable in constant time.

The OrOpt neighborhood examination follows the same scheme proposed in [183], and illustrated by Algorithm 21. Briefly, all possible sequences of length $k \in \{1, 2, 3\}$ are considered for insertion in a random position of the tour after the extraction point. Then, only the 'best' move among those of length k is chosen. The 'best' move is the move corresponding to the most negative move cost, which is computed by Equation (B.4) in the *VRPSD* approach and by Equation (B.5) in the *TSP* approach.

B.2.2 Benchmark

In the literature there is no commonly used benchmark for the VRPSD, therefore we have generated our own testbed. We have tried to consider instances which are 'interesting' from different points of view, by controlling four factors in the generation of instances: customer position, capacity over demand ratio, variance of the stochastic demand, and number of customers.

Instances may be divided into two groups, uniform and clustered, according to the position of customers. In uniform instances, the position of customers is chosen uniformly at random on a square of fixed size. In clustered instances, coordinates are chosen randomly with normal distributions around a given number of centers. This results in clusters of nodes, a typical situation for companies serving customers positioned in different cities.

The ratio between the total (average) demand of customers and the vehicle's capacity is an important factor that influences the 'difficulty' of a VRPSD instance [86]. The bigger the ratio, the more 'difficult' the instance. Here, the vehicle capacity Q is chosen as follows

$$Q = \left\lceil \frac{\text{total average demand} \cdot r}{n} \right\rceil, \tag{B.6}$$

Algorithm 21 OrOpt(p), with $p \in \{VRPSD, TSP\}$

let $k = 3$ and λ be an initial route
compute the expected cost of route λ by means of Algorithm 20
for all S^k, a set of successive nodes from λ **do**
  if $p = VRPSD$ **then**
    compute Δ_{VRSPD} by means of Equation (B.4)
  else if $p = TSP$ **then**
    compute Δ_{TSP} by means of Equation (B.5)
  end if
end for
if none of the sets S^k corresponds to a negative cost **then**
  go to Step 15
else
  select the set S^k that results in the most negative cost and perform the associated OrOpt move
end if
if $k = 1$ **then**
  stop
else
  decrease k by 1, and go to Step 3
end if

where the parameter r may be approximately interpreted as the average number of served customers before restocking.

Each customer's demand is an integer stochastic variable uniformly distributed on an interval. The demand interval for each customer i is generated using two parameters: the average demand D_i, and the spread S_i, so that the possible demand values for customer i are the $2S_i + 1$ integers in the interval $[D_i - S_i, D_i + S_i]$. The spread is a measure of the variance of the demand of each customer.

The particular parameters used to generate the test instances for our experiments are reported in Section B.4.

B.3 The metaheuristics

We aim at an *unbiased* comparison of the performance of well-known five different metaheuristics for this problem. In order to obtain a fair and meaningful analysis of the results, we have restricted the metaheuristic approaches to the use of the common OrOpt local search. In the following we briefly and schematically describe the main principles of each metaheuristic and give the details of the implementations for the VRPSD.

- **Simulated Annealing (SA)**

  Determine initial candidate solution s
  Set initial temperature T according to *annealing schedule*
  While termination condition not satisfied:
    Probabilistically choose a neighbor s' of s
    If s' satisfies probabilistic acceptance criterion (depending on T):
      $s := s'$
    Update T according to annealing schedule

  The initial temperature T_i is given by the average cost of a sample of 100 solutions of the initial tour multiplied by μ; every $\psi \cdot n$ iterations the temperature is updated by $T \leftarrow \alpha \times T$ (standard geometric cooling); after $\rho \cdot \psi \cdot n$ iterations without improvement, the temperature is increased by adding T_i to the current value; the solution considered for checking improvements is the best since the last re-heating.

- **Tabu Search (TS)**

  Determine initial candidate solution s
  While termination criterion is not satisfied:
    Choose the best neighbor s' of s that is either non-tabu or
      satisfies the aspiration criterion, and set $s := s'$
    Update tabu attributes based on s'

  The neighborhood of the current solution s is explored. The non-tabu neighbors are considered for the selection of the next current solution. If the value of a tabu neighbor is better than the best found solution, then this neighbor is also considered for selection (aspiration criterion). The best considered neighbor, i.e. the one with the lowest value, is selected. In order to avoid cycling around the same set of visited solutions, we found convenient to have a *variable neighborhood*: for any current solution s, instead of considering the whole neighborhood, we choose a subset of it according to a probability distribution. We experimentally verified that the used variable neighborhood is able to avoid cycles and to explore a larger part of the search space.

- **Iterated Local Search (ILS)**

  Determine initial candidate solution s
  Perform local search on s
  While termination criterion is not satisfied:
    $r := s$
    Perform perturbation on s
    Perform local search on s
    Based on acceptance criterion, keep s or revert to $s := r$

  The perturbation consists in a sampling of n neighbors according to the 2-opt exchange neighborhood [124]; each new solution is evaluated with by exact cost

function (Procedure 1) and if a solution is found that has cost smaller than the best solution found so far plus ε, the sampling ends; otherwise, the best solution obtained during the sampling is returned; the acceptance criterion keeps s if it is the best solution found so far.

- **Ant Colony Optimization (ACO)**

  Initialise weights (pheromone trails)
  While termination criterion is not satisfied:
  Generate a population *sp* of solutions by a randomised constructive heuristic
  Perform local search on *sp*
  Adapt weights based on *sp*

  Pheromone trails are initialised to τ_0; *sp* solutions are generated by a constructive heuristic and refined by local search; a *global update rule* is applied r times; *sp* solutions are then constructed by using information stored in the pheromone matrix; after each construction step a *local update rule* is applied to the element $\tau_{i,j}$ corresponding to the chosen customer pair: $\tau_{i,j} = (1 - \psi) \cdot \tau_{i,j} + \psi \cdot \tau_0$, with $\psi \in [0,1]$; after local search, weights are again updated by the *global update rule*: $\tau_{i,j} = (1 - \rho) \cdot \tau_{i,j} + \rho \cdot \frac{q}{C^{\text{bs}}}$, with $\rho \in [0,1]$ and C^{bs} the cost of the *best-so-far* solution; (note that heuristic information is only used in the initialisation phase).

- **Evolutionary Algorithm (EA)**

  Determine initial population *sp*
  While termination criterion is not satisfied:
  Generate a set *spr* of solutions by recombination
  Generate a set *spm* of solutions from *spr* by mutation
  Perform local search on *spm*
  Select population *sp* from solutions in *sp*, *spr*, and *spm*

  At each iteration two solutions are chosen among the best ones to generate a new solution *spr* through Edge Recombination [182] (a tour is generated using edges present in both two other tours, whenever possible); The mutation swaps adjacent customers (without considering the depot) with probability p_m; finally, the solution improved by local search replaces the worst solution in the population.

The initial solution(s) for all metaheuristics are obtained by the Farthest Insertion constructive heuristic [125]; it builds a tour by choosing as next customer the *not yet visited* customer which is farthest from the current one. Here, we consider a randomised version of this heuristic (RFI) which picks the first customer at random, and after the tour has been completed, shifts the starting customer to the depot.

In order to use a reference algorithm for comparison among metaheuristics (see Section B.4), we also implemented a simple random restart algorithm which uses the RFI heuristics *plus* local search and restarts every-time a local optimum is found, until the termination condition is reached; the best solution found among all the restarts is picked as the final solution; we call such algorithm RR.

B.4 Experimental Setup

Here we report two computational experiments with two different goals: i) we analyse metaheuristics performance to test the hypothesis of the relation between the TSP and VRPSD; ii) we study the performance of enhanced versions of the best algorithms found in the first set of experiments. Moreover, we relate these results with parameters of the instance.

We consider instances with 50, 100 and 200 customers and with customers uniformly distributed or grouped in clusters. In uniform instances, the position of customers is chosen uniformly at random in the square $[0, 99]^2$. Clustered instances are generated with two clusters, with centres randomly chosen in the square $[0, 99]^2$. The variance of the normal distribution used to generate customers coordinates around the centres is equal to $(0.8/\sqrt{n}) \cdot (\text{max. coordinate})$. This corresponds to variance values of about 11, 8, and 6, respectively for instances with 50, 100, and 200 customers. The position of the depot is fixed at (1,1). The average number of customers served before restocking is maintained fixed to 4, thus yielding ratios for total demand over vehicle capacity in the range from 12 to 50. Typical values for this ratio in the VRPSD literature are below 3, however higher values are closer to the needs of real contexts [34]. In all instances, average demands D_i at each customer i are taken with equal probability from $[1, 49]$ or $[50, 100]$. With regard to demand spread, instead, instances may be divided in two groups: low spread instances, in which each customer i has S_i chosen at random in $[1, 5]$, and high spread instances, in which each customer i has S_i chosen at random in $[10, 20]$. For each combination of size, distribution of customers and demand spread 75 instances were generated, making a total of 900 instances.

The metaheuristic parameters were chosen in order to guarantee robust performances over all the different classes of instances; preliminary experiments suggested the following settings:

SA: $\mu = 0.05$, $\alpha = 0.98$, $\psi = 1$, and $\rho = 20$;

TS: $p_{nt} = 0.8$ and $p_t = 0.3$;

ILS: $\varepsilon = \frac{n}{10}$;

ACO: $m = 5$, $\tau_0 = 0.5$, $\psi = 0.3$, $\rho = 0.1$, $q = 10^7$, and $r = 100$;

EA: $spm = 10$, $p_m = 0.5$.

Given the results reported in [43, 44], we decided to only perform one run for each metaheuristic on each instance.[1] The termination criterion for each algorithm was set to a time equal to 30, 120 or 470 seconds for instances respectively of 50, 100 or 200 customers. Experiments were performed on a cluster of 8 PCs with AMD Athlon(tm)

[1] In [43] it is formally proved that if a total of N runs of a metaheuristics can be performed for estimating its expected performance, the *best* unbiased estimator, that is the one with the least variance, is the one based on *one single run* on N randomly sampled (and therefore typically distinct) instances.

XP 2800+ CPU running GNU/Linux Debian 3.0 OS, and all algorithms were coded in C++ under the same development framework.

In order to compare results among different instances, we normalised results with respect to the performance of RR. For a given instance, we denote as c_{MH} the cost of the final solution of a metaheuristic MH, c_{RFI} the cost of the solution provided by the RFI heuristic, and C_{RR} by cost of the final solution provided by RR; the normalised value is then defined as

$$\text{Normalized Value for MH} = \frac{c_{MH} - c_{RR}}{c_{RFI} - c_{RR}}. \tag{B.7}$$

Besides providing a measure of performance independent from different instance hardness, this normalisation method gives an immediate evaluation of the minimal requirement for a metaheuristic; it is reasonable to request that a metaheuristic performs at least better than RR within the computation time under consideration.

B.5 First hybridization: using approximate move costs in local search

The main goal of this first experiment is to see whether approximating the exact but computationally demanding objective with the fast computing length of the a priori tour is convenient or not. Our hypothesis is that the speedup due to the use of a fast approximation of the objective is an advantage especially during the phase of local search, when many potential moves must be evaluated before one is chosen.

In order to test the advantage of a speedup across the metaheuristics, we apply to each of them the OrOpt local search described in Section B.2.1, and we test two versions for each metaheuristic according to the type of approximation scheme used in the local search, *VRPSD-approximation* or *TSP-approximation*. This set up allows to use statistical techniques for a systematic experimental analysis. In particular, we consider a two-way factor analysis, where metaheuristics and type of objective function are factors and instances are considered as blocks [66].

The first plot of Figure B.3 is the boxplot of the results over all instances after the normalisation. Negative values indicate an improvement over random restart and the larger is the absolute value the larger is the improvement. It emerges that, in average, ILS, EA and TS are able to do better than RR while SA and TS perform worse. We check whether these results are also statistically significant. The assumptions for a parametric analysis are not met, hence we rely on non-parametric methods.

The central plot of Figure B.3 shows the interaction between the two factors, metaheuristic and approximation scheme. Interaction plots give an idea of how different combinations of metaheuristic and approximation scheme affect the average normalised result. The lines join the average value for each metaheuristic. If lines are not parallel it means that there is an *interaction effect*, that is, metaheuristics perform differently with different approximation scheme. From the plot, we see that a certain interaction effect is present between the two factors, hence, it would be appropriate to report the

effects of one factor separately for each level of the other. In the third plot of Figure B.3 we report, however, together both VRPSD and TSP approximation schemes, but we distinguish the effects of this factor on each metaheuristic. The plot presents the simultaneous confidence intervals for the all-pairwise comparison of algorithms. Interval widths are obtained by the Friedman two-way analysis of variance by ranks [64], after rejecting the hypothesis that all algorithms perform the same. The difference between two algorithms is statistically significant at a level of confidence of 5% if their intervals do not overlap.

From the interaction and the all-pairwise comparison plots of Figure B.3 we can conclude that:

- The improvements of EA, ILS, and TS over RR are statistically significant.
- The presence of an interaction effect between metaheuristic and approximation scheme shows that EA, ILS and ACO perform better with *TSP-approximation* while the opposite result holds for TS and SA. While EA, ILS and ACO use the local search as a black-box, TS and SA employ their own strategy for examining the neighborhood in the local search. This result indicates that, for becoming competitive, these two methods require a good approximation of the objective function.
- The metaheuristics which perform better are EA, ILS and TS. Furthermore, EA and ILS take significant advantage from *TSP-approximation* scheme.

B.6 Second hybridization: further exploiting the TSP analogy

Given the results in the previous section, it is reasonable to investigate what happens if we exploit even more the hybridization based on the TSP objective function. For this purpose, we consider one of the best performing TSP state-of-the-art metaheuristics, and we observe that it is based on iterated local search with the 3-opt local search operator [170]. We, therefore, hybridize the best algorithms determined in the previous section (ILS, EA) with the 3-opt local search for TSP. We do not hybridize TS, instead, since we observed that it exhibits better results with the *VRPSD-approximation* rather than with the *TSP-approximation*. The new algorithms that we consider are the following.

TSP-soa If the solution found solving the TSP was comparable in quality to those found by our metaheuristics, there would be no point in investigating algorithms specific for the VRPSD. We consider therefore a transformation of the problem into TSP by focusing only on the coordinates of the customers (depot included). We then solve the TSP with a TSP state-of-the-art algorithm [170] and shift the TSP solution found to start with the depot thus yielding a VRPSD solution. This solution is finally evaluated by the VRPSD objective function (Procedure 1). The algorithm for TSP is an iterated

local search algorithm that uses a 3-opt exchange neighborhood, and a double bridge move as perturbation, i.e., it removes randomly four edges from the current tour and adds four other edges, also chosen at random, that close the tour.

ILS-hybrid It is obtained by modifying the acceptance criterion in *TSP-soa*, that is, the best solution according to Procedure 20 is chosen rather that the best TSP solution.

EA-hybrid It is derived from *EA-tsp* by replacing the local search with a 3-opt local search based on TSP objective function. The recombination and mutation operators are maintained unchanged, since these are deemed the components that determined the success of EA in Figure B.3. The selection operator remains also unchanged, and is based on the VRPSD objective function computed by Procedure 1.

In order to have a comparison with previous methods from the VRPSD literature, we also test the effective CYCLIC heuristic [25], that we implemented as follows:

- solve a TSP over the n customers, plus the depot, by using the TSP-soa algorithm;
- consider the n a priori tours obtained by the cyclic permutations $s_i = (0, i, i + 1, ..., n, 1, ..., i - 1), i = 1, 2, ..., n$;
- evaluate the n a priori tours by the VRPSD objective function (Procedure 1) and choose the best one.

For the comparison of these algorithms we designed an experimental analysis based on a single factor layout with blocks [66]. Since the assumption for parametric tests are again violated, we use the Friedman two-way analysis of variance by ranks.

In Figure B.4, B.5, B.6 we present the simultaneous confidence intervals after rejecting the hypothesis that all algorithms perform the same. We distinguish results by presenting the instances grouped according to the three main features defined in Section B.4. Since these the analysis is repeated on the same data, we adjust the "family-wise" level of confidence for each test dividing it by a factor of three [165].

The main indication arising from the results is that a mere TSP algorithm is not sufficient to produce high quality solutions for the VRPSD instances under any of the circumstances tested. VRPSD problem-specific algorithms, which take into account the stochasticity of the problem, yield always better solutions. Nevertheless, the best performances are achieved by hybridization of TSP and VRPSD approaches. *EA-hybrid* and *ILS-hybrid* in many cases perform statistically better than all other algorithms and, when significant, differences are also practically relevant. On the contrary, the difference between these two approaches is statistically significant only in one case. The comparison with the CYCLIC heuristic indicates that the algorithms we presented improve considerably the solution for VRPSD with respect to previous known approaches.

Considering the characteristics of the instances, the position of customers seems not to have an impact on the rank of algorithms. On the contrary, the different position occupied by the algorithms in the ranks with respect to instance size indicates that the larger the instances the higher is the importance of speed over precision in the

evaluation of neighbors in local search. This expected result is exemplified by the worsening of the average rank of *EA-tsp* as size increases.

Finally, spread of demand has also a considerable influence. With small spread it is convenient to consider the costs of "preventive restocking" and thus the use of Procedure 1 is appropriate. With large spread, on the contrary, the higher uncertainty of the right point in the tour for "preventive restocking" makes the evaluation of solution by means of Procedure 20 not far from that provided by a TSP evaluation, with the negative impact of a higher computational cost.

B.7 Conclusions

The main contribution of this section is the study of the hybridization of objective function approximations on five well known metaheuristics to solve the VRPSD. In particular, we introduced a *TSP-approximation*, which uses the length of the a priori tour as an approximation of the exact but computationally demanding VRPSD evaluation. We showed its superiority with respect to a known *VRPSD-approximation*.

More precisely, first we considered a local search algorithm (OrOpt) for which a good approximation for the exact VRPSD objective function is available, and we compared metaheuristics using this local search by applying both the *VRPSD-approximation* and the *TSP-approximation*. Secondly, we exploited further the TSP analogy, by choosing the 3-opt local search operator, which is very good for the TSP, but for which there is no immediate VRPSD approximation.

With the first type of hybridization, we have shown that metaheuristics using the local search as a black-box (EA, ILS and ACO) perform better when using the *TSP-approximation*, while metaheuristics that use their own strategy for examining the neighborhood in the local search, perform better with the more precise but more computationally expensive *VRPSD-approximation*. With the second type of hybridization based on the use of the 3-opt local search, we considerably improved the performance of the best metaheuristics for VRPSD, and we were unable to discover significant differences between the two best algorithms, EA and ILS.

All the metaheuristics implemented found better solutions with respect to the CYCLIC heuristic (which is known from the literature to perform well on different types of instances) and with respect to solving the problem as a TSP. This latter point is important because it demonstrates that the stochasticity of the problem and the finite demand over capacity ratio is not negligible and that the development of VRPSD-specific algorithms is needed. We proposed a TSP-VRPSD hybrid approach that clearly outperforms the current state-of-the-art.

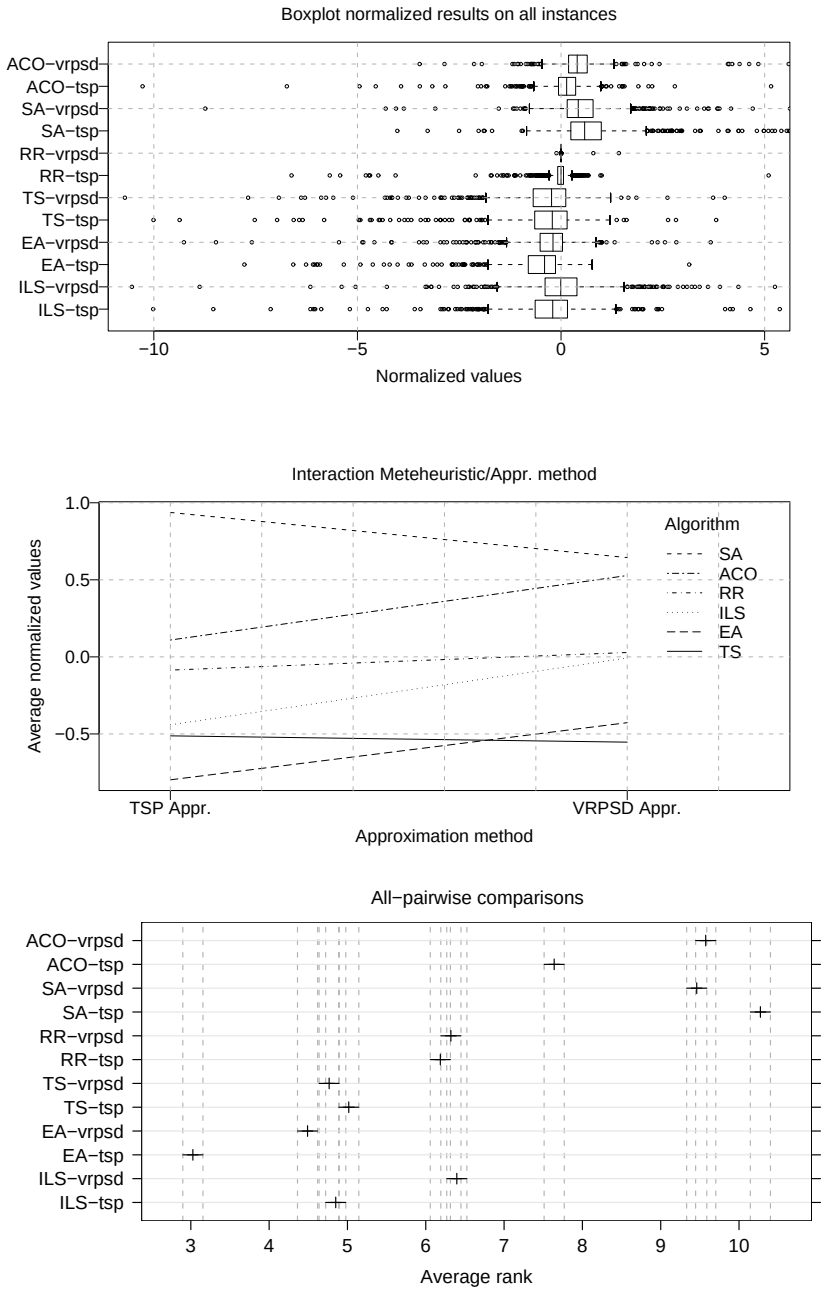


Figure B.3: Aggregate results over all 900 instances. From top to bottom: boxplot of normalised results; interaction plot for the two factors: metaheuristic and objective function approximation scheme; error bars plot for simultaneous confidence intervals in the all-pairwise comparison. The boxplot is a restricted to values in $[-10.5, 5]$, few outliers fell outside this interval. In the error bar plot, ranks range from 1 to 12, the number of algorithms considered in the experiment.

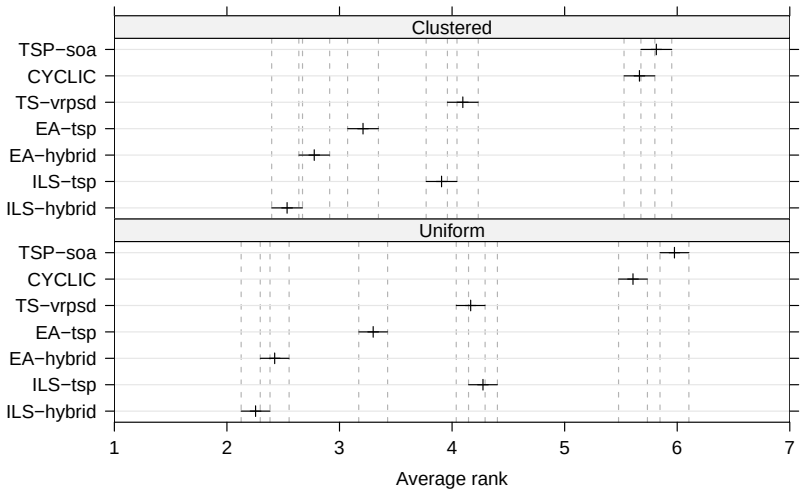


Figure B.4: All-pairwise comparisons by means of simultaneous confidence intervals on uniform and clustered instances.

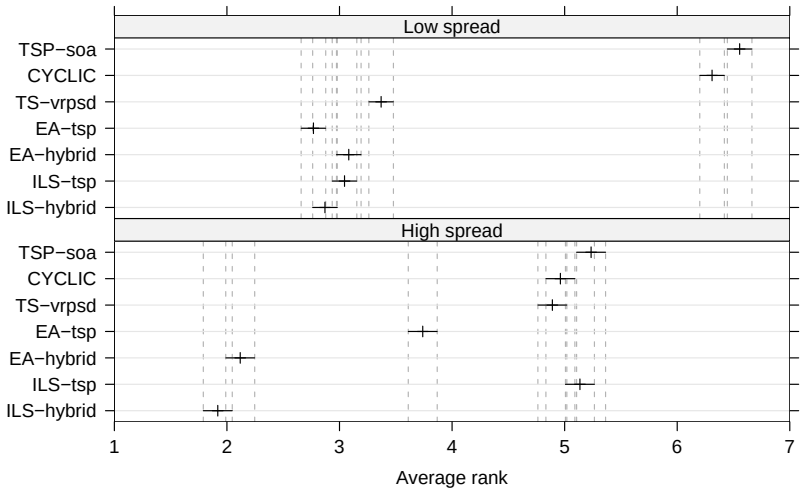


Figure B.5: All-pairwise comparisons by means of simultaneous confidence intervals on instances with different demand spread.

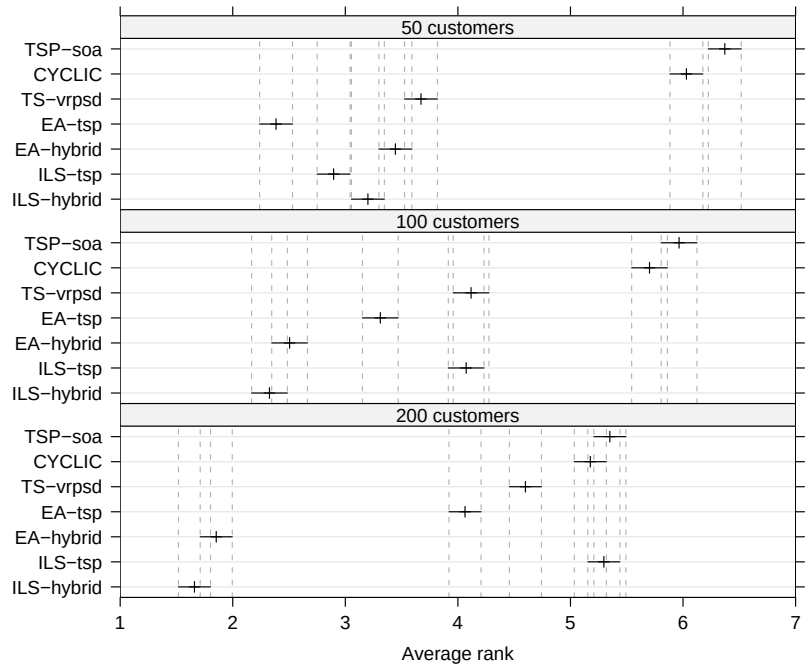


Figure B.6: All-pairwise comparisons by means of simultaneous confidence intervals on instances with different number of customers.

Bibliography

[1] E. Aarts and J. Korst. *Simulated Annealing and the Boltzmann Machine*. John Wiley & Sons, New York, NY, USA, 1990.

[2] H. Abelson and A. di Sessa. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. The MIT Press, Cambridge, MA, USA, 1982.

[3] A. N. Aizawa and B. W. Wah. Scheduling of genetic algorithms in a noisy environment. *Evolutionary Computation*, 2:97–122, 1994.

[4] S. Albers. Online algorithms: A survey. *Mathematical Programming*, 97(1-2):3–26, 2003.

[5] T. M. Alkhamis and M. A. Ahmed. Simulation-based optimization using simulated annealing with confidence intervals. In R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, editors, *Proceedings of the 2004 Winter Simulation Conference (WSC04)*, pages 514–518. IEEE Press, Piscataway, NJ, USA, 2004.

[6] T. M. Alkhamis, M. A. Ahmed, and W. Kim Tuan. Simulated annealing for discrete optimization with estimation. *European Journal of Operational Research*, 116:530–544, 1999.

[7] M. H. Alrefaei and S. Andradóttir. A simulated annealing algorithm with constant temperature for discrete stochastic optimization. *Management Science*, 45:748–764, 1999.

[8] S. Andradóttir. A review of simulation optimization techniques. In D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, editors, *Proceedings of the 1998 Winter Simulation Conference (WSC98)*, pages 151–158. IEEE Press, Piscataway, NJ, USA, 1998.

[9] R. Aringhieri. Solving chance-constrained programs combining tabu search and simulation. In C. C. Ribeiro and S. L. Martins, editors, *Proceedings of the 3rd International Workshop on Experimental and Efficient Algorithms (WEA04)*, volume 3059 of *Lecture Notes in Computer Science*, pages 30–41. Springer, Berlin, Germany, 2004.

[10] D. Arnold. In *Noisy Optimization with Evolutionary Strategies*, volume 8 of *Genetic Algorithms and Evolutionary Computation Series*. Kluwer Academic Publishers, Boston, MA, USA, 2002.

[11] T. Baeck, D. Fogel, and Z. Michalewicz, editors. *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol, UK, 2000.

[12] J.-F. M. Barthelemy and R. T. Haftka. Approximation concepts for optimum structural design - a review. *Structural Optimization*, 5:129–144, 1993.

[13] J. J. Bartholdi III and L. K. Platzman. An O(N logN) planar travelling salesman heuristic based on spacefilling curves. *Operations Research Letters*, 1(4):121–125, 1982.

[14] T. Beielstein and S. Markon. Threshold selection, hypothesis tests, and DOE methods. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, volume 1, pages 12–17. IEEE Press, Piscataway, NJ, USA, 2002.

[15] R. Bellman and L. A. Zadeh. Decision-making in a fuzzy environment. *Management Science*, 17:141–161, 1970.

[16] P. Beraldi and A. Ruszczyński. Beam search heuristic to solve stochastic integer problems under probabilistic constraints. *European Journal of Operational Research*, 167(1):35–47, 2005.

[17] O. Berman and D. Simchi-Levi. Finding the optimal a priori tour and location of a traveling salesman with nonhomogeneous customers. *Transportation Science*, 22(2):148–154, 1988.

[18] O. Berman and D. Simchi-Levi. Finding the optimal a priori tour and location of a traveling salesman with nonhomogeneous customers. Technical Report WP-08-86, Faculty of Management, University of Calgary, Calgary, Canada, April 1988.

[19] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Volumes 1 and 2*. Athena Scientific, Belmont, MA, USA, 1995.

[20] D. P. Bertsekas. *Network Optimization: Continuous and Discrete Models*. Athena Scientific, Belmont, MA, USA, 1998.

[21] D. P. Bertsekas and D. A. Castañon. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5:89–108, 1998.

[22] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic Programming*. Athena Scientific, Belmont, MA, USA, 1996.

[23] D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3(3):245–262, 1997.

[24] D. J. Bertsimas. *Probabilistic Combinatorial Optimization Problems*. PhD thesis, MIT, Cambridge, MA, USA, 1988.

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

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

[27] D. J. Bertsimas and L. Howell. Further results on the probabilistic traveling salesman problem. *European Journal of Operational Research*, 65(1):68–95, 1993.

[28] D. J. Bertsimas, P. Jaillet, and A. Odoni. A priori optimization. *Operations Research*, 38(6):1019–1033, 1990.

[29] 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.

[30] H.-G. Beyer. Evolutionary algorithms in noisy environments: Theoretical issues and guidelines for practice. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):239–267, 2000.

[31] H.-G. Beyer, E. Brucherseifer, W. Jakob, H. Pohlheim, B. Sendhoff, and T. Binh To. Evolutionary algorithms - terms and definitions. `http://ls11-www.cs.uni-dortmund.de/people/beyer/EA-glossary/def-engl-html.html`.

[32] 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. In X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo Guervós, J. A. Bullinaria, J. Rowe, P. Tiňo, A. Kabán, and H.-P. Schwefel, editors, *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, volume 3242 of *Lecture Notes in Computer Science*, pages 450–460. Springer, Berlin, Germany, 2004.

[33] L. Bianchi, M. Birattari, M. Manfrin, M. Mastrolilli, L. Paquete, O. Rossi-Doria, and T. Schiavinotto. Hybrid metaheuristics for the vehicle routing problem with stochastic demands. *Journal of Mathematical Modelling and Algorithms*. To appear.

[34] L. Bianchi, M. Birattari, M. Manfrin, M. Mastrolilli, L. Paquete, O. Rossi-Doria, and T. Schiavinotto. Research on the vehicle routing problem with stochastic demand. Technical Report IDSIA-07-04, IDSIA - Dalle Molle Institute for Artificial Intelligence, Manno, Switzerland, March 2004.

[35] L. Bianchi and A. M. Campbell. Extension of the 2-p-opt and 1-shift algorithms to the heterogeneous probabilistic traveling salesman problem. *European Journal of Operational Research*. To appear.

[36] L. Bianchi and A. M. Campbell. Extension of the 2-p-opt and 1-shift algorithms to the heterogeneous probabilistic traveling salesman problem. Technical Report IDSIA-01-04, IDSIA - Dalle Molle Institute for Artificial Intelligence, Manno, Switzerland, 2004.

[37] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr. Metaheuristics in stochastic combinatorial optimization: a survey. Technical Report IDSIA-08-06, IDSIA - Dalle Molle Institute for Artificial Intelligence, Manno, Switzerland, 2006.

[38] L. Bianchi, L. M. Gambardella, and M. Dorigo. An ant colony optimization approach to the probabilistic traveling salesman problem. In J. J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacañas, and H.-P. Schwefel, editors, *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN VII)*, volume 2439 of *Lecture Notes in Computer Science*, pages 883–892. Springer, London, UK, 2002.

[39] L. Bianchi, L. M. Gambardella, and M. Dorigo. Solving the homogeneous probabilistic traveling salesman problem by the ACO metaheuristic. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Proceedings of the 3rd International Workshop on Ant Algorithms (ANTS 2002)*, volume 2463 of *Lecture Notes in Computer Science*, pages 176–187. Springer, London, UK, 2002.

[40] L. Bianchi, L. M. Gambardella, and M. Dorigo. Ant colony optimization and local search for the probabilistic traveling salesman problem. Technical Report IDSIA-02-06, IDSIA - Dalle Molle Institute for Artificial Intelligence, Manno, Switzerland, 2006.

[41] L. Bianchi and J. Knowles. Local search for the probabilistic traveling salesman problem: a proof of the incorrectness of Bertsimas' proposed 2-p-opt and 1-shift algorithms. Technical Report IDSIA-21-02, IDSIA - Dalle Molle Institute for Artificial Intelligence, Manno, Switzerland, 2002.

[42] L. Bianchi, J. Knowles, and N. Bowler. Local search for the probabilistic traveling salesman problem: correction to the 2-p-opt and 1-shift algorithms. *European Journal of Operational Research*, 162(1):206–219, 2005.

[43] M. Birattari. On the estimation of the expected performance of a metaheuristic on a class of instances. How many instances, how many runs? Technical Report TR/IRIDIA/2004-01.2, IRIDIA - Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle, Univeristé Libre de Bruxelles, Brussels, Belgium, 2004.

[44] M. Birattari. *The Problem of Tuning Metaheuristics, as seen from a machine learning perspective*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2004.

[45] M. Birattari, P. Balaprakash, and M. Dorigo. ACO/F-Race: ant colony optimization and racing techniques for combinatorial optimization under uncertainty. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. J. Gutjahr, R. F. Hartl, and M. Reimann, editors, *Proceedings of the 6th Metaheuristics International Conference (MIC 2005)*, pages 107–112, 2005.

[46] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer, New York, NY, USA, 1997.

[47] Z. W. Birnbaum. On random variables with comparable peakedness. *Annals of Mathematical Statistics*, 19:76–81, 1948.

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

[49] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, Cambridge, MA, USA, 1998.

[50] N. E. Bowler, T. M. A. Fink, and R. C. Ball. Characterization of the probabilistic traveling salesman problem. *Physical Review E*, 68(036703), 2003.

[51] J. Branke and M. Guntsch. New ideas for applying ant colony optimization to the probabilistic TSP. In *Proceedings of the 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2003)*, volume 2611 of *Lecture Notes in Computer Science*, pages 165–175. Springer, Berlin, Germany, 2003.

[52] J. Branke and M. Guntsch. Solving the probabilistic TSP with ant colony optimization. *Journal of Mathematical Modelling and Algorithms*, 3(4):403–425, 2004.

[53] J. Branke and C. Schmidt. Selection in the presence of noise. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U. M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 766–777. Springer, Berlin, Germany, 2003.

[54] M. Brusco and L. Jacobs. A simulated annealing approach to the cyclic staff-scheduling problem. *Naval Research Logistics*, 40(1):69–84, 1993.

[55] M. Brusco and L. Jacobs. A simulated annealing approach to the solution of flexible labour scheduling problems. *Journal of the Operational Research Society*, 44(12):1191–1200, 1993.

[56] A. A. Bulgak and J. L. Sanders. Integrating a modified simulated annealing algorithm with the simulation of a manufacturing system to optimize buffer sizes in automatic assembly systems. In M. Abrams, P. Haigh, and J. Comfort, editors,

Proceedings of the 1988 Winter Simulation Conference (WSC98), pages 684–690. IEEE Press, Piscataway, NJ, USA, 1988.

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

[58] E. Cantú-Paz. Adaptive sampling for noisy problems. In K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, volume 3103 of *Lecture Notes in Computer Science*, pages 947–958. Springer, Berlin, Germany, 2004.

[59] H. S. Chang. An ant system based exploration-exploitation for reinforcement learning. In *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, pages 3805–3810. IEEE Press, Piscataway, NJ, USA, 2004.

[60] H. S. Chang, W. J. Gutjahr, J. Yang, and S. Park. An ant system approach to Markov decision processes. In *Proceedings of the 23rd American Control Conference (ACC04)*, volume 4, pages 3820–3825. IEEE Press, Piscataway, NJ, USA, 2004.

[61] H. S. Chang, H-G. Lee, M. Fu, and S. I. Marcus. Evolutionary policy iteration for solving Markov decision processes. *IEEE Transactions on Automatic Control*, 2005. To appear.

[62] P. Chervi. A computational approach to probabilistic vehicle routing problems. Master's thesis, MIT, Cambridge, MA, USA, 1988.

[63] Concorde TSP solver. `http://www.tsp.gatech.edu/concorde.html`.

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

[65] D. Costa and E. A. Silver. Tabu search when noise is present: an illustration in the context of cause and effect analysis. *Journal of Heuristics*, 4:5–23, 1998.

[66] A. Dean and D. Voss. *Design and Analysis of Experiments*. Springer, New York, NY, USA, 1999.

[67] B. Dengiz and C. Alabas. Simulation optimization using tabu search. In J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, editors, *Proceedings of the 2000 Winter Simulation Conference (WSC00)*, pages 805–810. IEEE Press, Piscataway, NJ, USA, 2000.

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

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

[70] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: an autocatalytic optimization process. Technical Report 91-016, Department of Electronics, Politecnico di Milano, Milan, Italy, 1991.

[71] 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.

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

[73] M. Dyer and L. Stougie. Computational complexity of stochastic programming problems. Technical Report SPOR-report 2003-20, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2003.

[74] F. Easton and N. Mansour. A distributed genetic algorithm for deterministic and stochastic labor scheduling problems. *European Journal of Operational Research*, 118(3):505–523, 1999.

[75] F. Easton and D. Rossin. A stochastic goal program for employee scheduling. *Decision Sciences*, 27(3):541–568, 1996.

[76] E. Erel, I. Sabuncuoglu, and H. Sekerci. Stochastic assembly line balancing using beam search. *International Journal of Production Research*, 43(7):1411–1426, 2005.

[77] D. A. Finke, D. J. Medeiros, and M. Traband. Shop scheduling using tabu search and simulation. In E. Yücesan, C. H. Chen, J. L. Snowdon, and J. M. Charnes, editors, *Proceedings of the 2002 Winter Simulation Conference (WSC02)*, pages 1013–1017. IEEE Press, Piscataway, NJ, USA, 2002.

[78] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial intelligence through simulated evolution*. John Wiley & Sons, New York, NY, USA, 1966.

[79] B. L. Fox and G. W. Heine. Probabilistic search with overrides. *Annals of Applied Probability*, 4:1087–1094, 1995.

[80] M. C. Fu. Optimization for simulation: Theory vs. practice. *INFORMS Journal on Computing*, 14(3):192–215, 2002.

[81] M. C. Fu. Guest editorial of the ACM TOMACS special issue on "simulation optimization". *ACM Transactions on Modeling and Computer Simulation*, 13(2):105–107, 2003.

[82] L. M. Gambardella and M. Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, pages 622–627. IEEE Press, Piscataway, NJ, USA, 1996.

[83] S. B. Gelfand and S. K. Mitter. Analysis of simulated annealing for optimization. In *Proceedings of the 24th IEEE Conference on Decision and Control (CDC'85)*, volume 2, pages 779–786. IEEE Press, Piscataway, NJ, USA, 1985.

[84] S. B. Gelfand and S. K. Mitter. Simulated annealing with noisy or imprecise measurements. *Journal of Optimization Theory and Applications*, 69:49–62, 1989.

[85] D. Geman and S. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. In *IEEE Transactions of Pattern Analysis and Machine Intelligence*, volume 6, pages 721–741, 1984.

[86] 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.

[87] M. Gendreau, G. Laporte, and R. Séguin. Stochastic vehicle routing. *European Journal of Operational Research*, 88:3–12, 1996.

[88] 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):469–477, 1996.

[89] M. Giacobini, M. Tomassini, and L. Vanneschi. Limiting the number of fitness cases in genetic programming using statistics. In J. J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacañas, and H.-P. Schwefel, editors, *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN-VII)*, volume 2439 of *Lecture Notes in Computer Science*, pages 371–380. Springer, London, UK, 2002.

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

[91] F. Glover. Tabu search - part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[92] F. Glover. Future paths for integer programming and links to artificial intelligence. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenaurer, and D. Snyers, editors, *Artificial Evolution*, volume 1363 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 1998.

[93] F. Glover. Tabu search and finite convergence. *Discrete Applied Mathematics*, 119:3–36, 2002.

[94] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[95] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes, 3rd Edition*. Oxford University Press, New York, NY, USA, 2001.

[96] G. Gutin and A. Punnen, editors. *The Traveling Salesman Problem and its Variations*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[97] W. J. Gutjahr. A graph-based Ant System and its convergence. *Future Generation Computer Systems*, 16(8):873–888, 2000.

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

[99] W. J. Gutjahr. A converging ACO algorithm for stochastic combinatorial optimization. In *Proceedings of the 2nd Symposium on Stochastic Algorithms, Foundations and Applicaions (SAGA 2003)*, volume 2827 of *Lecture Notes in Computer Science*, pages 10–25. Springer, Berlin, Germany, 2003.

[100] W. J. Gutjahr. S-ACO: An ant-based approach to combinatorial optimization under uncertainty. In *Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2004)*, volume 3172 of *Lecture Notes in Computer Science*, pages 238–249. Springer, Berlin, Germany, 2004.

[101] W. J. Gutjahr, A. Hellmayr, and G. Ch. Pflug. Optimal stochastic single-machine tardiness scheduling by stochastic branch-and-bound. *European Journal of Operational Research*, 117:396–413, 1999.

[102] W. J. Gutjahr and G. Ch. Pflug. Simulated annealing for noisy cost functions. *Journal of Global Optimization*, 8:1–13, 1996.

[103] W. J. Gutjahr, C. Strauss, and M. Toth. Crashing of stochastic activities by sampling and optimization. *Business Process Management Journal*, 6:65–83, 2000.

[104] W. J. Gutjahr, C. Strauss, and E. Wagner. A stochastic branch-and-bound approach to activity crashing in project management. *INFORMS Journal on Computing*, 12:125–135, 2000.

[105] J. Haddock and J. Mittenthal. Simulation optimization using simulated annealing. *Computers & Industrial Engineering*, 22:387–395, 1992.

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

[107] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13:311–329, 1988.

[108] S. Hanafi. On the convergence of tabu search. *Journal of Heuristics*, 7:47–58, 2000.

[109] W. K. K. Haneveld and M. H. van der Vlerk. Stochastic integer programming: state of the art. *Annals of Operations Research*, 85:39–57, 1999.

[110] P. Hansen. The steepest ascent mildest descent heuristics for combinatorial programming. Talk presented at the Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy, 1986.

[111] K. K. Haugen, A. Løkketangen, and D. L. Woodruff. Progressive hedging as a meta-heuristic applied to stochastic lot-sizing. *European Journal of Operational Research*, 132:116–122, 2001.

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

[113] A. Hertz, E. Taillard, and D. de Werra. Tabu search. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 121–136. John Wiley & Sons, New York, NY, USA, 1997.

[114] J. H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Harbor, MI, USA, 1975.

[115] T. Homem-de-Mello. Variable-sample methods and simulated annealing for discrete stochastic optimization. Stochastic Programming E-Print Series, `http://hera.rz.hu-berlin.de/speps/`, 2000.

[116] T. Homem-de-Mello. Variable-sample methods for stochastic optimization. *ACM Transactions on Modeling and Computer Simulation*, 13:108–133, 2003.

[117] S. Irani, X. Lu, and A. Regan. On-line algorithms for the dynamic traveling repair problem. *Journal of Scheduling*, 7(3):243–258, 2004.

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

[119] 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.

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

[121] O. Jellouli and E. Châtelet. Monte Carlo simulation and genetic algorithm for optimising supply chain management in a stochastic environment. In *Proceedings of the 2001 IEEE Conference on Systems, Man, and Cybernetics*, volume 3, pages 1835–1839. IEEE Press, Piscataway, NJ, USA, 2001.

[122] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2005.

[123] Y. Jin. Evolutionary optimization in uncertain environments - a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

[124] D. S. Johnson and L. A. McGeoch. The travelling 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, USA, 1997.

[125] D. S. Johnson and L. A. McGeoch. Experimental analysis of heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[126] H. Jönsson and E. A. Silver. Some insights regarding selecting sets of scenarios in combinatorial stochastic problems. *Journal of Production Economics*, 45:463–472, 1996.

[127] P. Kall and S. W. Wallace. *Stochastic Programming*. John Wiley & Sons, Chichester, UK, 1994. Wiley has released the copyright on the book, and the authors made the text available to the scientific community: it can be downloaded for free at `http://www.unizh.ch/ior/Pages/Deutsch/Mitglieder/Kall/bib/ka-wal-94.pdf`.

[128] A. Kenyon and D. P. Morton. A survey on stochastic location and routing problems. *Central European Journal of Operations Research*, 9:277–328, 2002.

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

[130] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*, volume 14 of *Nonconvex optimization and its applications*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997.

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

[132] G. Laporte, F. Louveaux, and H. Mercure. An exact solution for the a priori optimization of the probabilistic traveling salesman problem. *Operations Research*, 42(3):543–549, 1994.

[133] S. Lin. Computer solution of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269, 1965.

[134] Z.-Z. Lin, J. C. Bean, and C. C. White III. Genetic algorithm heuristics for finite horizon partially observed Markov decision processes. Technical Report 98-24,

Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI ,USA, 1998.

[135] Z.-Z. Lin, J. C. Bean, and C. C. White III. A hybrid genetic/optimization algorithm for finite-horizon, partially observed Markov decision processes. *INFORMS Journal on Computing*, 16(1):27–38, 2004.

[136] A. Løkketangen and D. L. Woodruff. Progressive hedging and tabu search applied to mixed integer (0,1) multistage stochastic programming. *Journal of Heuristics*, 2:111–128, 1996.

[137] H. R. Lourenço, O. Martin, and T. Stützle. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management*, chapter Iterated Local Search, pages 321–353. Kluwer Academic Publishers, Boston, USA, 2002.

[138] C. M. Lutz, K. R. Davis, and M. Sun. Determining buffer location and size in production lines using tabu search. *European Journal of Operational Research*, 106:301–316, 1998.

[139] K. L. Mak and Z. G. Guo. A genetic algorithm for vehicle routing problems with stochastic demand and soft time windows. In M. H. Jones, S. D. Patek, and B. E. Tawney, editors, *Proceedings of the 2004 IEEE Systems and Information Engineering Design Symposium (SIEDS04)*, pages 183–190. IEEE Press, Piscataway, NJ, USA, 2004.

[140] S. Markon, D. V. Arnold, T. Bäck, T. Beielstein, and H.-G. Beyer. Thresholding – a selection operator for noisy ES. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2001)*, volume 1, pages 465–472. IEEE Press, Piscataway, NJ, USA, 2001.

[141] 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.

[142] B. L. Miller and D. E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131, 1997.

[143] Metaheuristics Network web site. `http://www.metaheuristics.org/`.

[144] V. I. Norkin, Y. M. Ermoliev, and A. Ruszczyński. On optimal allocation of indivisibles under uncertainty. *Operations Research*, 46(3):381–395, 1998.

[145] V. I. Norkin, G. Ch. Pflug, and A. Ruszczyński. A branch and bound method for stochastic global optimization. *Mathematical Programming*, 83:425–450, 1998.

[146] S. Ólafsson and J. Kim. Simulation optimization. In E. Yücesan, C. H. Chen, J. L. Snowdown, and J. M. Charnes, editors, *Proceedings of the 2002 Winter Simulation Conference (WSC02)*, pages 89–84. IEEE Press, Piscataway, NJ, USA, 2002.

[147] I. Or. *Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking*. PhD thesis, Department of Industrial Engineering and Management Sciences, Nortwestern University, Evanston, IL, USA, 1976.

[148] J. Pichitlamken. *A combined procedure for optimization via simulation*. PhD thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, USA, 2002.

[149] J. Pichitlamken and L. B. Nelson. Selection-of-the-best procedures for optimization via simulation. In B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, editors, *Proceedings of the 2001 Winter Simulation Conference (WSC01)*, pages 401–407. IEEE Press, Piscataway, NJ, USA, 2001.

[150] J. Pichitlamken and L. B. Nelson. A combined procedure for optimization via simulation. *ACM Transactions on Modeling and Computer Simulation*, 13(2):155–179, 2003.

[151] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York, NY, USA, 2005.

[152] M. Rauner, S. C. Brailsford, W. J. Gutjahr, and W. Zeppelzauer. Optimal screening policies for diabetic retinopathy using a combined discrete event simulation and ant colony optimization approach. In J. G. Andersen and M. Katzper, editors, *Proceedings of the 15th International Conference on Health Sciences Simulation, Western MultiConference 2005*, pages 147–152. SCS - Society of Computer Simulation International, San Diego, CA, USA, 2005.

[153] R. I. Rechenberg. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, Germany, 1973.

[154] C. R. Reeves and J. E. Rowe. *Genetic Algorithms: Principles and Perspectives - a Guide to GA Theory*. Operaations Research/Computer Science Interfaces Series. Kluwer Academic Publishers, Boston, MA, USA, 2003.

[155] R. T. Rockafellar and R. J.-B. Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16:119–147, 1991.

[156] N. Roenko. Simulated annealing under uncertainty. Technical report, Institute for Operations Research, University of Zurich, Switzerland, 1990.

[157] S. L. Rosen and C. M. Harmonosky. An improved simulated annealing simulation optimization method for discrete parameter stochastic systems. *Computers & Operations Research*, 32(2):343–358, 2005.

[158] S. Rosenow. A heuristic for the probabilistic TSP. In W. Herroelen, E. Demeulemeester, B. De Reyck, U. Zimmerman, U. Derigs, W. Gaul, R. H. Mohring, and K. P. Schuster, editors, *Operations Research Proceedings 1996*. Springer, Berlin, Germany, 1997.

[159] F. A. Rossi and I. Gavioli. Aspects of heuristic methods in the probabilistic traveling salesman problem. In *Advanced School on Statistics in Combinatorial Optimization*, pages 214–227. World Scientific, Singapore, 1987.

[160] R. Y. Rubinstein. *Simulation and the Monte Carlo method*. John Wiley & Sons, New York, NY, USA, 1981.

[161] G. Rudolph. Convergence of evolutionary algorithms in general search spaces. In *Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC'96)*, pages 50–54. IEEE Press, Piscataway, NJ, USA, 1996.

[162] N. Secomandi. Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. *Computers & Operations Research*, 27(5):1171–1200, 2000.

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

[164] N. Secomandi. Analysis of a rollout approach to sequencing problems with stochastic routing applications. *Journal of Heuristics*, 9:321–352, 2003.

[165] D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, Boca Raton, FL, USA, 2nd edition, 2000.

[166] L. Shi and S. Ólafsson. Nested partitions method for global optimization. *Operations Research*, 48(3):390–407, 2000.

[167] P. Stagge. Averaging efficiently in the presence of noise. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN-V)*, volume 1498 of *Lecture Notes in Computer Science*, pages 188–200. Springer, Berlin, Germany, 1998.

[168] Stochastic Programming Community Homepage. `http://stoprog.org/`.

[169] 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.

[170] T. Stützle and H. Hoos. In P. Hansen and C. Ribeiro, editors, *Essays and Surveys on Metaheuristics*, chapter Analyzing the Run-time Behaviour of Iterated Local Search for the TSP, pages 589–612. Kluwer Academic Publishers, Boston, USA, 2002.

[171] J. Sudhir Ryan Daniel and C. Rajendran. A simulation-based genetic algorithm for inventory optimization in a serial supply chain. *International Transactions in Operational Research*, 12(1):101–127, 2005.

[172] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. The MIT Press, Cambridge, MA, USA, 1998.

[173] J. R. Swisher, S. H. Jacobson, and E. Yücesan. Discrete-event simulation optimization using ranking, selection, multiple comparison procedures: a survey. *ACM Transactions on Modeling and Computer Simulation*, 13(2):134–154, 2003.

[174] A. Teller and D. Andre. Automatically choosing the number of fitness cases: The rational allocation of trials. In J. R. Koza, D. Kalyanmoy, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Proceedings of the 2nd Annual Genetic Programming Conference (GP-97)*, pages 321–328. Morgan Kaufmann, San Francisco, CA, USA, 1997.

[175] D. Teodorović and G. Pavković. A simulated annealing technique approach to the vehicle routing problem in the case of stochastic demand. *Transportation Planning and Technology*, 16:261–273, 1992.

[176] G. Tesauro and G. R. Galperin. On-line policy improvement using monte carlo search. *Advances in Neural Information Processing Systems*, 9:1068–1074, 1997.

[177] J. N. Tsitsiklis. Asyncronous stochastic approximation and Q-learning. *Machine Learning*, 16:185–202, 1994.

[178] TSPLIB. `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.

[179] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, Dordrecht, The Netherlands, 1987.

[180] M. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. The MIT Press, Cambridge, MA, USA, 1999.

[181] J. P. Watson, S. Rana, L. D. Whitley, and Howe A. E. The impact of approximate evaluation on the performance of search algorithms for warehouse scheduling. *Journal of Scheduling*, 2(2):79–98, 1999.

[182] D. Whitley, T. Starkweather, and D. Shaner. The travelling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 350–372. Van Nostrand Reinhold, New York, NY, USA, 1991.

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

[184] M. Yokoyama and H. W. Lewis III. Optimization of the stochastic dynamic production problem by a genetic algorithm. *Computers & Operations Research*, 30:1831–1849, 2003.

[185] Y. Yoshitomi. A genetic algorithm approach to solving stochastic job-shop scheduling problems. *International Transactions in Operational Research*, 9(4):479–495, 2002.

[186] Y. Yoshitomi and R. Yamaguchi. A genetic algorithm and the Monte Carlo method for stochastic job-shop scheduling. *International Transactions in Operational Research*, 10(6):577–596, 2003.

[187] H. J. Zimmermann. *Fuzzy Set Theory and its Application*. Kluwer Academic Publishers, Boston, MA, USA, 2nd edition, 1991.