Université Libre de Bruxelles
Faculté des Sciences Appliquées
CoDE, Department of Computer & Decision Engineering
IRIDIA, *Institut de Recherches Interdisciplinaires*
et de Développements en Intelligence Artificielle

Estimation-based Metaheuristics for Stochastic Combinatorial Optimization:

Case Studies in Stochastic Routing Problems

Prasanna BALAPRAKASH

Promoteur de Thèse:

Prof. Marco DORIGO

Co-promoteurs de Thèse:

Dr. Mauro BIRATTARI

Dr. habil. Thomas STÜTZLE

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

Année Académique 2009/2010

Summary

Stochastic combinatorial optimization problems are combinatorial optimization problems where part of the problem data are probabilistic. The focus of this thesis is on stochastic routing problems, a class of stochastic combinatorial optimization problems that arise in distribution management. Stochastic routing problems involve finding the best solution to distribute goods across a logistic network. In the problems we tackle, we consider a setting in which the cost of a solution is described by a random variable; the goal is to find the solution that minimizes the expected cost. Solving such stochastic routing problems is a challenging task because of two main factors. First, the number of possible solutions grows exponentially with the instance size. Second, computing the expected cost of a solution is computationally very expensive.

To tackle stochastic routing problems, stochastic local search algorithms such as iterative improvement algorithms and metaheuristics are quite promising because they offer effective strategies to tackle the combinatorial nature of these problems. However, a crucial factor that determines the success of these algorithms in stochastic settings is the trade-off between the computation time needed to search for high quality solutions in a large search space and the computation time spent in computing the expected cost of solutions obtained during the search.

To compute the expected cost of solutions in stochastic routing problems, two classes of approaches have been proposed in the literature: analytical computation and empirical estimation. The former exactly computes the expected cost using closed-form expressions; the latter estimates the expected cost through Monte Carlo simulation.

Many previously proposed metaheuristics for stochastic routing problems use the analytical computation approach. However, in a large number of practical stochastic routing problems, due to the presence of complex constraints, the use of the analytical computation approach is difficult, time consuming or even impossible. Even for the prototypical stochastic routing problems that we consider in this thesis, the adoption of the analytical computation approach is computationally expensive. Notwithstanding

the fact that the empirical estimation approach can address the issues posed by the analytical computation approach, its adoption in metaheuristics to tackle stochastic routing problems has never been thoroughly investigated.

In this thesis, we study two classical stochastic routing problems: the probabilistic traveling salesman problem (PTSP) and the vehicle routing problem with stochastic demands and customers (VRPSDC). The goal of the thesis is to design, implement, and analyze effective metaheuristics that use the empirical estimation approach to tackle these two problems.

The main results of this thesis are:

- The empirical estimation approach is a viable alternative to the widely-adopted analytical computation approach for the PTSP and the VRPSDC.
- A principled adoption of the empirical estimation approach in metaheuristics results in high performing algorithms for tackling the PTSP and the VRPSDC. The estimation-based metaheuristics developed in this thesis for these two problems define the new state-of-the-art.

To my parents and sisters...

Acknowledgements

The research work presented in this thesis would have taken far longer or even would not have been possible to complete without three fantastic scientists: Prof. Marco Dorigo, Dr. Mauro Birattari, and Dr. Thomas Stützle. I feel extremely lucky and privileged to be supervised by them. Their supervision had a major influence not only on this thesis but also on my personal life.

First, I wish to express my deep gratitude to my supervisor, Prof. Marco Dorigo for giving me the opportunity to work at IRIDIA, where I had all the necessary support, and even more, to do research. I always admire his energy level and he has been a continuous source of inspiration. His guidance and insight have helped to shape this thesis into what it is. I greatly appreciate his careful proof reading of the draft of the articles and the final thesis manuscript in spite of his busy schedule. Special thanks to his constant advices that helped me to improve my research, presentation, and communication skills.

I am indebted to my co-supervisor, Dr. Mauro Birattari, who played a crucial role for helping me tackle a challenging research that lies behind this thesis. He has been a very supportive mentor. He shaped me as a researcher and I learned the essential tools to perform research from him. He constantly pushed me beyond what I thought I could do. He inspired me to perform serious research and extracted good ideas out of me. He taught me how to write academic articles and made me a better programmer. He was always there to meet and talk about my ideas and to ask me good questions that helped me to dive deep into my problems.

I am grateful to my co-supervisor, Dr. Thomas Stützle, who has always been offering continual support that stimulated my research over the past five years. He made me to think big! He infected me with his passion for developing state-of-the-art algorithms. He shared his expertise and research insight that enabled me to enjoy the beauty of details in stochastic local search algorithms. His critical thought, support, and encouragement catalyzed my publication activities. His ability to rapidly assess the quality of ideas is awesome! Without his advise, I would have definitely wasted several months during my research. What I learned from him is probably just a drop—he has an ocean to offer. For that matter, working under him for a period of five years is too short and I guess even a life time is not sufficient!

In a nutshell, I could not have wished for better supervisor and co-supervisors than Prof. Marco Dorigo, Dr. Mauro Birattari, and Dr. Thomas Stützle. They were instrumental in each and every step of my research project. I also wish to express my

appreciation to Prof. Hugues Bersini, co-director of IRIDIA with Prof. Marco Dorigo, for contributing to make of IRIDIA such an enjoyable and friendly research lab.

In fact, IRIDIA is a fantastic place—not only to do research but also to make new friends from different parts of the world. I throughly enjoyed the international atmosphere of IRIDIA. I would like to say tons of thanks to my friends at IRIDIA for their friendship and for being my surrogate family: Marco Montes de Oca, Max Manfrin, Ali Emre Turgut, Nithin Mathews, Arne Brutschy, Alexandre Campo, Eliseo Ferrante, Rehan O'Grady, Sabrina Oliveira, Carlo Pinciroli, Giovanni Pini, Yuan Zhi, Carlotta Piscopo, Francisco Santos, Renaud Lenne, Bruno Marchal, Jérémie Dubois-Lacoste, Mohamed Saifullah, Manuele Brambilla, Antal Decugnière, Tom Lenaerts, Elio Tuci, Christos Ampatzis, Anders Christensen, Roderich Groß, Thomas Halva Labella, Shervin Nouyan, Krzysztof Socha, Vito Trianni, Giacomo Di Tollo, Matteo Gagliolo, Álvaro Gutiérrez, Jodelson Sabino, Cristina Teixeira, Thijs Urlings, Levent Bayindir, Jessica Rivero, Navneet Bhalla, Paola Pellegrini, Colin Twomey, Francesco Sambo, Christophe Philemotte, Amin Mantrach, Utku Salihoglu, and Manuel López-Ibáñez. It was a pleasure to work in a lab with such smart and passionate people and to benefit from their knowledge. I thank Ali Emre Turgut, Max Manfrin, Eliseo Ferrante, and Mohamed Saifullah for proof reading several chapters of the thesis. Special thanks to Muriel Decreton, who rescued me from several red-tape crisis.

My special appreciation goes to my parents, Balaprakash and Sakunthala, for their unconditional support, love, and encouragement to pursue my interests. They are the primary source of my energy. I am very much indebted to my beloved sisters Siva Sankari and Deepa Aishwarya for their love, affection, and care. I would also like to thank Lakshmi, Arthi, Prabhu, and Arun for their friendship. Special thanks to Heleen for her emotional support.

I thank Dr. Leonora Bianchi and Prof. Ann Melissa Campbell for providing the source code of pACS+1-shift and the aggregation approach, respectively.

My research was supported by COMP^{2}SYS, an Early Stage Training project funded by the European Commission within the Marie Curie Actions program (MEST-CT-2004-505079), and by ANTS and META-X, two ARC projects funded by the French Community of Belgium. Since 2006, my research has been supported by the fund for scientific research F.R.S.-FNRS of the French Community of Belgium.

P. B.

Brussels, Belgium
December 8, 2009

Contents

Chapter 1

Introduction

Combinatorial optimization involves finding the best solution from a large number of possible solutions to a given problem. Numerous problems arising in production planning, distribution management, internet routing, business administration, economic systems, marketing strategies, and investment planning are combinatorial optimization problems (Hoos and Stützle, 2005). Combinatorial optimization problems are conceptually easy to model but most of them are quite difficult to solve in practice. The primary difficulty in solving combinatorial optimization problems arises from the fact that the number of possible solutions from which the best needs to be selected grows exponentially with the size of the problem instances. For example, consider a classical combinatorial optimization problem, the traveling salesman problem (TSP) that involves finding a shortest Hamiltonian cycle[1] in a given weighted graph. An instance of the problem with 10 nodes has 1.81×10^5 possible solutions: enumerating all of them and choosing the best one is feasible in a short computation time. However, an instance of 100 nodes already has 4.66×10^{155} solutions: in this case, enumerating all the possible solutions needs a prohibitively large computation time.

Algorithms available for solving combinatorial optimization problems fall into two main classes: exact methods and approximate algorithms. Exact methods are guaranteed to find an optimal solution and prove that it is actually optimal or show that no feasible solution exists. However, the computation time for these methods for many problems increases exponentially with respect to the problem size, and often only small or moderately sized problem instances can be practically solved. Approximate algorithms sacrifice the guarantee of finding optimal solution for heuristically searching

[1] An Hamiltonian cycle is a path that visits each node exactly once and returns to the starting node.

high quality solutions in short computation times. Important classes of approximate algorithms are constructive heuristics, iterative improvement algorithms, and metaheuristics. Heuristics are simple problem-specific algorithms based on an algorithm designers' intuition or some rules of thumb. Iterative improvement algorithms start from some initial solution and repeatedly try to move from a current solution to a lower cost neighboring one. Metaheuristics are high-level strategies which typically guide problem-specific heuristics or iterative improvement algorithms, to increase their performance. Many approximate algorithms rely on intelligently biased probabilistic choices during the search and they are referred to as stochastic local search (SLS) algorithms (Hoos and Stützle, 2005).

Metaheuristics are among the most powerful techniques for solving large scale combinatorial problems. They are currently established as state-of-the-art algorithms for many real-world problems in which obtaining high quality solutions in relatively short computation time is of primary importance. The effectiveness of the metaheuristics can be illustrated with the following example. To solve a TSP instance with 15112 nodes, the state-of-the-art exact algorithm required 22.6 CPU years on a Compaq EV6 Alpha processor running at 500 MHz. On the same TSP instance, a high performing metaheuristic needed only 7 CPU seconds to find a solution whose cost is 1% away from the cost of the optimal solution (Hoos and Stützle, 2005). Due to the enormous significance of combinatorial optimization problems for the academic as well as the industrial world, the field of the design, implementation, and study of metaheuristics is a very active area of research.

In a large number of combinatorial optimization problems emerging from science, business, and engineering, the parameters that define problem instances are affected by uncertainty. Examples include investment decisions in portfolio management, vehicle routing, resource allocation, job scheduling, and modeling and simulation of large molecular systems in bioinformatics. In the presence of data uncertainty, combinatorial optimization problems lead to stochastic combinatorial optimization problems. The focus of this thesis is on stochastic routing problems, a class of stochastic combinatorial optimization problems that arise in distribution management. These problems involve finding a cost-effective way to distribute or to collect goods across a logistic network. Entities that distribute goods are referred to as vehicles or salesmen and entities that demand goods are referred to as customers. In stochastic routing problems, customers visits, theirs demands, or vehicle travel times are affected by uncertainty. This uncertainty is characterized through a given probability distribution. The introduction of probabilistic elements into the routing problem increases dramatically the difficulty of

the problem and, as a result, solving them is an even more challenging task than for their deterministic counterparts.

Designing efficient algorithms for solving stochastic routing problems is a hard task and it is perhaps one of the most challenging tasks facing the computer science and operations research communities. The difficulty is due to two factors:

- The number of possible solutions grows exponentially with instance size as in combinatorial optimization problems.
- Evaluating the quality of solutions under uncertainty is notoriously difficult and computationally more expensive than in the deterministic case.

Metaheuristics are quite attractive as solution techniques to tackle stochastic combinatorial optimization problems because they provide well established strategies to tackle the combinatorial nature of these problems. However, a crucial issue becomes the computation time trade-off between the search and the solution quality evaluation under uncertainty.

One of the most effective strategies to solve stochastic routing problems is *a priori* optimization. In this strategy, an *a priori* solution is decided in advance and followed by a salesman/vehicle every day. On a given day, the salesman/vehicle follows the *a priori* solution. When the salesman/vehicle fails to meet the demand of a customer, a recourse action is performed. This recourse action could be in the form of skipping a customer who does not require being visited, going back to the depot to replenish, or any other problem-specific recourse action. The solution obtained by performing the recourse action on the *a priori* solution is the *a posteriori* solution. The goal then becomes finding an *a priori* solution that minimizes the expected cost of the associated *a posteriori* solution. To compute the expected cost in *a priori* optimization, two approaches have been discussed in the literature: *analytical computation* and *empirical estimation*. The former exactly computes the expected cost of the *a posteriori* solutions using a complex analytical development. The latter estimates the expected cost through Monte Carlo simulation.

Exact methods can solve small sized stochastic routing problems to optimality (Gendreau et al., 1996a). Recent and pragmatic approaches to tackle stochastic routing problems mainly involve the application of SLS methods, in particular iterative improvement algorithms and metaheuristics. In this thesis, we tackle two classical stochastic routing problems namely, the probabilistic traveling salesman problem (PTSP) and the vehicle routing problem with stochastic demands and customers (VRPSDC). These problems are academic variants of many real world stochastic routing problems (Jaillet, 1985,

1987; Bertsimas, 1988; Gendreau et al., 1996a). The core of this thesis is on the application of iterative improvement algorithms and metaheuristics for the solution of the PTSP and the VRPSDC.

The PTSP is a central problem in stochastic routing and has a number of practical applications not only in transportation but also in strategic planning and scheduling. In the PTSP, each customer has a probability of requiring being visited. The *a priori* solution must be found prior to knowing which customers are to be visited; the associated *a posteriori* solution, which is computed *after* knowing which customers need to be visited, is obtained by visiting them in the order prescribed by the *a priori* solution but skipping the customers who do not require being visited.

The VRPSDC is concerned with minimizing the cost involved in routing a vehicle with a limited capacity that collects goods from a number of customers, where each customer has a probability of requiring being visited and a stochastic demand. This problem models a number of practical problems in the areas of truckload operations and package delivery systems. The *a priori* solution is a sequence of all customers, which is decided before knowing customers presence and their demands; the associated *a posteriori* solution, which is computed after knowing customers presence and their demands, is obtained by following the *a priori* solution but with the following recourse actions: a customer who does not require being visited is skipped; whenever the vehicle fails to meet the demand of a customer, it has to go back to the depot for unloading.

Motivation and goal

Before the start of the work towards this thesis, the majority of iterative improvement algorithms and metaheuristics designed for stochastic routing problems were based on the analytical computation approach. In particular, for the PTSP and the VRPSDC, all algorithms used the analytical computation approach. There are two crucial drawbacks of this approach. Firstly, evaluating the cost of a solution using the analytical computation approach is computationally expensive; thus, the full potential of the iterative improvement algorithms and metaheuristics cannot be obtained. Secondly, the analytical computation approach is not general-purpose and it cannot be applied to complex problems, in which the cost computation cannot be derived in an analytical way. In spite of the fact that the empirical estimation approach has the potential to overcome the difficulty posed by analytical computation, surprisingly, to the best of our knowledge, its adoption in iterative improvement algorithms and metaheuristics was still, at the time we started the work presented in this thesis, in an early stage and

it had never been thoroughly investigated before.
The goal of the thesis is to design, implement, and analyze effective estimation-based iterative improvement algorithms and metaheuristics to tackle the PTSP and the VRPSDC.

1.1 Original contributions

The main contributions of the thesis are listed in the following.

Estimation-based algorithms for tackling stochastic routing problems

This thesis provides a thorough investigation of using the empirical estimation approach in iterative improvement algorithms and metaheuristics for the PTSP and the VRPSDC. On the results side, the thesis presents extensive experimental evidence to show that the empirical estimation approach is an effective alternative to the widely-adopted analytical computation approach.

Estimation-based iterative improvement algorithm for the PTSP

For the PTSP, previous state-of-the-art iterative improvement algorithms used delta evaluation based on the analytical computation approach: the cost difference between two neighboring solutions was computed by considering the cost contribution of solution components that are not common to them; this cost difference is given by recursive closed-form expressions.

The thesis proposes an effective estimation-based iterative improvement algorithm for the PTSP. The main novelty of the proposed algorithm consists of using an empirical estimation approach and designing an effective data structure for the PTSP delta evaluation. Inspired by iterative improvement algorithms proposed for the closely related TSP, particular attention is devoted to the adoption of neighborhood reduction techniques. A systematic experimental analysis is carried out to assess the effectiveness of each algorithmic component adopted in the estimation-based iterative improvement algorithm.

Adaptive sample size and variance reduction techniques in delta evaluation for the PTSP

The use of an adaptive sample size procedure offers computational benefit by prescribing the most appropriate number of realizations needed for cost estimation. Variance

reduction techniques are used to reduce the number of realizations needed for obtaining precise cost estimates. The thesis proposes a new way of combining two variance reduction techniques, namely, the method of common random numbers, importance sampling, and an adaptive sample size procedure for the PTSP delta evaluation.

Estimation-based metaheuristics for the PTSP

For the PTSP, the estimation-based metaheuristics proposed so far in the literature are prototypical algorithms. The thesis discusses three high performing estimation-based metaheuristics, namely, iterated local search, memetic algorithms, and ant colony optimization to solve the PTSP. These algorithms use an estimation approach to evaluate the solution cost and exploit the estimation-based iterative improvement algorithm as subsidiary solution improvement procedure. A particularity of the estimation approach is the adoption of the method of common random numbers and of an adaptive sample size procedure. The parameters of all the estimation-based algorithms are rigorously fine-tuned and tested on instances with different characteristics.

Estimation-based metaheuristics for VRPSDC

So far, the VRPSDC has been tackled by an analytical computation algorithm that adopts a computationally expensive closed-form expression. However, for large instances with several hundreds of nodes, high computational cost affects the performance of the algorithm considerably. For the first time, an estimation-based approach is adopted within the metaheuristics framework to tackle the VRPSDC and it is shown to be more effective than the previously proposed analytical computation approach.

Advancement of the state-of-the-art for the PTSP and the VRPSDC

The estimation-based algorithms proposed in the thesis are compared to the previously proposed algorithms using an accurately designed and statistically sound experimental methodology. Primary importance is given for assessing the performance of the algorithms on large instances. Our estimation-based iterative improvement algorithm proposed for the PTSP completely outperforms previous iterative improvement algorithms. Our estimation-based metaheuristics for the PTSP are more effective than the previous best performing metaheuristics. Our estimation-based metaheuristics developed for the VRPSDC define the new state-of-the-art.

Engineering algorithms for stochastic routing problems

The thesis follows a principled engineering approach to develop high performing algorithms for stochastic routing problems. This approach is a bottom-up process. In order to tackle the PTSP and the VRPSDC, the process is strongly focused on the development and refinement of the PTSP iterative improvement algorithm. Although not always necessary in other stochastic routing problems, the strong focus on the iterative improvement algorithm played a crucial role in attaining high performing algorithms for the PTSP and the VRPSDC.

Iterative F-race for parameter tuning

While typically parameters of iterative improvement algorithms and metaheuristics are tuned by hand, recent studies have shown that automatic tuning procedures can effectively handle this task and often find better parameter values than that of the widely used ad-hoc tuning methods. `F-Race` is a successful automatic tuning procedure that has proven to be very effective in a number of tuning tasks.

The thesis introduces the Iterative F-Race algorithm for tuning the configuration of metaheuristics. Iterative F-Race is an improved variant of `F-Race` that on the one hand makes it suitable for tuning tasks with a large number of initial parameter values and, on the other hand allows a significant reduction of the computation time needed for tuning tasks without any major loss in solution quality. This algorithm is used to fine tune the parameter values of all estimation-based algorithms developed for the PTSP and the VRPSDC.

1.2 Scientific publications in connection with this thesis

Several chapters presented in this thesis are based on articles that the author, together with co-authors, has published or submitted for publication to journals, workshops, and conferences.

The design and development of new iterative improvement algorithms that use an estimation-based delta evaluation for the PTSP, described in Chapter 4, is based on the following journal and conference articles:

> M. Birattari, P. Balaprakash, T. Stützle, and M. Dorigo. Estimation-based local search for stochastic combinatorial optimization using delta evaluations: A case study in the probabilistic traveling salesman problem. *INFORMS Journal on Computing*, 20(4):644–658, 2008.

M. Birattari, P. Balaprakash, T. Stützle, and M. Dorigo. Estimation-based local search for the probabilistic traveling salesman problem. In M. Gendreau et. al. (Eds.) MIC 2007: The Seventh Metaheuristics International Conference. Montreal, Canada. June 25-29, 2007.

The integration of the adaptive sample size and the importance sampling procedures in the estimation-based iterative improvement algorithm for the PTSP, given in Chapter 5, is published in a journal article and has been disseminated in a workshop.

P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Adaptive sample size and importance sampling in estimation-based local search for the probabilistic traveling salesman problem. *European Journal of Operational Research*, 199(1):98–110, 2009.

P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Sampling strategies and local search for stochastic combinatorial optimization. In E. Ridge et. al. (Eds.) SLS-DS 2007: Doctoral Symposium on Engineering Stochastic Local Search Algorithms, pp. 16-20, September 6-8, 2007, Brussels, Belgium.

The bottom-up engineering methodology that we followed in Chapter 4 and Chapter 5 for deriving the estimation-based iterative improvement algorithm for the PTSP has been published as the following book chapter:

P. Balaprakash, M. Birattari, and T. Stützle. Engineering stochastic local search algorithms: A case study in estimation-based local search for the probabilistic travelling salesman problem. In C. Cotta and J. van Hemert, editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of Studies in Computational Intelligence, pages 53–66, Berlin, Germany, 2008. Springer Verlag.

The systematic study of ant colony optimization algorithms for tackling the PTSP, given in Appendix A, is contained in the following journal article. Moreover, the initial experimental investigations are published as a book chapter and as conference article:

P. Balaprakash, M. Birattari, T. Stützle, Z. Yuan, and M. Dorigo. Estimation-based ant colony optimization and local search for the probabilistic traveling salesman problem. *Swarm Intelligence*, 3(3):223–242, 2009.

M. Birattari, P. Balaprakash, and M. Dorigo. The ACO/F-RACE algorithm for combinatorial optimization under uncertainty. In K. F. Doerner et. al. (Eds.), *Metaheuristics - Progress in Complex Systems Optimization*, Operations Research/Computer Science Interfaces Series, pages 189–203, Berlin, Germany, 2006.

M. Birattari, P. Balaprakash, and M. Dorigo. ACO/F-Race: Ant colony optimization and racing techniques for combinatorial optimization under uncertainty, In R. F. Hartl et. al. (Eds.) MIC 2005: The sixth Metaheuristics International Conference, August 22-26, 2005, Vienna, Austria.

An article on the estimation-based metaheuristics developed for the PTSP, described in Chapter 6, is submitted to a journal. The preliminary work on this topic has been the subject of three short conference articles.

P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Estimation-based metaheuristics for the probabilistic traveling salesman problem. *Computers and Operations Research*, (under review).

P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. An experimental study of estimation-based metaheuristics for the probabilistic traveling salesman problem. In V. Maniezzo et. al. (Eds.) LION 2007 II: Learning and Intelligent Optimization, December 8-12, 2007, Trento, Italy.

P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Estimation-based stochastic local search algorithms for the stochastic routing problems. In E.-G. Talbi and K. Mellouli (Eds.) International Conference on Metaheuristics and Nature Inspired Computing, META'08, October 29-31, 2008, Hammamet, Tunisia.

P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Applications of estimation-based SLS algorithms to stochastic routing problems. In P. Hansen and S. Voss (Eds.) Metaheuristics 2008, Second International Workshop on Model Based Metaheuristics, June 16-18, 2008, Bertinoro, Italy.

The extension of the PTSP estimation-based metaheuristics to tackle the VRPSDC, described in Chapter 7, forms the basis of a planned journal article:

P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Estimation-based metaheuristics for the vehicle routing problem with stochastic demands and customers. (Completed as of October 2009 and to be submitted).

The design and analysis of Iterative F-race, which is used to fine-tune the parameter values of all the estimation-based algorithms described in Appendix B, has been published as a workshop article, a book chapter, and a short conference article:

P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In T. Bartz-Beielstein et. al. (Eds.), *Hybrid Metaheuristics*, volume 4771 of LNCS, pages 113–127, Berlin, Germany, 2007. Springer Verlag.

M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-Race and iterated F-Race: An overview of racing algorithms for algorithm tuning and design. In T. Bartz-Beielstein et. al. (Eds.), *Empirical Methods for the Analysis of Optimization Algorithms*, Natural Computing Series, Germany, 2009. Springer Verlag. (In press)

M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle. Automated algorithm tuning using F-races: Recent developments. In S. Voss and M. Caserta (Eds.) MIC 2009: The eighth Metaheuristics International Conference, July 13-16, 2009, Hamburg, Germany.

A feasibility study of adopting the estimation-based PTSP iterative improvement algorithm to tackle the index tracking problem in portfolio selection has been presented in the following workshop:

G. di Tollo and P. Balaprakash. Index tracking by estimation-based local search. In A. Amendola et. al. (Eds.) 2nd International Workshop on Computational and Financial Econometrics, CFE'08, June 19-21, 2008, Neuchatel, Switzerland.

1.3 Organization of the thesis

The thesis continues in Chapter 2, which is divided into two parts. The first part introduces combinatorial optimization and gives a brief summary of some widely used SLS methods. The second part presents stochastic combinatorial optimization problems, their difficulty, followed by a comprehensive review of how metaheuristics tackle

stochastic combinatorial optimization problems. Several techniques described here serve as a basis for designing effective algorithms for the PTSP and the VRPSDC.

Chapter 3 provides a survey of stochastic routing problems. A particular emphasis is given to the PTSP and the VRPSDC. For each problem, we cover exact methods, constructive heuristics, iterative improvement algorithms, and metaheuristics proposed in the literature.

Chapter 4 introduces `2.5-opt-EEs`, a new estimation-based iterative improvement algorithm for the PTSP. We carry out a systematic experimental comparison to show that the adoption of TSP-specific neighborhoods and an estimation-based approach for evaluating solution costs allow the algorithm to find high quality solutions in a very short computation time. We also compare the proposed `2.5-opt-EEs` algorithm to iterative improvement algorithms, which were previously proposed in the literature, on large instances and on other new random instances.

Chapter 5 presents an improved variant of `2.5-opt-EEs` called `2.5-opt-EEais`. This improvement is obtained by using the adaptive sample size and of importance sampling procedures. An extensive experimental analysis to assess the contribution of the adaptive sample size and of the importance sampling procedures on computation time and solution quality is presented.

Chapter 6 focuses on metaheuristics to tackle the PTSP. We engineer three metaheuristics namely, iterated local search, memetic algorithms, and ant colony optimization, which are known to have very good performance on the deterministic TSP. This process consists in adopting the estimation approach to evaluate the solution cost, exploiting `2.5-opt-EEais` as local search, and tuning the metaheuristics parameters. A random restart local search is also used as a control algorithm. We present a rigorous experimental comparison between the estimation-based algorithms on a wide range of instances. We evaluate the effectiveness of the proposed algorithms against the so far best performing metaheuristics for the PTSP.

Chapter 7 is concerned with the extension of the estimation-based metaheuristics to the VRPSDC. The proposed extension comprises the customization of the PTSP cost evaluation procedure to take into account the stochastic demands. The iterative improvement algorithm `2.5-opt-EEais` developed for the PTSP is used as a local search inside all metaheuristics. The best performing metaheuristic for the VRPSDC is re-implemented and used for an experimental comparison. We present a systematic experimental analysis to study the effectiveness of adopting the analytical computation approach and several variants of the empirical estimation approach to the VRPSDC. Then, we study the effectiveness of the proposed algorithms on large instances.

Chapter 8 concludes the thesis with a summary of the main contributions followed by directions for future research.

The thesis comprises two appendices (A and B), which are also part of the contributions. Appendix A, a supplement to Chapter 6, presents a systematic experimental study of several ant colony optimization algorithms for the PTSP. The estimation-based ant colony optimization algorithm used in Chapter 6 is chosen based on the results of appendix A. Appendix B describes the Iterative F-Race algorithm, which is used to fine-tune the parameter values of the estimation-based iterative improvement algorithm `2.5-opt-EEais`, given in Chapter 5, and all estimation-based metaheuristics developed for the PTSP and the VRPSDC discussed in Chapters 6, 7, and Appendix A.

Chapter 2

Background

In recent years, much attention has been devoted to the development of metaheuristics for tackling stochastic combinatorial optimization problems. In this chapter, we provide an overview of recent developments. As one may expect, many algorithms developed for stochastic combinatorial optimization problems are extensions of those designed for their deterministic counterparts. In Section 2.1, we provide some background knowledge about combinatorial optimization and we give a brief summary of some widely used metaheuristics designed to solve such problems. Then, in Section 2.2, we introduce stochastic combinatorial optimization problems and explain why designing effective algorithms for these problems is more difficult than for their deterministic counterparts. Finally, we discuss metaheuristics developed to tackle stochastic combinatorial optimization problems.

2.1 Combinatorial optimization and metaheuristics

According to Papadimitriou and Steiglitz (1982), a combinatorial optimization problem $\mathcal{C} = (S, f)$ is an optimization problem that comprises a finite set S of solutions and a cost function $f : \mathcal{S} \rightarrow \Re^+$ that assigns a positive cost value to each solution $s \in \mathcal{S}$. The goal is to find a solution x^* with minimum cost[1]. A solution x^* is called an optimal solution or a global optimum of the given problem $\mathcal{C}$. The solutions are typically permutations of numbers, subsets of a set of items or a graph structure. Combinatorial optimization problems are frequently encountered in areas such as planning, scheduling, time-tabling, and resource allocation, where there are many possible solutions to consider and the

[1]Note that maximizing over a cost function f is the same as minimizing over $-f$. Therefore, it is possible to describe every combinatorial optimization problem as a minimization problem.

goal is to determine the best solution.

A paradigmatic example of a combinatorial optimization problem is the traveling salesman problem (TSP). It is defined on a complete graph $G = \{V, A, C\}$ with $V = \{1, 2, ..., n\}$ being a set of nodes, $A = \{\langle i, j\rangle : i, j \in V, i \neq j\}$ being a set of edges, where an edge $\langle i, j\rangle$ connects nodes i and j, and $C = \{c_{ij} : \langle i, j\rangle \in A\}$ being a set of costs, where c_{ij} is the cost of traversing an edge $\langle i, j\rangle$. Typically, the costs are assumed to be symmetric, that is, for all pairs of nodes i, j we have $c_{ij} = c_{ji}$. The goal is to find the minimum cost Hamiltonian cycle (a complete tour) in G. The set S consists of all possible Hamiltonian cycles in G.

A combinatorial optimization problem comprises a set of problem instances. The terms *problem* and *instance* are distinguished as follows. The former refers to the general question to be answered, typically without specifying the values for the problem parameters—for the TSP, it refers to the general problem of finding a minimum cost Hamiltonian cycle in G. The latter refers to a problem with specific values for all the parameters of the problem—for the TSP, it refers to a specific problem, where the number of nodes and all edge costs are defined.

A naive method to solve a combinatorial optimization problem is exhaustive search that consists in enumerating all possible solutions to find the one with minimum cost. This method becomes rapidly infeasible because the number of possible solutions grows exponentially with instance size. For some combinatorial optimization problems, the exploitation of problem-specific knowledge allows the definition of algorithms that find an optimal solution much faster than exhaustive search. However, for many problems, even the best algorithms of this kind cannot do much better than exhaustive search.

When solving combinatorial problems, it is important to assess the difficulty of solving them. An important theory that describes this difficulty is the theory of $\mathcal{NP}$-completeness. Using this theory, the difficulty of combinatorial optimization problems is investigated in two steps. In the first place, it deals with the computational complexity of solving decision problems including decision variants of optimization problems. The decision variants of an optimization problem generally ask the question (in the minimization case) whether for a given problem instance there exists a solution, which is better than some value. The decision problems are classified as $\mathcal{P}$ or $\mathcal{NP}$ according to the time complexity function[2] of the best available algorithm to solve it. The decision

[2]The time complexity function $g(n)$ of an algorithm indicates for each possible input size n, the maximum amount of time needed by the algorithm to solve an instance of that size. The function $g(n)$ is $\mathcal{O}(h(n))$ if there exist integers c and N such that $|g(n)| \leq c \cdot |h(n)|$ for all values of $n \geq N$. If an algorithm has time complexity function $\mathcal{O}(h(n))$ for some polynomial function h, it is referred to as a polynomial time algorithm. If the time complexity function of an algorithm cannot be bounded by a

problems for which an algorithm exists that outputs in polynomial time the correct answer (either *yes* or *no*) belong to the class $\mathcal{P}$; the decision problems for which one can verify the correctness of a given answer in polynomial time, belong to the class $\mathcal{NP}$. Although it is clear that $\mathcal{P} \subseteq \mathcal{NP}$, the question of whether $\mathcal{P}$ is equal to $\mathcal{NP}$ or not remains unanswered so far. However, it is widely conjectured that $\mathcal{P} \neq \mathcal{NP}$ (Fortnow, 2009). Note that proving $\mathcal{P} = \mathcal{NP}$ implies that all problems in $\mathcal{NP}$ can be solved in polynomial time.

A problem is $\mathcal{NP}$-hard, if every other problem in $\mathcal{NP}$ can be transformed to it by the so-called polynomial time reduction, a procedure that transforms a problem into another one by a polynomial time algorithm. It should be noted that $\mathcal{NP}$-hard problems do not necessarily belong to the class $\mathcal{NP}$ themselves, since their time complexity may actually be much higher or, simply, they may not be decision but optimization problems. $\mathcal{NP}$-hard problems that are also in $\mathcal{NP}$ are said to be $\mathcal{NP}$-complete. Since solving the original optimization problem is at least as hard as solving its decision version, proving the intractability of the latter implies intractability of the former. Many combinatorial optimization problems are $\mathcal{NP}$-hard problems, which means that in the worst case the computation time needed to find the optimal solution increases exponentially with the size of the problem instance. We refer the reader to Garey and Johnson (1979), Papadimitriou and Steiglitz (1982), Papadimitriou (1994) for a detailed explanation of the computational complexity of combinatorial optimization problems.

The solution techniques developed to tackle combinatorial optimization problems can be categorized into two groups: exact methods and approximate algorithms. Exact methods are guaranteed to find the optimal solution for every finite size instance of a combinatorial optimization problem within an instance-dependent computation time. However, for $\mathcal{NP}$-hard combinatorial optimization problems, this computation time cannot be bounded by a polynomial time. In other words, exact methods have an exponential worst case time complexity. More importantly, not only in theory, but also in practice, exact methods often cannot solve instances of even moderate size. Approximate algorithms sacrifice the guarantee of finding the optimal solution for finding good solutions in a reasonable amount of computation time. These algorithms, when correctly used, are state-of-the-art for many combinatorial optimization problems.

From an algorithmic perspective, approximate algorithms can be classified as constructive heuristics, iterative improvement algorithms, and metaheuristics. Since most of the approximate algorithms make use of randomized choices in generating or selecting solutions for combinatorial optimization problems, they are also referred to as

polynomial, it is referred to as an exponential time algorithm.

stochastic local search (SLS) methods (Hoos and Stützle, 2005). In the rest of this section, we give an overview of SLS methods with a primary focus on metaheuristics. For a comprehensive coverage of approximate algorithms, we refer the reader to Aarts and Lenstra (1997), Glover and Kochenberger (2002), and Hoos and Stützle (2005).

Constructive heuristics

Constructive heuristics start from an initial empty solution and iteratively add solution components to a partial solution until a complete solution is obtained. These algorithms are usually fast but the quality of the solutions they obtain is typically worse than that of iterative improvement algorithms and metaheuristics. Therefore, they are often not used as standalone algorithms to tackle combinatorial optimization problems. Instead, they are primarily used to generate good initial solutions for iterative improvement algorithms and metaheuristics.

A popular constructive heuristic for the TSP is the nearest neighbor heuristic. It is a greedy algorithm that starts from a randomly chosen initial node and iteratively adds an edge connecting the current node to a closest neighboring node that has not yet been added. Another heuristic that we use in this thesis is the farthest insertion heuristic. It starts with the edge of maximal cost and iteratively adds the remaining nodes in the following way. Among all nodes not in the partial solution, it chooses the one that is farthest to a node in the partial solution; the chosen node is then inserted between two consecutive nodes in the partial solution in such a way that the insertion causes the smallest increase in the cost of the current partial solution.

Iterative improvement algorithms

Iterative improvement algorithms start from some initial solution and repeatedly try to move from a current solution x to a lower cost neighbor solution x'. Which solutions are neighbors is defined by a neighborhood structure $\mathcal{N} : S \mapsto 2^x$ that assigns to every $x \in S$, a set of neighbor solutions $\mathcal{N}(x) \subseteq S$. The choice of an appropriate neighborhood structure is crucial for the effectiveness of an iterative improvement algorithm. Typically, the choice is problem-specific. A well-known neighborhood structure for combinatorial optimization problems is the k-exchange neighborhood, in which a set of neighbor solutions of x are obtained by exchanging k solution components from x. An iterative improvement algorithm terminates when the current solution x does not have any lower cost neighbor solution, that is, $\forall x' \in \mathcal{N}(x) : f(x) \leq f(x')$. Iterative improvement algorithms are widely referred to as local search algorithms because they

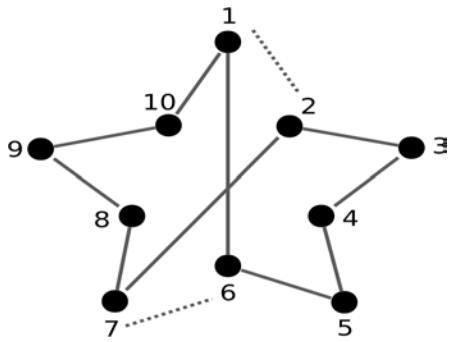


Figure 2.1: A 2-exchange neighbor solution x' that is obtained by deleting two edges $\langle 1, 2\rangle$ and $\langle 6, 7\rangle$ (indicated in dotted lines) of a solution x and by replacing them with $\langle 1, 6\rangle$ and $\langle 2, 7\rangle$.

search in a user-defined neighborhood structure. The final solution obtained by an iterative improvement algorithm is called local optimum since it is only guaranteed to be optimal with respect to its neighborhoods.

In iterative improvement algorithms, the cost of solutions can be evaluated by a full evaluation that computes the cost of each solution from scratch or a partial evaluation that computes only the cost difference between a particular solution and a neighboring solution. The latter is known as delta evaluation, which is highly profitable in terms of computation time whenever it is feasible.

Typically, iterative improvement algorithms are implemented using a first-improvement or a best-improvement rule. In the former case, an improving move is immediately applied as soon as it is detected; in the latter case, the whole neighborhood is examined and a move that gives the best improvement is chosen.

One of the most widely used iterative improvement algorithms for the TSP is 2-opt local search. It uses a 2-exchange neighborhood structure, in which the neighbor solutions are obtained from the current solution x by exchanging two edges with two other edges. Note that to maintain feasibility of a solution, there is only one possible way to exchange two edges. See Figure 2.1 for an example. The algorithm adopts delta evaluation to compute the cost difference between a current solution x and a neighbor solution x' by considering the cost contribution of the solution components that are not common between x and x'. The cost difference between the neighboring solutions shown in Figure 2.1 is simply given by $c_{1,6} + c_{2,7} - c_{1,2} - c_{6,7}$.

Metaheuristics

Metaheuristics are general algorithmic frameworks, which can be applied to different optimization problems with relatively few modifications to make them adapted to a

specific problem (Metaheuristics Network, 2003). Metaheuristics are among the most effective approaches to tackle combinatorial optimization problems. These algorithms have greatly increased our ability to find high quality solutions to difficult, large, and practically relevant combinatorial optimization problems in a reasonable computation time. For a comprehensive description of metaheuristics, we follow Stützle (1998):

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

Blum and Roli (2003) characterize metaheuristics in the following way:

> Metaheuristics are high level strategies for exploring search spaces by using different methods. Of great importance hereby is that a dynamic balance is given between diversification and intensification. The term diversification generally refers to the exploration of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience. The balance between diversification and intensification is important, on one side to quickly identify regions in the search space with high quality solutions and on the other side not to waste too much time in regions of the search space which are either already explored or which do not provide high quality solutions.

Unlike constructive heuristics that end when a complete solution has been constructed or iterative improvement algorithms that end when a local optimum has been reached, metaheuristics do not have a natural stopping criterion. They use arbitrary stopping criteria such as maximum computation time or number of iterations.

Examples of metaheuristics include simulated annealing (SA) (Černý, 1985; Kirkpatrick et al., 1983), tabu search (TS) (Glover, 1989, 1990; Glover and Laguna, 1997), iterated local search (ILS) (Martin et al., 1991; Lourenço et al., 2002), evolutionary computation (EC) (Rechenberg, 1973; Fogel et al., 1966; Holland, 1975; Goldberg, 1989), ant colony optimization (ACO) (Dorigo, 1992; Dorigo and Stützle, 2004), scatter search (SS) (Glover, 1977), and greedy randomized adaptive search procedures (GRASP) (Feo and Resende, 1989, 1995). These algorithms can be classified in several ways based on a number of criteria (Stützle, 1998). In this section, we describe some of the most widely used metaheuristics that are also most relevant for this thesis. We proceed from those that work on single solutions, to more sophisticated algorithms that work on a population of solutions.

Simulated annealing

SA (Černý, 1985; Kirkpatrick et al., 1983) is inspired by the physical process of annealing. A simple variant of simulated annealing is obtained from the iterative improvement algorithm by adding a mechanism called probabilistic acceptance criterion that can allow the algorithm to move to a worse neighboring solution. This probabilistic acceptance criterion takes into account the cost of x and x' and the value of a parameter T called temperature. Many simulated annealing algorithms adopt the so-called Metropolis acceptance criterion (Metropolis et al., 1953): When x' has a lower cost than x, the algorithm moves to x'; otherwise, the algorithm moves to x' with a probability $e^{|f(x)-f(x')|/T}$. An annealing schedule slowly reduces the value of T during the search steps with the consequence that at the beginning of the search the probability of accepting a worse solution is higher; as the search progresses, this probability decreases. The mechanism of accepting worse solutions during the search helps the algorithm to escape from the basin of attraction of bad quality local optima encountered in the early stages of the search.

Tabu search

TS (Glover, 1989, 1990; Glover and Laguna, 1997) is one of most widely used metaheuristics. A simple tabu search can be obtained from the iterative improvement algorithm with the best improvement rule by adding a short term memory called tabu list. This list is a first-in first-out queue of previously visited solutions or solutions attributes. Typically, the solution components modified in a search step are declared tabu for a fixed number of subsequent search steps. These solution components cannot

be added or removed from the current candidate solution. The tabu list tries to prevent the search returning to a previously visited solution and to avoid (short) cycles. Nevertheless, the tabu condition can be overridden using an aspiration criterion since the forbidding of the solution components that are tabu also excludes previously unvisited solutions from being visited. Often the simple tabu search is enhanced with a number of intensification and diversification strategies to achieve high performance.

Random restart local search

The primary drawback of iterative improvement algorithms is that they often return unsatisfactory solutions when they get trapped in low quality local optima. A straightforward way to address this problem is to restart the local search algorithm a number of times from a new initial solution, which is generated independently of the previously found local optima. The rationale behind this algorithm is that if high quality local optima are uniformly distributed in the search space, by repeating the local search for a sufficiently large number of times, the chance of obtaining one of those high quality local optima will be high. Unfortunately, in many practical applications, high quality local optima are not uniformly distributed in the search space. However, this simple metaheuristic is often used as a baseline metaheuristic for comparing more sophisticated ones.

Iterated local search

ILS (Martin et al., 1991; Lourenço et al., 2002) is based on the rationale that high quality local optima are clustered in the search space. This algorithm consists in a sequence of runs of a local search algorithm. Starting from an initial candidate solution, a local search is applied until a local optimum is found. Each iteration of the algorithm comprises three phases. First, a perturbation is applied to the current locally optimal candidate solution s resulting in an intermediate candidate solution s', which is not locally optimal. Next, local search is applied starting from s' resulting in a locally optimal candidate solution s''. Finally an acceptance criterion is used to decide whether the search is continued from s or s''. Crucial to the performance of iterated local search is the strength of the perturbation: It should not be too strong because a strong perturbation makes it closer to a random restart local search algorithm and it should not be too weak because this could cause the search to return to the previously found local optimum.

Greedy randomized adaptive search procedures

Each iteration of GRASP (Feo and Resende, 1989, 1995) comprises two phases. In the first phase, a constructive heuristic is used to generate a large number of different high quality candidate solutions. At each step of the constructive heuristic, a greedy function assigns values to the solution components and a number of the best-ranked components are included in a restricted candidate list. From this list, a solution component is then chosen randomly, according to a uniform distribution. In the second phase, the generated solutions are improved by using a local search algorithm. Once the termination criterion is met, the algorithm returns the best solution that it has found during the search.

Evolutionary computation

EC (Rechenberg, 1973; Fogel et al., 1966; Holland, 1975; Goldberg, 1989) is inspired by the ability shown by populations of living beings to evolve and adapt to changing conditions under the mechanism of natural selection. They are iterative procedures that start with an initial population of solutions, which is then repeatedly modified by applying a series of evolutionary operators such as reproduction, recombination, and mutation. The reproduction operator selects high quality solutions that will be part of the population for the next iteration. The recombination operator combines solution components of two or more solutions to generate a new offspring solution. The mutation operator injects diversity in the population of solutions by perturbing some of them. Several high performing evolutionary algorithms designed for combinatorial optimization use a subsidiary local search to refine the candidate solutions in the population. They are seen as EC algorithms that search a space of locally optimal solutions and are usually referred to as memetic algorithms (MAs) (Moscato, 1989, 1999).

Scatter search

SS (Glover, 1977) has several similarities with MAs. It operates on a population of solutions called reference solutions and generates new solutions called trial solutions for the next iteration using a recombination operator that is analogous to recombination operators of EC algorithms. A local search algorithm is then applied on the trial solutions. These improved solutions form the set of dispersed solutions. The new set of reference solutions that will be used in the next iteration is selected from the current set of reference solutions and the newly created set of dispersed solutions.

Ant colony optimization

ACO (Dorigo, 1992; Dorigo and Stützle, 2004) is a metaheuristic inspired by the pheromone trail laying and following behavior of some ant species. In ACO algorithms, artificial ants are stochastic solution construction procedures that generate solutions in each iteration guided by artificial pheromones trails and by the cost of the solution components; the ants' solutions are then used to modify the artificial pheromone trails. This mechanism shifts the stochastic solution construction procedure towards the construction of solutions similar to the better ones seen in the previous iterations of the algorithm. Usually, high performance ACO algorithms designed for tackling combinatorial optimization problems use a subsidiary local search algorithm. Once ants complete their solution construction phase, the local search algorithm is used to refine their solutions before using them for the pheromone update.

2.2 Metaheuristics for stochastic combinatorial optimization

Stochastic combinatorial optimization problems are similar to combinatorial optimization problems except that some of the problem parameters are not known exactly but probability distributions governing them are known or can be estimated. Many important and interesting problems in strategic planning for collection and distribution services, communication and transportation systems, job scheduling, and modeling and simulation of large molecular systems in bio-informatics can be modeled as stochastic combinatorial optimization problems (Fu, 2002). In fact, in these applications, stochastic combinatorial optimization problems offer more realistic models than their deterministic counterpart. In order to tackle these problems, it is customary that a setting is considered in which the cost of each solution is a random variable, and the goal is to find a solution that minimizes some statistics of the latter. For a number of practical and theoretical reasons, the optimization is performed with respect to the expectation (Fu, 1994, 2002). In this thesis, we consider stochastic combinatorial optimization problems that can be described as:

$$\text{minimize} \quad F(x) = E\big[f(x, \boldsymbol{\omega})\big], \tag{2.1}$$

where x is a solution from S, the finite set of feasible solutions, the operator E denotes the mathematical expectation, and f is the cost function, which depends on x and on

a multivariate random variable $\boldsymbol{\omega}$. The presence of the latter makes $f(x, \boldsymbol{\omega})$ a random variable. The goal is to find a feasible solution that minimizes the expected cost.

A paradigmatic example of a stochastic combinatorial optimization problem is the probabilistic traveling salesman problem (PTSP) (Jaillet, 1985), which is also one of the two problems that we consider in this thesis. The PTSP is similar to the TSP with the main difference being that each node has a probability of requiring a visit. The goal is to find a TSP tour that minimizes the expected cost of a pruned tour; it is obtained only after knowing the nodes that require being visited and by skipping the nodes that do not require being visited according to some predefined rules.

Designing efficient algorithms for solving stochastic combinatorial optimization problems is a hard task. The difficulty is due to two factors. Firstly, these problems suffer from the same combinatorial explosion of the number of potential solutions as combinatorial optimization problems do; secondly, the exact computation of the cost of a solution is typically complicated and computationally expensive because of the added element of uncertainty in the problem parameters. In fact, the second factor makes stochastic combinatorial optimization problems much more difficult to solve when compared to their deterministic counterparts. In contrast to deterministic problems, for stochastic combinatorial optimization problems exact methods have achieved less success and they can solve only small instances to proven optimality (Bianchi et al., 2009). This motivated researchers and practitioners to focus on the development of SLS methods. The study of stochastic combinatorial optimization problems through the viewglass of SLS methods and, in particular of metaheuristics, is a quite recent and fast growing research area.

The central issue in applying metaheuristics to stochastic combinatorial optimization problems is the computation of the expected cost and the consequent computation time trade-off between the search time and the cost computation time. The cost computation can be performed using two main approaches: *analytical computation*, that uses closed form expressions for computing the expected cost, and *empirical estimation*, that uses Monte Carlo methods for estimating the expected cost.

In the rest of this section, we discuss how this computation time trade-off has been addressed in metaheuristics literature for stochastic combinatorial optimization problems. Note that the overview provided in this chapter is in fact a bird's eye view of the literature. For an exhaustive coverage on metaheuristics applied to stochastic combinatorial optimization problems, we refer the reader to the excellent survey by Bianchi et al. (2009).

2.2.1 Analytical computation algorithms

For a number of stochastic combinatorial optimization problems, it is possible to compute the exact cost of a solution using closed-form expressions. When such an expression is available for a given problem, solving the given stochastic combinatorial optimization problem is not essentially different from a deterministic problem. Thus, a metaheuristic designed for a deterministic combinatorial optimization problem can easily be extended to solve the stochastic problem by using the closed-form expression to compute the solution cost. For the PTSP, the cost of a solution can be computed with a $O(n^2)$ closed-form expression. Using this closed-form expression, several authors extended TSP metaheuristics to the PTSP: Bianchi et al. (2002a,b); Bianchi (2006); Bianchi and Gambardella (2007) proposed ACO and local search algorithms; Rosenow (1997) studied an EC algorithm; Marinakis et al. (2008) applied GRASP and Marinakis and Marinaki (2010) designed a particle swarm optimization algorithm. Other examples of analytical computation metaheuristics to stochastic combinatorial optimization problems include an EC algorithm to tackle stochastic scheduling problems (Easton and Mansour, 1999) and a vehicle routing problem with stochastic demand and soft time windows (Mak and Guo, 2004).

The primary disadvantage of analytical computation algorithms is that they do not take into account the central issue, that is, the trade-off between the time taken for searching solutions and the time taken for evaluating their cost. For most stochastic combinatorial optimization problems, including the PTSP, the closed-form expressions are computationally expensive. The adoption of these expressions in optimization algorithms negatively affects the time spent in exploring the search space and eventually the final solution quality. This issue can be addressed by using less computationally expensive closed-form expressions to approximate the cost of the solution. Often, this expression is a truncated version of the original closed-form expression. This is the central idea in analytical approximation algorithms[3], a prominent subclass of analytical computation algorithms. For the PTSP, an analytical approximation technique has been used in ACO algorithms by Bianchi et al. (2002a) and Branke and Guntsch (2003, 2004) and in SS by Liu (2007, 2008). Similarly, for the vehicle routing problem with stochastic demands, Bianchi et al. (2006) developed approximation schemes and investigated them in SA, TS, ILS, ACO, and MA. Séguin (1994) and Gendreau et al.

[3]It should be noted that the usage of word "approximation" in analytical approximation algorithms for stochastic combinatorial optimization problems refers to the approximation of the cost function, and, hence, has a different meaning from the usage of this word in "approximation algorithms", which are guaranteed to find solutions within a specified factor of the optimal.

(1996b) derived an approximation scheme for TS to tackle the vehicle routing problem with stochastic demands and customers. Haugland et al. (2007) used an analytical approximation TS to tackle a variant of the vehicle routing problem with stochastic demands. Teodorović and Pavković (1992) addressed the vehicle routing problem with stochastic demands with multiple vehicles using a SA that adopts analytical approximation. The main drawback of analytical approximation is that the design of a proxy function is strongly problem-specific and there is no general rule for finding an efficient proxy function (Bianchi et al., 2009).

2.2.2 Empirical estimation algorithms

In a vast majority of stochastic combinatorial optimization problems, it is very difficult, or even impossible to derive closed-form expressions and appropriate analytical approximation schemes. In this case, the common approach is empirical estimation. In this approach, the cost $F(x)$ of a solution x is estimated on the basis of sample costs $f(x,\omega_1), f(x,\omega_2), \ldots, f(x,\omega_M)$ obtained from M independent realizations $\omega_1, \omega_2, \ldots, \omega_M$ of the random variable $\boldsymbol{\omega}$:

$$\hat{F}_M(x) = \frac{1}{M}\sum_{r=1}^{M} f(x,\omega_r). \tag{2.2}$$

As it can easily be shown, $\hat{F}_M(x)$ is an *unbiased* estimator of $F(x)$.

Crucial to the effectiveness of using the estimation approach in metaheuristics is the accuracy of the obtained cost estimate $\hat{F}_M(x)$. More precisely, metaheuristics compare two or more solutions at each iteration and they move to a solution having the least cost estimate: the lower the accurateness of the cost estimates of the solutions, the higher the uncertainty in determining whether one solution is better than the other. For example, given two solutions x and x', and their estimates $\hat{F}_M(x)$ and $\hat{F}_M(x')$, $\hat{F}_M(x) < \hat{F}_M(x')$ does not guarantee that $F(x) < F(x')$. The accuracy of the cost estimate can be assessed by the variance associated with the cost estimate. The convergence rate of estimation via Monte Carlo simulation suggests that the variance of the estimator decreases with $O(1/\sqrt{M})$. This means that a large number of realizations is required to estimate with a low variance the cost of each solution. Unfortunately, this strategy results in a high computation time for estimating the cost of each solution. As a consequence, the time spent in exploring the solution space will be significantly reduced.

To address the aforementioned issue, several variants of the basic estimation ap-

proach have been proposed. They can be classified as exterior sampling methods and interior sampling methods (Verweij et al., 2003). In exterior sampling methods, the cost of a solution is estimated as in Equation 2.2, but all the realizations are generated at the beginning of the search and kept unchanged throughout the search. Since a fixed set of realizations is used in all iterations, the cost estimate of a solution x remains unchanged during the search. Similar to the analytical computation, this results in the transformation of the given stochastic combinatorial optimization problem into a deterministic problem, which is then solved by a deterministic optimization algorithm. This procedure may be repeated by generating several fixed sets of realizations and solving the deterministic optimization problem for each fixed set. The number of realizations is crucial to the effectiveness of the exterior sampling methods and it is often difficult to determine. A prominent exterior sampling method is sample average approximation (Kleywegt et al., 2002; Verweij et al., 2003). Although this technique offers the flexibility of using metaheuristics to solve the deterministic problem, only exact techniques have been studied in the literature.

In interior sampling methods, the realizations are changed during the search. This is performed by adding new realizations to the previously generated ones, by considering a subset of previously generated realizations, or by generating new realizations at each iteration. These methods are widely used within metaheuristics designed to tackle stochastic combinatorial optimization problems. Several interior sampling methods have been proposed for metaheuristics. Based on the pivotal idea behind the interior sampling method used, we can group metaheuristics as follows: Static sample size algorithms, progressional sample size algorithms, and adaptive sample size algorithms.

Static sample size algorithms: These algorithms, at each iteration, use a sufficiently large number of realizations to obtain the cost estimate of each solution. The number of realizations is determined a priori and it is kept constant throughout the algorithm. Numerous metaheuristics have been proposed based on this method. Haddock and Mittenthal (1992) applied an estimation-based SA to determine optimal parameter levels of a stochastic system. They modify a typical SA algorithm by substituting an estimate of the solution cost in all places requiring a deterministic cost. Since computing an accurate solution cost is computationally expensive, the authors use an annealing schedule that decreases the temperature rapidly so that the algorithm does not accept non-improving solutions after few iterations. Lutz et al. (1998), Finke et al. (2002), Dengiz and Alabas (2000) adopt a standard TS algorithm that uses an external simulator to estimate the solution cost. Aringhieri (2004) proposed an estimation-based TS for

two stochastic combinatorial optimization problems that arise in the design of telecommunication networks. Cheung et al. (2007) tackled a stochastic delivery problem using an estimation-based TS. Daniel and Rajendran (2005) applied an EC algorithm to the inventory optimization problem. Jellouli and Châtelet (2001) used an EC algorithm to address a supply-chain management problem in a stochastic environment.Wang and Singh (2008) applied a particle swarm optimization to tackle resource portfolio planning optimization under demand uncertainty. Yoshitomi (2002) and Yoshitomi and Yamaguchi (2003) used an EC algorithm for tackling the stochastic job-shop scheduling problem. The algorithm keeps track of promising solutions obtained during the search process. Finally, each promising solution is evaluated using a large number of realizations and the one with the least cost is selected as the best solution. Watson et al. (1999) tackled a stochastic warehouse scheduling problem using an EC algorithm. A particularity of the cost estimation is that the simulator is run with a fast but less accurate mode until the algorithm stops and the algorithm stores a number of promising solutions during the search. These solutions are evaluated in a slow but more accurate mode to determine the best solution. The algorithm also exploits knowledge from the internal states of the simulator to construct solutions.

Progressional sample size algorithms: These are theoretically elegant, general-purpose algorithms, which can be applied to any stochastic combinatorial optimization problem. In these algorithms, the number of realizations is gradually increased with respect to the iteration number. Often, a predefined sample size schedule is used to increase the number of realizations at some rate in order to guarantee the asymptotic convergence of the algorithm to the optimal value. The key idea behind the sample size schedule is to save computation time during the initial search phase, when the quality of the solutions is rather low. In this stage, the comparison is performed with less accurate cost estimates using few realizations. As the search progresses, high quality solutions are obtained, and a large number of realizations is used to obtain accurate cost estimates.

Gutjahr and Pflug (1996) proposed an estimation-based SA. In this algorithm, the sample size schedule increases the number of realizations more than quadratically with respect to the iteration number. Later, Gutjahr et al. (2000a) used this algorithm to tackle a discrete time/cost tradeoff problem in activity planning. Gutjahr (2004) also used this algorithm as a yardstick in the context of the TSP with time windows and stochastic service times. A restrictive variant of this algorithm has been studied earlier by Gelfand and Mitter (1985). Alrefaei and Andradóttir (1999) proposed a general

purpose SA, in which the temperature is kept constant. This algorithm uses a sample size schedule that slowly increases the number of realizations at a particular rate with respect to the iteration number. They discussed two schemes for comparing solutions. The first scheme consists of counting the number of visits to promising solutions and selecting the best solution based on the number of visits. In the second scheme, the algorithm checks if the previous estimate of a given solution is already available from earlier iterations. When available, instead of generating new realizations, only a few realizations are generated for cost estimation. This cost estimate is then cumulated with the previously available estimate. The second scheme has also been used in an SA by Fox and Heine (1995). Bowler et al. (2003) proposed an SA algorithm for the PTSP. In this algorithm, the temperature parameter is changed with respect to the variance associated with the cost estimate. An increasing sample size schedule is used to reduce the variance of the cost estimate so that in the beginning of the search the algorithm accepts non improving solutions and, as the search progresses, it starts to accept only improving solutions. Gutjahr (2003, 2004) proposed S-ACO, an ACO algorithm for stochastic combinatorial optimization problems and later Gutjahr et al. (2007) developed S-VNS, a variable neighborhood search algorithm for stochastic combinatorial optimization problems. In both algorithms, the sample size schedule increases the number of realizations linearly in dependence of the iteration counter. S-ACO and S-VNS have been tested on the TSP with stochastic service time and on the stochastic project portfolio selection problem, respectively.

Adaptive sample size algorithms: In this class of algorithms, the number of realizations at each iteration is chosen adaptively in dependence of the variance associated with the cost estimates of solutions involved in the comparison. These algorithms make use of inferential statistics to determine the appropriate number of realizations needed for each estimation. In inferential statistics, the measure of accuracy of the cost estimate is assessed by the standard error of the cost estimate, which is given by $\sqrt{s_M^2/M}$, where s_M^2 is the variance associated with the cost estimate. The central idea behind using an adaptive sample size can be explained as follows. Given two solutions, the cost estimate of each solution is computed on a realization-by-realization basis. Using the cost estimate of a solution and its standard error, a confidence interval is constructed. The confidence interval is a range of values within which the true cost falls with a given level of confidence. If the confidence interval of a solution x falls outside the confidence interval of another solution x', then the adaptive sample size procedure infers that the cost estimates of the two solutions are different. As soon as

the inference is obtained, the cost computation is stopped and the solution with the least cost estimate is selected as the best. Otherwise the cost estimate computation is continued. Often a statistical test is used to detect differences in the cost estimates of two or more solutions.

Alkhamis et al. (1999) use a decreasing cooling schedule SA and used an adaptive sample size procedure based on a statistical test to compare two neighbor solutions. Homem-de-Mello (2000, 2003) proposed a general framework of interior sampling methods called variable sample methods to be used in metaheuristics. This framework comprises the use of different realizations along the search within the adaptive sample size procedure. To guarantee the convergence of the algorithm, the sample size is increased at some specific iterations regardless of the number of realizations prescribed by the adaptive sample size. The author illustrated this framework using a SA. Alkhamis and Ahmed (2004) proposed a constant temperature SA that adopts the adaptive sample size procedure of Alkhamis et al. (1999) to stop the cost estimation; however, the best solution is selected based on the number of times a particular solution has been visited. Gutjahr (2003, 2004) proposed a variant of S-ACO called S-ACOa that uses a statistical test for comparing solutions. ACO/F-Race (Birattari et al., 2005) is an improved variant of S-ACOa, in which the number of realizations for each comparison is determined based on the F-Race procedure (Birattari et al., 2002; Birattari, 2009). Bulgak and Sanders (1998) used a standard SA to tackle a stochastic buffer allocation problem in the context of a complex manufacturing system. The authors use an adaptive sample size procedure similar to the one proposed by Homem-de-Mello (2003).

There exist a number of sophisticated Monte Carlo methods that can substantially reduce the variance of the cost estimate without requiring a large number of realizations. Examples include the method of common random numbers, control variates, and importance sampling. Homem-de-Mello (2000, 2003) and Alkhamis et al. (1999) discussed the possibility of using of the method of common random numbers and the importance sampling procedures in a SA. Costa and Silver (1998) describe a TS algorithm for a stochastic ordering problem with time constraints. In this algorithm, the variance is reduced with a variance reduction procedure called descriptive sampling (Jönsson and Silver, 1996). A statistical test is used to compare solutions. The algorithm memorizes a number of promising solutions; at the end, each promising solution is evaluated using a large number of realizations and the one with least cost is selected as the best solution.

2.2.3 Other algorithmic approaches

Here, we briefly mention some algorithms that we omitted because they are beyond the scope of the thesis. Although less predominant in metaheuristics literature, these algorithms are established as highly effective approaches to tackle stochastic combinatorial optimization problems. Stochastic partitioning methods are algorithms that recursively divide the search space into a number of subregions and allot the computational effort to the subregions where high quality solutions have been obtained in previous iterations. Examples of stochastic partitioning methods are beam search, stochastic branch and bound, and nested partition methods. In these algorithms, metaheuristics are used as a subsidiary algorithmic component. Beam search (Beraldi and Ruszczynski, 2005; Erel et al., 2005) adopts the analytical approximation approach. Stochastic branch and bound (Norkin et al., 1998a,b; Gutjahr et al., 1999, 2000a,b; Doerner et al., 2006) and the nested partition methods (Shi and Ólafsson, 2000; Pichitlamken and Nelson, 2003) use the empirical estimation approach; in particular, they belong to the class of interior sampling methods.

Progressive hedging (Rockafellar and Wets, 1991) is a framework that consists in using a number of realizations of the random variable and solving a deterministic optimization subproblem for each realization. A repairing procedure is used to obtain a feasible combined solution that hedges against the future uncertainty. Similar to stochastic partitioning methods, metaheuristics have been used as a component to solve the deterministic subproblems (Løkketangen and Woodruff, 1996; Haugen et al., 2001; Hvattum and Løkketangen, 2008).

Rollout algorithms (Bertsekas et al., 1997) were first proposed for the approximate solution of discrete optimization problems. They are constructive heuristics that adopt sophisticated heuristic function in the solution construction step. Bertsekas and Castanon (1999) and Secomandi (2000, 2001) and applied rollout algorithms to stochastic combinatorial optimization problems.

A particular field that is extremely relevant to the estimation-based metaheuristics for stochastic combinatorial optimization is simulation optimization, which is dominated by provably convergent random search algorithms. Some prominent examples include the stochastic ruler method (Yan and Mukai, 1992; Alrefaei and Andradóttir, 2001) and the random search algorithms of Andradóttir (1995, 1996). Moreover, there exists a large amount of literature on applying EC algorithms to optimization problems with noisy cost functions. We refer the reader to Jin and Branke (2005) for a survey. In a number of settings, meta-modeling approaches such as the response surface methodol-

ogy using regression techniques and neural networks have been used. When the number of possible solutions is rather small, ranking and selection/ordinal optimization methods (Ho et al., 1992) have been applied. Although they lack the search functionality, in principle, they could be easily used inside metaheuristics to select the best solution at each iteration.

Finally, when a stochastic combinatorial optimization problem can be formulated as a linear programming problem, then stochastic integer programming techniques can be applied. We refer the reader to van der Vlerk (1996-2007) for a bibliography.

2.3 Summary

In this chapter, we reviewed the field of metaheuristics with a focus on their applications to stochastic combinatorial optimization problems. The primary difficulty in solving stochastic combinatorial optimization problems is the combinatorial explosion of potential solutions, which is further exacerbated by the added element of uncertainty in the problem parameters. We introduced combinatorial optimization problems and we provided a high level description of some common metaheuristics for solving them. We discussed why stochastic combinatorial optimization problems are more difficult than their deterministic counterparts and how metaheuristics take into account the stochastic character of these problems.

To tackle stochastic combinatorial optimization problems, metaheuristics are quite appealing because they offer several intelligent strategies to attack the combinatorial nature of these problems. However, a crucial factor that determines the success of metaheuristics for stochastic combinatorial optimization problems is the way in which the computational effort is allocated between the search and the solution cost computation.

Metaheuristics proposed for stochastic combinatorial optimization problems can be grouped into two main classes: *analytical computation* algorithms and *empirical estimation* algorithms. In analytical computation algorithms, the cost computation is performed using closed-form expressions. A prominent subclass of analytical computation algorithms is analytical approximation, where truncated versions of the original closed-form expressions for the given problem are adopted for cost computation. The main drawbacks of the analytical computation approach are that it is not general-purpose and it cannot be applied to problems in which the cost difference cannot be expressed in an analytical way. In empirical estimation algorithms, the cost of a solution for a given problem is estimated by Monte Carlo simulation. These algorithms can overcome the difficulties posed by analytical computation. However, since solution

costs are estimated in the empirical estimation approach, the number of realizations adopted is crucial to address the trade-off between the computation time used for the search and the one used for the cost computation. Several Monte Carlo methods have been proposed to control the number of realizations in estimation-based metaheuristics. The widely used Monte Carlo methods belong to the so-called interior sampling methods, where the realizations are generated afresh at each iteration. Three major schemes are found in interior sampling methods to control the number of realizations. In the first scheme, the number is determined a priori and kept constant throughout all the iterations of the algorithm. In the second scheme, the algorithm starts with a small number of realizations, which is then increased as the search progresses according to a predefined scheme. In the third scheme, the number of realizations is determined dynamically based on a statistical test. Moreover, in this scheme, several variance reduction techniques have been discussed to reduce the number of realizations needed for the cost estimation.

Chapter 3

Stochastic routing problems: A review

Routing problems in distribution management, often referred to as vehicle routing problems in the literature, are among the most intensively investigated combinatorial optimization problems. Generally speaking, these problems involve finding a cost-effective way to distribute or collect goods across a logistic network. Routing problems are used to model a number of important problems such as mail delivery, school bus routing, solid waste collection, heating oil distribution, parcel pick-up and delivery, and dial-a-ride systems. Even a small improvement in the efficiency of the distribution process can lead to a significant cost saving because the process is repeated on a regular basis. Often, routing problems are defined on a graph G with the following elements:

- a set $V = \{1, 2, ..., n\}$ of nodes that represent customers with node 1 being the depot;
- a set $A = \{\langle i, j\rangle : i, j \in V, i \neq j\}$ of edges, where an edge $\langle i, j\rangle$ connects two nodes i and j;
- a set $C = \{c_{ij} : \langle i, j\rangle \in A\}$ of travel costs, where c_{ij} is the cost of using an edge $\langle i, j\rangle \in E$. Depending on a given setting, the travel cost c_{ij} may represent distance, time, fuel consumption, etc.;
- a set $D = \{d_i : i \in V, i \neq 1\}$ of demands, where d_i is the non-negative demand of node (customer) i.

The customers are served by a fleet of m vehicles that distribute or collect goods from customers, where m is either a constant or a decision variable. The goal is to find

m vehicle routes that minimize the total cost involved in routing the vehicles subject to given constraints. The most common constraints are that each customer should be served once by exactly one vehicle and each vehicle route starts and ends at the depot. In practical applications, the routing problems are solved with a number of side constraints. Some classical side constraints used in the literature are:

- limited capacity: customers are served by m vehicles of same capacity and the total demand that arises in each route may not exceed the vehicle's capacity;
- time window constraint: each customer must be visited within a given time interval;
- multiple depots: there are more than one depot to serve the customers and a solution also includes the assignment of customers to depots;
- pickup and delivery: vehicles are allowed to collect and to deliver goods from customers;
- split delivery: a customer is allowed to be served by more than one vehicle to reduce costs.

There exists a vast literature on algorithms proposed for routing problems. For a detailed overview, we refer the reader to Laporte (1992), Laporte et al. (2000), Toth and Vigo (2001), and Cordeau et al. (2002).

Stochastic routing problems (SRPs) belong to the class of stochastic combinatorial optimizations problems. SRPs have enormous practical importance not only in transportation but also in strategic planning and scheduling (Bertsimas, 1988). They are similar to deterministic routing problems except that customers visits, their demands, or travel times are not known exactly but probability distributions governing them are known or can be estimated. The introduction of probabilistic elements into the routing problem increases the difficulty of tackling the problem and, as a result, solving them is quite difficult. It has been shown that several fundamental properties of routing problems no longer hold in stochastic variants. For an Euclidean TSP instance, a property of the optimal route is that edges in the optimal route do not cross each other but for the PTSP, this property is not true (Gendreau et al., 1996a). Exact methods are often limited to rather small instances. Therefore, in recent years, much attention has been devoted to the development of stochastic local search (SLS) algorithms such as iterative improvement algorithms and metaheuristics for tackling SRPs. In particular,

a significant progress has been made towards exploiting problem-specific knowledge in designing effective algorithms.

The most widely used approach to tackle the SRPs is *a priori* optimization (Jaillet, 1985; Bertsimas et al., 1990). It consists of two stages. In the first stage, a vehicle route is determined before the actual realization of the random variables is available. This is the so-called *a priori* solution. In the second stage, after the realizations of the random variables are known, it may then be impossible to follow the *a priori* solution because, for example, the total demand associated with the route may exceed the capacity of the vehicle or some customers do not require being visited. In this case, recourse actions are applied to the *a priori* solution such as going back to the depot to empty the vehicle or skipping customers. The solution obtained from the *a priori* solution by recourse actions is the *a posteriori* solution. Note that in problems with m vehicles, m *a priori* solutions are determined. Typically, *a priori* optimization is modeled as a stochastic program with recourse, in which the goal is to find an *a priori* solution that minimizes the expected cost of the associated *a posteriori* solution. As described in Chapter 2, the cost of the *a priori* solution can be evaluated in two ways: analytical computation and empirical estimation. In this thesis, we follow an *a priori* optimization approach via stochastic program with recourse model to tackle the probabilistic traveling salesman problem (PTSP) and the vehicle routing problem with stochastic demands and customers (VRPSDC).

Besides the aforementioned stochastic program with recourse, *a priori* optimization can be modeled as a chance constrained program. Here, the goal is to find an *a priori* solution such that the probability of performing recourse actions does not exceed a certain threshold. Since this model does not take into account the cost associated with recourse actions, it is less realistic than the stochastic program with recourse; for the PTSP, this is not an applicable goal. As a consequence, although solving SRPs by stochastic program with recourse is more difficult than the chance constrained program, the former is widely adopted.

An alternative approach to *a priori* optimization is re-optimization. In this approach, the route followed by a vehicle is recomputed when realizations of the random variables become available. The intensity of re-optimizing routes ranges from restricted re-optimization, which is performed only when required, to greedy re-optimization, which is performed as soon as new data is available (Psaraftis, 1995). For algorithms that solve the SRPs using re-optimization, we refer the reader to Dror et al. (1989), Dror (1993), Psaraftis (1995), and Secomandi (2000, 2001). Apart from *a priori* optimization and re-optimization, the possibility of solving the SRPs using Markov decision

processes is considered by Dror et al. (1989). However, in a later work, Dror (1993) observed that instances with just few customers are computationally intractable.

The PTSP and the VRPSDC that we consider in the thesis are among the classical SRPs in distribution management (Gendreau et al., 1996a). Other SRPs that belong to this category are the vehicle routing problem with stochastic customers (VRPSC), the vehicle routing problem with stochastic demands (VRPSD), and the vehicle routing problem with stochastic travel and service times (VRPSTST). In the remainder of the chapter, we summarize the literature on these problems and some other extensions that are relevant to the thesis.

3.1 The probabilistic traveling salesman problem

The PTSP is a central problem in stochastic routing (Bertsimas, 1988). It is similar to the TSP with the main difference being that each node has a probability of requiring a visit. The *a priori* solution must be found before knowing the nodes that require being visited; the associated *a posteriori* solution, which is computed once the nodes that require being visited are known, is obtained by visiting them in the order prescribed by the *a priori* solution and by skipping the ones that do not require being visited. A PTSP instance is called homogeneous if all node probabilities are the same, and it is called heterogeneous otherwise.

Theoretical studies

Jaillet (1985, 1988) introduced the PTSP and investigated several theoretical properties of the problem. The author derived a $O(n^2)$ closed-form expression to compute the exact solution cost, bounds for the optimal solution, and the relationship between the optimal solutions of the PTSP and of the TSP. Moreover, the author showed that algorithms designed for the TSP could perform poorly on PTSP instances with very low probability values. Berman and Simchi-Levi (1988) extended the theoretical research of Jaillet (1985, 1988) to heterogeneous instances. Bertsimas (1988) and Bertsimas and Howell (1993) investigated several other properties. This resulted in the attainment of improved bounds on the optimal solution value. Bowler et al. (2003) studied the behavior of the optimal solution value on homogeneous instances as a function of instance size and probability.

Exact algorithms

An exact method that solves the PTSP to proven optimality has been proposed by Laporte et al. (1994). It is a branch-and-cut algorithm that can solve instances of size up to 50. Jaillet (1985, 1988), Bertsimas (1988), Berman and Simchi-Levi (1988) considered the possibility of using a branch-and-bound algorithm. Nevertheless, the authors speculated that this algorithm could be useful only for small instances and therefore it has not been tested experimentally. Rosenow (1998) implemented this branch-and-bound algorithm and showed that it can tackle instances of size up to 14.

Constructive heuristics

Jaillet (1985, 1988) and Jézéquel (1985) studied the adoption of two TSP heuristics namely, the nearest neighbor and the Clarke-Wright savings algorithm, to the PTSP. Rossi and Gavioli (1987) customized the two TSP heuristics in such a way that the construction steps take into account the node probabilities. Experiments with homogeneous instances showed that the customized variants are particularly effective for instances where the node probability is less than 0.6. Bertsimas (1988) and Bertsimas and Howell (1993) investigated the use of the space filling curve heuristic and its behavior. Bianchi (2006) compared the space filling curve heuristic with radial sort, farthest insertion and nearest neighbor heuristics, observing that the farthest insertion heuristic performs better than the others.

Iterative improvement algorithms

For the homogeneous PTSP, Bertsimas (1988) derived closed-form delta evaluation expressions for the 2-exchange neighborhood and the node-insertion neighborhood. Equipped with these expressions, the author proposed two iterative improvement algorithms: `2-p-opt` and `1-shift`. For both algorithms, the total time complexity of the neighborhood exploration and evaluation is $O(n^2)$. For the heterogeneous case, Chervi (1988) proposed closed-form delta evaluation expressions for `2-p-opt` and `1-shift`, where each algorithm explores and evaluates the neighborhood in $O(n^3)$. Bianchi et al. (2005) and Bianchi and Campbell (2007) proved that the expressions derived by Bertsimas (1988) and Chervi (1988) are incorrect, and corrected the errors. Furthermore, for the heterogeneous PTSP, Bianchi and Campbell (2007) showed that the neighborhoods in `2-p-opt` and `1-shift` can be explored and evaluated in $O(n^2)$ rather than $O(n^3)$. Tang and Miller-Hooks (2004) investigated an approximation scheme for `2-p-opt`. In this scheme, the closed-form equation is truncated and the level of truncation decreases

with an increase in the number of improvements. This scheme was tested on uniform instances of size up to 100 and a speedup of two orders of magnitude has been reported at the expense of a drop of solution quality of 0.5%. Campbell (2006) proposed the idea of node aggregation, in which the size of an instance is reduced by grouping the nodes that are close to each other. The author tested the idea in `2-p-opt` and `1-shift` with instances of size ranging from 100 to 1000. The results showed that the idea of aggregation is quite effective on clustered instances.

Metaheuristics

Much of the early research in the development of metaheuristics for the PTSP focused on algorithms that use the closed-form expression of Jaillet (1985, 1988). Bianchi et al. (2002a,b) applied ant colony system (ACS) (Dorigo and Gambardella, 1997), a high performing ACO variant, to solve the PTSP. The authors considered two algorithms in their study: the first algorithm is an ACS that computes the cost of the PTSP solutions using the TSP cost function, in which probabilities associated with the nodes are simply ignored; the second is pACS, an ACS that adopts the closed-form expression to compute the cost of the solutions. They showed that on homogeneous instances pACS achieves significantly better solutions when the probability value is less than 0.5. Rosenow (1997) studied a genetic algorithm that uses a simple crossover operator (Grefenstette et al., 1985) without any mutation. The effectiveness of the proposed algorithm has been established by comparing it with the branch-and-bound algorithm proposed by Jaillet (1985, 1988). However, we speculate that this algorithm requires very high computation time because at each iteration the algorithm generates 100 individuals, each of which is evaluated by the closed-form expression.

Branke and Guntsch (2003, 2004) investigated a truncated version of the closed-form expression in pACS and showed that for homogeneous instances with a probability value greater than 0.5, the computation time can be reduced significantly without significant loss in solution quality. Liu (2007, 2008) used the truncated version within a scatter search. The author experimented with a number of neighborhood relations for the scatter search and found that a combination of the node-insertion and 2-exchange neighborhood relations is very effective.

Bianchi (2006) and Bianchi and Gambardella (2007) integrated `1-shift` into pACS and showed that the resulting pACS+1-shift algorithm significantly outperforms the farthest insertion heuristic, pACS, scatter search (Liu, 2007, 2008), and the pACS variant that adopts the truncated version of the closed-form expression (Branke and

Guntsch, 2003, 2004). Marinakis et al. (2008) investigated the usage of a greedy randomized adaptive search variant, ENS-GRASP, which uses the closed-form expression. For few homogeneous instances with a probability value greater than 0.5, the authors showed that ENS-GRASP achieved solution costs that are slightly better than those of pACS+1-shift. However, a main issue in this comparative study is that the two algorithms use different stopping criteria. ENS-GRASP and pACS+1-shift are allowed to run for 1000 iterations and $n^2/100$ CPU seconds, respectively, and the computation time needed by ENS-GRASP is not reported in Marinakis et al. (2008). Recently, Marinakis and Marinaki (2010) proposed HybMSPSO, a particle swarm optimization algorithm built on top of ENS-GRASP. HybMSPSO also adopts the closed-form expression to compute the solution cost. The authors show that HybMSPSO obtains slightly better solutions than pACS+1-shift on homogeneous instances with a probability value equal to 0.5. However, there are again two main problems in this comparative study. Firstly, it is not clear if the observed differences are significant in a statistical sense. Secondly, the same set of instances is used to fine tune the parameters of HybMSPSO, to select HybMSPSO as the best from a set of seven algorithms, and to compare HybMSPSO with pACS and pACS+1-shift: this might possibly induce a bias in favor of HybMSPSO. Note that the second problem is known as over-tuning (Birattari et al., 2006b; Birattari, 2009).

Concerning estimation-based algorithms, Bowler et al. (2003) proposed a proof-of-concept stochastic simulated annealing for the PTSP, in which the annealing schedule is controlled by the sampling error of the cost estimation. The proposed algorithm has been compared to the re-optimization strategy; the results showed that the solutions obtained by the former is not more than 14% worse than that of the latter. Gutjahr (2003, 2004) proposed a general purpose, estimation-based ACO algorithm called S-ACO and a variant S-ACOa. In S-ACO, the solutions produced at a given iteration are compared on the basis of a single realization; then the iteration-best solution is compared with the best-so-far solution on the basis of a number of realizations prescribed by a sample size schedule. In S-ACOa, the number of realizations is determined based on a statistical test. Gutjahr used the PTSP to calibrate the algorithm parameters of S-ACO. Although, not directly tested on the PTSP, S-ACOa can be easily applied to it. Birattari et al. (2006a) proposed ACO/F-Race that adopts the F-Race procedure (Birattari, 2004, 2009). It is an improved variant of S-ACOa, in which the solutions produced at a given iteration, together with the best-so-far solution, are compared using a pairwise statistical test for multiple comparisons. The preliminary results showed that for homogeneous uniform instances with probability values less than 0.5, ACO/F-Race

achieved solution costs that are significantly better than those of S-ACO and S-ACOa. However, S-ACO, S-ACOa, and ACO/F-Race are not expected to perform as well as pACS+1-shift. This is due to the following facts: firstly, these algorithms are proposed as proof-of-concepts; secondly, they are based on ant system, which is typically not as well performing as ACS; thirdly, they do not use any local search as a subsidiary solution improvement procedure. Note that the adoption of local search is crucial to the performance of ACO algorithms (Dorigo and Stützle, 2004). Bianchi (2006) and Bianchi and Gambardella (2007) also considered the sample size schedule of Gutjahr (2003, 2004) in pACS+1-shift, but concluded that this variant is significantly worse performing than the analytical computation variant of pACS+1-shift.

3.2 The vehicle routing problem with stochastic customers

In this problem, each customer has a probability of requiring being visited and has a deterministic demand. A vehicle with finite capacity has to collect goods from customers. The *a priori* solution is a sequence of all customers, which is decided before realizing customers presence and their demands; the associated *a posteriori* solution is obtained by following the *a priori* solution but by skipping customers who do not require being visited; as soon as the capacity is exhausted, the vehicle goes back to the depot for unloading and visits the next customer who requires being visited in the *a priori* solution.

Jézéquel (1985), Bertsimas (1988), and Bertsimas et al. (1990) investigated the properties, bounds, asymptotic behavior, and the adoption of heuristics for this problem. In all these studies, it is assumed that each customer has a unit demand. Waters (1989) tackled this problem with integer demands in a real time problem. The author compared the customary stochastic program with recourse model to re-optimization and showed that the savings obtained through re-optimization is rather small. Moreover, the drivers of the vehicles disliked the idea of changing routes frequently on a given day. Note that the author did not mention how the *a priori* solution is obtained.

3.3 The vehicle routing problem with stochastic demands

Undoubtedly, among all SRPs, the VRPSD is the one that has received most attention in the literature. In this problem, all customers require being visited but their demands are described by random variables. Tillman (1969) first introduced this problem in the context of a multiple depot delivery problem. Bertsimas (1988, 1992), Laporte

and Louveaux (1990), and Chepuri and Homem-de-Mello (2005) investigated bounds, asymptotic results, and theoretical properties. The recourse policies investigated for the VRPSD can be classified into two groups: reactive recourse and proactive recourse. The reactive recourse is a widely used policy, in which a capacitated vehicle returns to the depot to unload when a customer demand cannot be satisfied and then follows the *a priori* solution—see Stewart Jr. and Golden (1983), Dror and Trudeau (1986), Dror et al. (1989), Teodorović and Pavković (1992), Savelsbergh and Goetschalckx (1995), and Laporte et al. (2002). The proactive recourse is one in which a vehicle is allowed to return to the depot for unloading before its capacity is exhausted—see Bertsimas et al. (1995), Yang et al. (2000), Chepuri and Homem-de-Mello (2005), and Bianchi et al. (2006). For the chronological algorithmic developments for the VRPSD, we refer the reader to Gendreau et al. (1996a). In the following of this section, we highlight some important research efforts devoted exclusively to the VRPSD.

Laporte et al. (1989) proposed a branch-and-cut algorithm that solves the VRPSD to optimality. The authors tackled the VRPSD within a broader context of stochastic location-routing problems, where the location of the depot is also a decision variable. The results showed that this algorithm can tackle instances of size up to 30. Hjorring and Holt (1999) used the Integer L-Shaped method to solve the VRPSD with a single vehicle. Laporte et al. (2002) used an improved variant of the Integer L-Shaped method to solve the VRPSD with multiple vehicles. The results showed that this method can solve instances with up to 4 vehicles and 25 customers.

Tillman (1969), Stewart Jr. and Golden (1983), and Dror and Trudeau (1986) customized the Clark-Wright savings heuristic for the VRPSD. Bertsimas (1992) and Bertsimas et al. (1995) customized the cyclic heuristic, which was originally proposed for the deterministic vehicle routing problem by Haimovich and Rinnooy Kan (1985). Yang et al. (2000) proposed a heuristic that uses the idea of clustering customers. This algorithm comprises two phases: a phase that consists in clustering the customers in such a way that each cluster is served by one vehicle; another phase consists in finding the best *a priori* solution for each cluster. The authors implemented two variants: route-first-cluster-next and cluster-first-route-next. The authors also customized the iterative improvement algorithm OrOpt (Or, 1976) for the VRPSD. This algorithm uses a problem-specific approximation scheme in delta evaluation. Within the same algorithm, Bianchi et al. (2006) investigated the use of the TSP approximation to the VRPSD in delta evaluation.

Teodorović and Pavković (1992) proposed a simulated annealing algorithm to the VRPSD with multiple vehicles. Bianchi et al. (2006) investigated the effectiveness of

simulated annealing, tabu search, iterated local search, ant colony optimization, and memetic algorithms. The authors obtained new state-of-the-art algorithms by using the TSP 3-opt local search in an iterated local search and in memetic algorithms. Note that all these algorithms use closed-form expression to compute the solution cost. Recently, Chepuri and Homem-de-Mello (2005) applied the cross entropy method that adopts empirical estimation. The authors developed an adaptive sample size procedure that allows the algorithm to select the appropriate number of realizations needed for estimation.

3.4 The vehicle routing problem with stochastic demands and customers

The VRPSDC is one of the most difficult stochastic vehicle routing problems (Gendreau et al., 1996a). It combines the PTSP and the VRPSD: each customer has a probability of requiring being visited and has a stochastic demand. The *a priori* solution is a sequence of all customers; the associated *a posteriori* solution, which is computed after the realization of the stochastic elements, is obtained by following the *a priori* solution but with recourse actions such as skipping customers who do not require being visited and going back to the depot for unloading if the vehicle capacity is exhausted.

Bertsimas (1988, 1992) derived a closed-form expression to compute the exact cost of the VRPSDC solution. Séguin (1994) and Gendreau et al. (1995) extended this expression to take into account split deliveries. Using these expressions, the authors derived upper and lower bounds for the VRPSDC solution. They also studied several theoretical properties such as asymptotic behavior of *a priori* optimization with respect to the re-optimization strategy. Recently, FuCe et al. (2005) investigated the effectiveness of tackling the problem using *a priori* optimization. However, the authors did not report the results and their claim that this was the first study to use *a priori* optimization for the VRPSDC is rather false. Benton and Rossetti (1992) tackled this problem with re-optimization, where the solution is re-optimized whenever the demands are revealed. The authors showed this model is more effective than the recourse model.

Séguin (1994) and Gendreau et al. (1995) proposed an exact method based on the Integer L-shaped algorithm that has solved instances with number of nodes up to 46. The authors also observed that the presence of stochastic customers makes the problem more difficult to solve than the presence of stochastic demands. To study the impact of the number of stochastic customers on the difficulty of solving the problem, the proposed algorithm has been tested on instances with 1, $(n-1)/2$, and $n-1$ stochastic

customers. The results showed that the performance of the exact method decreases with an increase in the number of stochastic customers.

Jézéquel (1985) studied the possibility of using the Clark-Wright savings algorithm. However, the usefulness of this heuristic is rather unclear because the numerical results are not reported in this study. Bertsimas (1992) used the cyclic heuristic and analyzed its worst and average case behavior.

Séguin (1994) and Gendreau et al. (1996b) tackled this problem using a tabu search algorithm, TABUSTOCH. The algorithm adopts a randomized node-insertion neighborhood and a random tabu tenure. The main novelty of this algorithm consists in using a simple approximation scheme to identify promising neighboring solutions, which are then evaluated by the closed-form expression. The authors demonstrated the effectiveness of TABUSTOCH by comparing it with the exact algorithm on instances of size up to 46; the results showed that TABUSTOCH is about 3 orders of magnitude faster than the exact algorithm to reach an average solution cost that is not more than 5% away from the optimum. To the best of our knowledge, TABUSTOCH is the only metaheuristic that explicitly tackles the VRPSDC.

3.5 The vehicle routing problem with stochastic travel and service times

The VRPSTST is introduced by Laporte et al. (1992). In this problem, travel times between customers and service times at customers are described by random variables. The vehicles do not have capacity constraints and each customer requires being serviced. The authors proposed two formulations. A chance constrained program in which the probability of completing the tour within a given deadline is maximized and a stochastic program with recourse in which the expected cost is penalized according to the excess time taken by the vehicles. The authors used the Integer L-Shaped algorithm to solve optimally instances of size up to 20. Kenyon and Morton (2003) studied several properties and derived bounds on the optimal solutions. The authors proposed an estimation-based branch-and-cut algorithm and showed that on an instance of size 28, it can obtain solutions whose cost is less than 3% from the optimal solution value.

Early studies on the VRPSTST focused on a simple variant known as traveling salesman problem with stochastic travel times (TSPST). The goal is to determine an *a priori* solution that maximizes the probability of completing the tour within a given deadline. Kao (1978) discussed two heuristics based on dynamic programming and implicit enumeration. Sniedovich (1981) showed that the adoption of dynamic

programming might result in sub-optimal solutions. This issue was later addressed by Carraway et al. (1989) using generalized dynamic programming. Verweij et al. (2003) proposed an algorithm that combines the empirical estimation and decomposition based branch-and-cut method. The authors observed that this algorithm can achieve solutions whose cost is less than 1% away from the optimum.

The m-TSPST is the m vehicle version of the TSPST, where m is a decision variable. Lambert et al. (1993) studied the m-TSPST in the context of optimizing collection routes through bank branches in a network. The authors proposed mathematical programming formulations and designed a heuristic based on the Clarke-Wright savings algorithm.

3.6 Other extensions

The PTSP with deadlines is proposed and extensively studied by Campbell and Thomas (2008a, 2009). This is an extension of the PTSP, in which each probabilistic customer requires being visited before a given deadline. The salesman is allowed to visit customers after the deadline with a per-unit-time penalty. The authors studied two versions of a stochastic program with recourse and a chance constrained program for the problem. They proposed three approximation schemes. First, expected arrival time that approximates the closed-form expression by computing the deadline violations with respect to the expected arrival time to a customer. Second, temporal aggregation that considers large time units instead of the given time units. Third, truncation that approximates penalty terms. The PTSP iterative improvement algorithm, `1-shift` is customized to solve this problem by using the three schemes. The results showed that the three schemes significantly reduce the computation time, however, the best scheme depends on problem-specific attributes such as deadline, penalty, and instance size.

The vehicle routing problem with stochastic service time is introduced by Hadjiconstantinou and Roberts (2002). In this problem, each customer requires being visited. A number m of vehicles, each with a time limit, needs to serve the customers. The travel time is deterministic but each customer has a service time which is described by a random variable. Whenever a vehicle exceeds its time limit, it returns to the depot for replenishment. The authors considered a stochastic program with recourse, in which the goal is to find m *a priori* routes with minimal expected cost such that all service time requirements are met. The authors proposed a paired tree search algorithm, which is similar to the stochastic branch-and-bound algorithm, to obtain optimal solutions.

The TSP with time windows and stochastic service time is investigated by Gutjahr

(2004). It is a variant of the vehicle routing problem with stochastic service time, in which a salesman has to visit a number of customers. If the salesman arrives before the start of the time window or after the end time of the time window, a penalty proportional to the waiting time or to the tardiness is imposed. The author followed the typical stochastic program with recourse model, in which the goal is to find an *a priori* solution that minimizes the travel time and expected penalty. The author tackled this problem using an estimation-based ACO algorithm and a stochastic simulated annealing algorithm. The results showed that the former outperforms the latter.

The time-constrained TSP with stochastic travel and service times is studied by Teng et al. (2004). The problem consists of finding a partial *a priori* solution under given time constraints. The authors formulated the problem as a stochastic program with recourse and used the Integer L-shaped method for solving it.

3.7 Summary

In this chapter, we reviewed classical stochastic routing problems (SRPs) that arise in the context of distribution management. For each problem, we discussed exact methods and stochastic local search algorithms—in particular, constructive heuristics, iterative improvement algorithms, and metaheuristics. Among all SRPs, the literature on the VRPSD is the richest, followed by the PTSP. For other SRPs, the development of metaheuristics is still in its early stages.

The review presented in this chapter illustrates that the majority of metaheuristics proposed to tackle the SRPs use the analytical computation approach. However, in a large number of practical SRPs, due to the presence of complex constraints, deriving closed-form expressions to compute solution costs is difficult, time consuming or even impossible. Even in the classical SRPs that we discussed, the adoption of the analytical computation approach is computationally expensive. Several research results from the stochastic optimization literature suggest that the empirical estimation has the potential to overcome the difficulties posed by analytical computation. Surprisingly, to the best of our knowledge, the effectiveness of using the empirical estimation approach over the analytical computation approach to tackle SRPs has not yet been thoroughly investigated. In fact, the central theme of this thesis consists in addressing this issue.

Chapter 4

Estimation-based Local Search for the PTSP

As a first step to tackle the probabilistic traveling salesman problem (PTSP), we develop an estimation-based iterative improvement algorithm. A particularity of the proposed algorithm is that the cost difference between two neighbor solutions is *estimated* by delta evaluation. Moreover, the proposed algorithm exploits the well known TSP neighborhood reduction techniques to speedup the search. We empirically assess the performance of the proposed iterative improvement algorithm and provide in-depth analyses of each algorithmic component that we use. We compare the proposed algorithm with previous state-of-the-art iterative improvement algorithms, which use analytical computation for delta evaluation. We show that the estimation-based approach is an effective alternative to the analytical computation approach and that the proposed algorithm is more effective than the analytical computation iterative improvement algorithms.

This chapter is organized as follows. In Section 4.1, we give a formal definition of the PTSP. In Section 4.2, we describe the iterative improvement algorithms proposed in the literature for the PTSP. In Section 4.3, we introduce the estimation-based iterative improvement algorithm for the PTSP. In Section 4.4, we study its performance. This chapter ends with a summary of the results in Section 4.5.

4.1 The probabilistic traveling salesman problem

An instance of the PTSP is defined on a graph G with the following elements:

- a set $V = \{1, 2, \ldots, n\}$ of nodes;

- a set $A = \{\langle i, j\rangle : i, j \in V, i \neq j\}$ of edges, where an edge $\langle i, j\rangle$ connects nodes i and j;
- a set $C = \{c_{ij} : \langle i, j\rangle \in A\}$ of travel costs, where c_{ij} is the cost of traversing an edge $\langle i, j\rangle$; the costs are assumed to be symmetric, that is, for all pairs of nodes i, j we have $c_{ij} = c_{ji}$;
- a set $P = \{p_i : i \in V\}$ of probabilities, where p_i specifies the probability that a node i requires being visited. The events that two distinct nodes i and j require being visited are assumed to be independent.

The probabilistic data of the PTSP can be modeled using a random variable $\boldsymbol{\omega}$ that follows an n-variate Bernoulli distribution. A realization of $\boldsymbol{\omega}$ is a vector of binary values, where a value `'1'` in position i indicates that node i requires being visited, whereas a value `'0'` means that it does not require being visited. Note that when $P = \{p_i = p : i \in V\}$, the PTSP instance is called homogeneous, otherwise, if for at least two nodes i and j we have $p_i \neq p_j$, it is called heterogeneous. A solution to the PTSP is a permutation of the nodes.

Usually, the PTSP is tackled by *a priori* optimization with recourse (Jaillet, 1985; Bertsimas et al., 1990). It consists of two stages. In the first stage, a solution is determined before the actual realization of the random variable $\boldsymbol{\omega}$ is available. This is the so-called *a priori* solution. In the second stage, after the realization of the random variable is known, an *a posteriori* solution is obtained from the *a priori* solution by visiting the nodes prescribed by the given realization in the order in which they appear in the *a priori* solution. The nodes that do not require a visit are simply skipped. Figure 4.1 shows an example. It should be noted that, since we assume travel cost to be symmetric, the cost of the *a posteriori* solution is invariant with respect to the orientation. For example, in Figure 4.1, the cost of the *a posteriori* solution does not change by visiting the nodes in counter clockwise direction. The goal in the PTSP is to find an *a priori* solution that minimizes the expected cost of the *a posteriori* solution, where the expectation is computed with respect to a given n-variate Bernoulli distribution.

The analytical computation approach computes the cost $F(x)$ of an *a priori* solution $x = (\pi(1), \pi(2), \ldots, \pi(n), \pi(n+1) = \pi(1))$, where π is a permutation of the set V, using the following closed-form expression (Jaillet, 1985):

$$F(x) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{\pi(i)\pi(j)}\ p_{\pi(i)} p_{\pi(j)} \prod_{k=i+1}^{j-1} (1 - p_{\pi(k)})$$

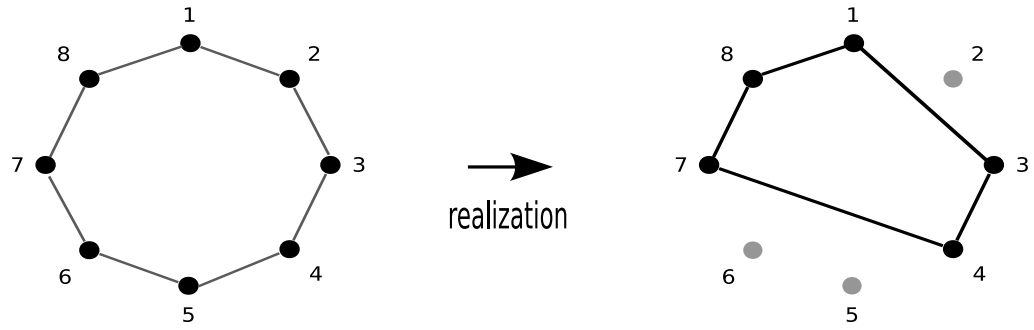


Figure 4.1: An *a priori* solution for a PTSP instance with 8 nodes. The nodes in the *a priori* solution are visited in the following order: 1, 2, 3, 4, 5, 6, 7, 8, and 1. Assume that a realization of $\boldsymbol{\omega}$ prescribes that nodes 1, 3, 4, 7, and 8 are to be visited. The resulting *a posteriori* solution is obtained by visiting the nodes in the order in which they appear in the *a priori* solution and by skipping the nodes 2, 5, and 6, which do not require being visited.

$$+ \sum_{j=1}^{n} \sum_{i=1}^{j-1} c_{\pi(j)\pi(i)} \; p_{\pi(i)} p_{\pi(j)} \prod_{k=j+1}^{n} (1 - p_{\pi(k)}) \prod_{k=1}^{i-1} (1 - p_{\pi(k)}). \tag{4.1}$$

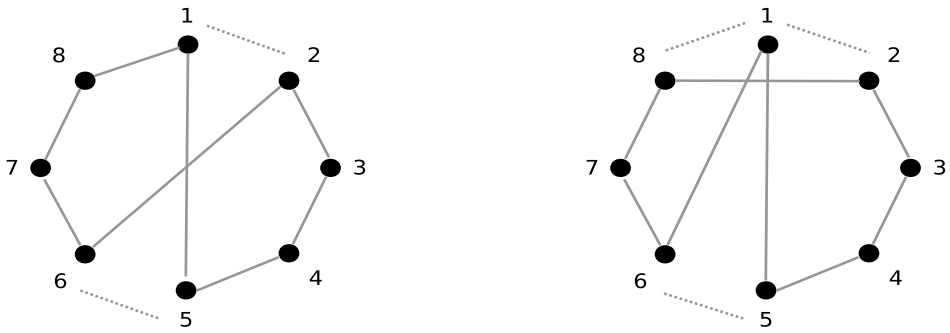
For the homogeneous PTSP, Equation 4.1 can be written as

$$F(x) = \sum_{i=1}^{n} \sum_{j=1}^{n-1} p^2 (1-p)^{j-1} \; c_{\pi(i),\pi(1+((i+j-1) \; mod \; n))},$$

where p is the probability value, which is common to all nodes, and *mod* is the modulo operator.

4.2 Local Search for the PTSP

Local search is a method for searching in a user-defined neighborhood structure. Many local search methods exist and the one that has received the most attention in the PTSP literature is iterative improvement. Iterative improvement algorithms start from some initial solution and repeatedly try to move from a current solution x to a lower cost neighboring solution x'. A solution that does not have any improving neighboring solution is a local minimum and the iterative improvement search terminates with such a solution. In the PTSP literature, mainly the following two neighborhood structures were considered:



(a) (b)

Figure 4.2: Plot 4.2(a) shows a 2-exchange move: two edges $\langle 1, 2\rangle$ and $\langle 5, 6\rangle$ are deleted and replaced by $\langle 1, 5\rangle$ and $\langle 2, 6\rangle$. Plot 4.2(b) shows a node-insertion move: node 1 is moved and inserted between nodes 5 and 6.

- ***2-exchange neighborhood***: The neighborhood of a solution is the set of solutions obtained by deleting any two edges $\langle a, b\rangle$ and $\langle c, d\rangle$ and by replacing them with $\langle a, c\rangle$ and $\langle b, d\rangle$. See Figure 4.2(a) for an example.

- ***Node-insertion neighborhood***: The neighborhood of a solution is the set of solutions obtained by deleting any node a and inserting it elsewhere in the solution. See Figure 4.2(b) for an example.

Iterative improvement algorithms can be implemented using a first-improvement or a best-improvement rule (Hoos and Stützle, 2005). Whereas in the former an improving move is immediately applied as soon as it is detected, in the latter the whole neighborhood is examined and a move that gives the best improvement is chosen.

Iterative improvement algorithms for the PTSP are similar to the usual iterative improvement algorithms for the TSP: the cost difference between two TSP neighboring solutions x and x' is computed by considering the cost contribution of solution components that are not common to x and x'. In the case of a 2-exchange move, the cost difference between the neighboring solutions is simply given by $c_{a,c} + c_{b,d} - c_{a,b} - c_{c,d}$. This technique is widely known as delta evaluation. The only difference between PTSP and TSP versions is that, in the former, the random variable $\boldsymbol{\omega}$ has to be taken into account in the delta evaluation. In the rest of this section, we describe how delta evaluation is performed in analytical computation iterative improvement algorithms for the PTSP.

4.2.1 `2-p-opt` and `1-shift`

The `2-p-opt` algorithm comprises two phases. A first phase consists of exploring a special case of the 2-exchange neighborhood, the swap-neighborhood, where the neighbors of the current solution are all those that can be obtained by swapping two consecutive nodes. If the swap-neighborhood is fully explored and no improvement is found, a second phase explores the 2-exchange neighborhood with the first-improvement rule. It should be noted that the neighborhood is explored in a fixed lexicographic order by considering pairs of edges that are separated by a fixed number k of nodes. Starting with $k = 2$, the lexicographic exploration proceeds by incrementing k, and, whenever an improvement is found, the search is restarted from the first phase: the swap-neighborhood exploration. Note that the swap-neighborhood is explored with the first-improvement rule. `1-shift` differs from `2-p-opt` only in the second phase: it uses the node-insertion neighborhood with the best-improvement rule.

We refer the reader to Bianchi et al. (2005) and to Bianchi and Campbell (2007) for the recursive delta evaluation expressions that are used in `2-p-opt` and `1-shift`. Even though the time complexity is $O(n^2)$ for both `2-p-opt` and `1-shift`, the asymptotic notation captures only the growth rates with respect to the number of neighboring solutions and does not reflect the multiplicative constant of 2. Moreover, the computational overhead due to the adoption of recursive delta evaluation expressions in `2-p-opt` and `1-shift` is typically high for large instances.

Before the development of the estimation-based iterative improvement algorithm discussed in this chapter, the iterative improvement algorithms for the PTSP used neighborhood-specific delta evaluation expressions that were based on analytical computation. The advantage of the adopted analytical computation approach is that the values of the computed cost differences are exact. However, from a practical perspective, this approach has some limitations.

4.2.2 Issues with the analytical computation iterative improvement algorithms for the PTSP

Theoretically, the delta evaluation expressions proposed by Bianchi et al. (2005) can be applied to solve PTSP instances of any size. However, in practice, these expressions suffer from numerical problems when applied to large instances. In fact, to use the delta evaluation expressions in `2-p-opt`, the term $(1-p)^{(k-n)}$ has to be computed, where p is the probability, k is the number of nodes between two considered edges, and n is the size of the instance. For some values of p, k, and n, this term can result in an overflow. As

Table 4.1: For a given probability p, the maximum size of the instance that can be handled by `2-p-opt` on a 32-bit system without resorting to arbitrary precision arithmetics.

p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$n_{critical}$	6733	3180	1990	1390	1025	775	591	442	309

an illustration, consider a typical 32-bit GNU system, where, according to the IEEE 754 standard (1985), double precision floating point variables can take a maximum value of about `1e+308` (Griffith, 2002). Given a homogeneous PTSP instance of probability p with n nodes, the condition for the numerical overflow is $(1-p)^{(k-n)} >$ `1e+308`. From this condition, one can obtain, after basic transformations, a critical value for n, above which the computation of the delta evaluation expression suffers from precision problems: $n_{critical} = 1 - \frac{308}{\log_{10}(1-p)}$. Table 4.1 shows the probability levels and the corresponding maximum size of instances that can be tackled by `2-p-opt` without any numerical overflow on a 32-bit system. The very same problem occurs in `1-shift` and in the analytical computation algorithms for the heterogeneous PTSP. This issue has to be addressed by resorting to methods for arbitrary precision arithmetics. As we show in section 4.4.3, this might entail a major computational overhead.

A second issue with the analytical computation iterative improvement algorithms for the PTSP is that the lexicographic neighborhood exploration does not allow the adoption of the classical TSP neighborhood reduction techniques such as fixed-radius search, candidate lists and don't look bits (Martin et al., 1991; Bentley, 1992). Based on results from the TSP literature (Johnson and McGeoch, 1997; Hoos and Stützle, 2005), we speculate that the usage of the neighborhood reduction techniques in the PTSP iterative improvement algorithms might speed up the search significantly.

4.3 Estimation-based iterative improvement algorithms

The cost $F(x)$ of a PTSP solution x can be empirically estimated on the basis of a sample $f(x,\omega_1), f(x,\omega_2), \ldots, f(x,\omega_M)$ of costs of *a posteriori* solutions obtained from M independent realizations $\omega_1, \omega_2, \ldots, \omega_M$ of the random variable $\boldsymbol{\omega}$:

$$\hat{F}_M(x) = \frac{1}{M} \sum_{r=1}^{M} f(x,\omega_r). \tag{4.2}$$

As it can be shown easily, $\hat{F}_M(x)$ is an *unbiased* estimator of $F(x)$.

In iterative improvement algorithms for the PTSP, we need to compare two neigh-

boring solutions x and x' to select the one of lower cost. This can be achieved by determining the sign of the cost difference $F(x') - F(x)$. For x', an unbiased estimator $\hat{F}_{M'}(x')$ of $F(x')$ can be obtained from a sample $f(x', \omega'_1), f(x', \omega'_2), \ldots, f(x', \omega'_{M'})$ of costs of *a posteriori* solutions through M' independent realizations of $\boldsymbol{\omega}$. Also, $\hat{F}_{M'}(x') - \hat{F}_M(x)$ is an unbiased estimator of $F(x') - F(x)$.

In order to increase the accuracy of this estimator, the well-known variance-reduction technique called *the method of common random numbers* can be adopted. In the context of the PTSP, this technique consists of using the same set of realizations of $\boldsymbol{\omega}$ for estimating the costs $F(x')$ and $F(x)$. Consequently, we have $M' = M$ and the estimator $\hat{F}_M(x') - \hat{F}_M(x)$ of the cost difference is given by:

$$\begin{aligned} \hat{F}_M(x') - \hat{F}_M(x) &= \frac{1}{M}\sum_{r=1}^{M} f(x', \omega_r) - \frac{1}{M}\sum_{r=1}^{M} f(x, \omega_r) \\ &= \frac{1}{M}\sum_{r=1}^{M} \Big(f(x', \omega_r) - f(x, \omega_r) \Big). \end{aligned} \tag{4.3}$$

In this chapter, we use the same set of M realizations for all steps of the iterative improvement algorithms. Other approaches could be adopted: for example, M realizations could be sampled anew for each step of the algorithm or even for each comparison. A discussion about this issue is given in Section 4.4.4.

Using Equation 4.3, given two neighboring solutions x and x' and a realization ω, a straightforward approach to compute the cost difference between two *a posteriori* solutions consists of computing first the complete cost of each *a posteriori* solution and then the difference between them. However, a more efficient algorithm can be obtained by adopting the idea of delta evaluation. Given the *a priori* solutions and a realization ω, such an algorithm requires identifying the edges that are not common to the two *a posteriori* solutions.

For example, consider the 2-exchange move shown in Figure 4.3. The edges that are not common to the *a priori* solutions are $\langle 1, 2\rangle$, $\langle 5, 6\rangle$ and $\langle 1, 5\rangle$, $\langle 2, 6\rangle$. For a realization prescribing that nodes 1, 3, 4, 7, and 8 are to be visited, the edges that are not common to the *a posteriori* solutions are $\langle 1, 3\rangle$, $\langle 4, 7\rangle$ and $\langle 1, 4\rangle$, $\langle 3, 7\rangle$. Therefore, the cost difference between the two *a posteriori* solutions is given by $c_{1,4} + c_{3,7} - c_{1,3} - c_{4,7}$. The delta evaluation procedure needs to identify these edges in the *a posteriori* solutions.

In general, for every edge $\langle i, j\rangle$ that is deleted from x, one needs to find the corresponding edge $\langle i^*, j^*\rangle$ in the *a posteriori* solution. We call this edge the *a-posteriori-edge*. It is obtained as follows. If node i has to be visited, then $i^* = i$, otherwise, i^*

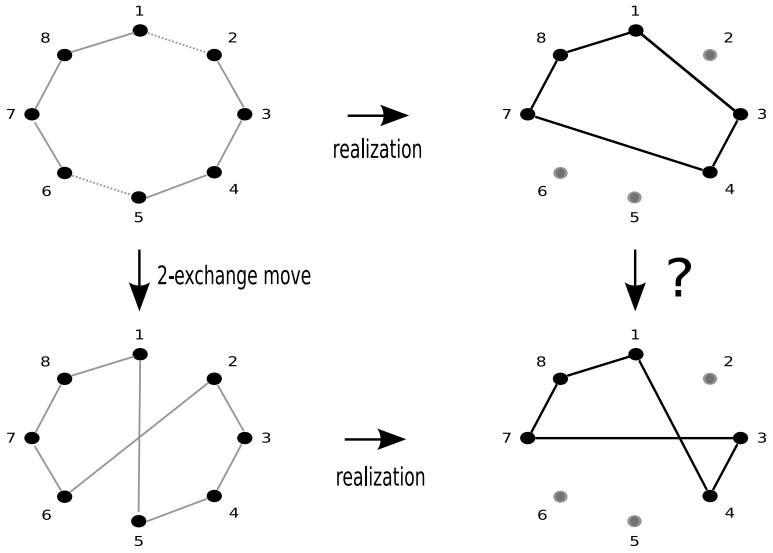


Figure 4.3: In this example, a 2-exchange move is obtained by replacing $\langle 1, 2\rangle$ and $\langle 5, 6\rangle$ in the *a priori* solution with $\langle 1, 5\rangle$ and $\langle 2, 6\rangle$. Assume that a realization of $\boldsymbol{\omega}$ prescribes that nodes 1, 3, 4, 7, and 8 are to be visited. The edges that are not common to the *a posteriori* solutions are $\langle 1, 3\rangle$, $\langle 4, 7\rangle$ and $\langle 1, 4\rangle$, $\langle 3, 7\rangle$. The delta evaluation procedure needs to identify these edges without considering the complete *a posteriori* solutions.

is the first predecessor of i in x such that $\omega[i^*] = 1$. If node j has to be visited, then $j^* = j$, otherwise, j^* is the first successor of j such that $\omega[j^*] = 1$. Recall that in a 2-exchange move, the edges $\langle a, b\rangle$ and $\langle c, d\rangle$ are replaced by $\langle a, c\rangle$ and $\langle b, d\rangle$. For the *a posteriori* edges $\langle a^*, b^*\rangle$ and $\langle c^*, d^*\rangle$, the cost difference between the two *a posteriori* solutions is $c_{a^*,c^*} + c_{b^*,d^*} - c_{a^*,b^*} - c_{c^*,d^*}$. Figure 4.4 shows the *a posteriori* edges for the example given in Figure 4.3. This procedure can be directly extended to the node-insertion move. See Figure 4.5 for an example.

It is worth discussing here some degenerate cases: in a 2-exchange move that deletes edges $\langle a, b\rangle$ and $\langle c, d\rangle$, and where no node between the nodes b and c or between nodes a and d requires being visited, the difference between the two *a posteriori* solutions is zero—see Figure 4.6(a); in a node-insertion move, if the insertion node does not require being visited, then the cost difference between the two *a posteriori* solutions is zero—see Figure 4.6(b). In this second case, one can avoid unnecessary computations by checking whether the insertion node requires being visited before finding the *a posteriori* edges.

The proposed approach has a number of advantages. First, the estimation-based delta evaluation procedure, in particular, the procedure for finding *a posteriori* edges can be applied to any neighborhood structure without requiring the complex mathematical derivations that have to be performed when adopting the analytical computation.

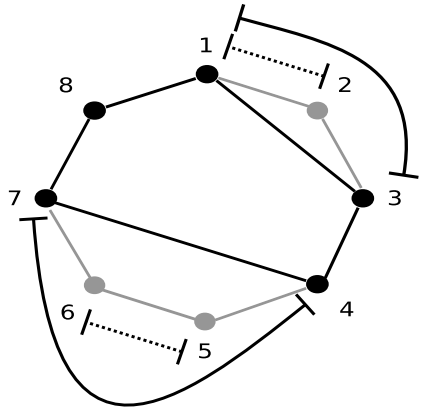


Figure 4.4: The steps performed for finding the *a posteriori* edges. Assume that, the nodes are visited in the order 1, 2, 3, 4, 5, 6, 7, 8, and 1. The edges $\langle 1, 2\rangle$ and $\langle 5, 6\rangle$ are deleted and the gray nodes do not require being visited. The first successor of node 2 that requires being visited is 3; the first predecessor of node 5 that requires being visited is 4; the first successor of node 6 that requires being visited is 7. The *a posteriori* edges are therefore $\langle 1, 3\rangle$ and $\langle 4, 7\rangle$.

In virtue of this versatility, rather than using the node-insertion neighborhood or the 2-exchange neighborhood structure, we can use a hybrid neighborhood structure that includes the node-insertion neighborhood on top of the 2-exchange neighborhood structure. In the TSP literature (Bentley, 1992), this hybrid neighborhood is widely known as the 2.5-exchange neighborhood: when checking for a 2-exchange move on any two edges $\langle a, b\rangle$ and $\langle c, d\rangle$, it is also checked whether deleting any one of the nodes of an edge, say for example a, and inserting it between nodes c and d results in an improved solution (Bentley, 1992).

Second, unlike `2-p-opt` and `1-shift`, the proposed approach does not impose any constraints on the order in which the neighborhood should be explored. This allows for an easy integration of the classical TSP neighborhood reduction techniques such as fixed-radius search, candidate lists, and don't look bits (Martin et al., 1991; Bentley, 1992; Johnson and McGeoch, 1997). Note that the candidate list is a static data structure that contains for each node, a number L of closest nodes, ordered by increasing cost. The algorithm considers only the moves that involve a given node and one of its closest nodes in the list.

We denote the proposed algorithm `2.5-opt-EEs` where `EE` and `s` stand for empirical estimation and speedup, respectively. Note that `2.5-opt-EEs` uses the first-improvement rule. In order to implement `2.5-opt-EEs` effectively, a specific data structure is needed. We use a structure in which data items can be accessed both

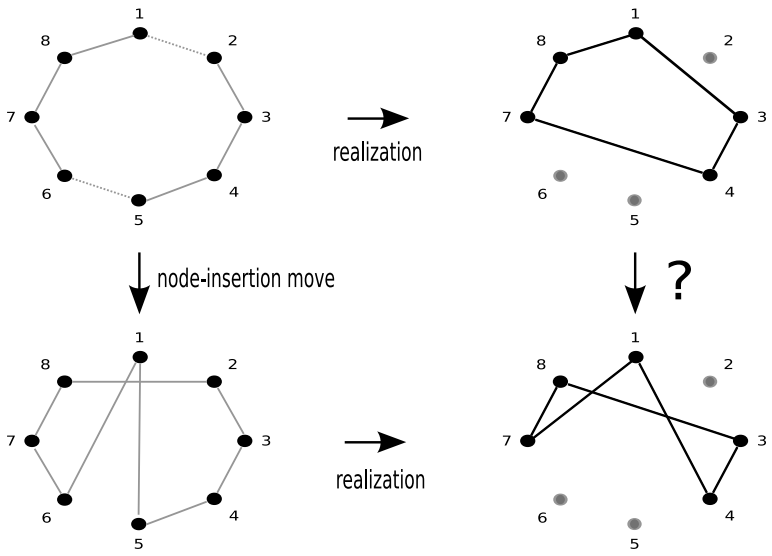
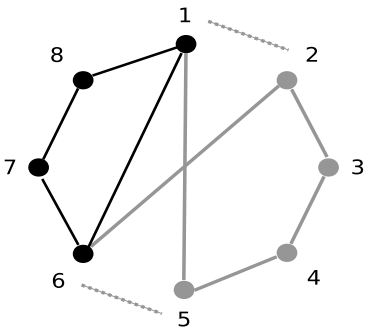
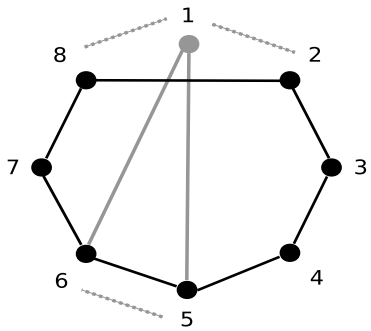


Figure 4.5: In this example, the node-insertion move is obtained by inserting node 1 between nodes 5 and 6. Consequently, the edges $\langle 8,1\rangle$, $\langle 1,2\rangle$, and $\langle 5,6\rangle$ in the *a priori* solution are replaced with $\langle 8,2\rangle$, $\langle 5,1\rangle$, and $\langle 1,6\rangle$. Assume that a realization of $\boldsymbol{\omega}$ prescribes that nodes 1, 3, 4, 7, and 8 are to be visited. The edges that are not common to the *a posteriori* solutions are $\langle 1,3\rangle$, $\langle 4,7\rangle$, $\langle 8,1\rangle$ and $\langle 8,3\rangle$, $\langle 4,1\rangle$, $\langle 1,7\rangle$.



(a) Assume that a realization of $\boldsymbol{\omega}$ prescribes that nodes 1, 6, 7, and 8 are to be visited. The 2-exchange neighboring solutions shown in Figure 4.2(a) lead to the same *a posteriori* solution. The cost difference is therefore zero.



(b) Assume that a realization of $\boldsymbol{\omega}$ prescribes that nodes 2, 3, 4, 5, 6, 7, and 8 are to be visited. The node-insertion neighboring solutions shown in Figure 4.2(b) lead to the same *a posteriori* solution. Since the two *a posteriori* solutions are the same, the cost difference is zero.

Figure 4.6: Some degenerate cases that can occur in the evaluation of cost differences.

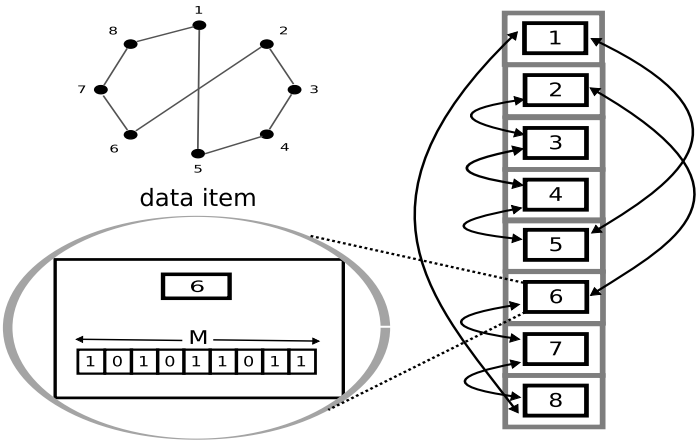


Figure 4.7: Assume that we have an *a priori* solution in which the nodes are visited in the order 1, 5, 4, 3, 2, 6, 7, 8, 1. This is encoded in the doubly circularly linked list data structure as shown in the plot. Also note that the data items can be accessed as the elements of the one dimensional array.

as elements of a doubly circularly linked list and as elements of a one dimensional array, both of size n. Each data item comprises an integer variable to store a node of the *a priori* solution. This structure is efficient for finding *a posteriori* edges: predecessor and successor of a node are simply obtained by following the links pointing towards the previous item and next item, respectively. To access data items as elements of the array, a data item representing node i is stored at position i of the array and this arrangement is kept unchanged throughout the search process. Consequently, given a node i, its data can be accessed in $O(1)$ time. Moreover, each data item stores an array of size M—the realization array—which is indexed from 1 to M. Element r of the realization array is either `1` or `0` indicating whether the node requires being visited or not in realization ω_r. Figure 4.7 shows the data structure. Whenever, an improved solution is found, only the links of the particular data items whose nodes are involved in the exchange move are modified. Furthermore, each data item comprises also two auxiliary fields for the neighborhood reduction techniques: one integer variable for the don't look bit and one integer array of size L for the candidate list of each node.

Concerning the computational complexity of the estimation-based iterative improvement algorithm, Bianchi (2006) reached the conclusion that the time complexity of a complete neighborhood scan is $O(Mpn^3)$. Indeed, the number of solutions in the 2.5-exchange neighborhood is $O(n^2)$; the maximum number of steps for finding the *a posteriori* edges for a given realization is n; and the number of realizations considered is M. The author included in the complexity also the probability p that a node requires being

visited, but the inclusion of this term is not completely justified and is not thoroughly discussed in the authors' work (Bianchi, 2006). The main result of the analysis is that the time complexity grows with the cube of n. On the basis of this result, the author decided that the estimation-based approach does not deserve any further attention. However, the above analysis does not hold for `2.5-opt-EEs`. The use of a candidate list of size L reduces the neighborhood size from $O(n^2)$ to $O(nL)$, which in turn reduces the worst-case time complexity of a neighborhood scan to $O(n^2LM)$. It should be noted that in large TSP instances, the value of L is typically chosen independent of the instance size and is usually set between 20 and 40 (Johnson and McGeoch, 1997). We expect that these conclusions hold also for the PTSP. Furthermore, it should be observed that, since we explicitly deal with a probabilistic model, a more informative average-case analysis can be derived easily. Let Y be a random variable that describes the number of steps required for finding the first successor in an *a posteriori* edge. Since the probability of finding the first successor at each step is p, the probability that there are k failures before finding the first successor is $\Pr(Y = k) = (1-p)^k p$. From this, we can obtain the expected value $E(Y)$ of Y as $(1-p)/p$. Thus, the expected number of steps for finding the first successor is $(1-p)/p$. This is same for finding the first predecessor. As a result, the average-case time complexity of a complete neighborhood scan is $O(nLMp^{-1})$.

4.4 Experimental analysis

We base our analysis on homogeneous PTSP instances that we obtained from TSP instances generated with the DIMACS instance generator (Johnson et al., 2001). We carried out experiments with two classes of instances. In the first class, nodes are uniformly distributed in a square; in the second, nodes are arranged in clusters. For each instance class, we generated 100 TSP instances of 100, 200, 300, and 1000 nodes. Note that 300 is the largest instance size that has been used by Bianchi (2006). From each TSP instance, we obtain 9 PTSP instances by letting the probability range in $[0.1, 0.9]$ with a step size of 0.1. We report the results obtained on the clustered instances of 300 and 1000 nodes for certain probability levels. The general trends of the experimental results obtained on instance with uniformly distributed nodes are similar; the complete set of results is given as a supplementary page (Birattari et al., 2007) at the following URL:

`http://iridia.ulb.ac.be/supp/IridiaSupp2007-001/`

All algorithms are implemented in `C` and the source codes are compiled with `gcc`, version 3.3. Experiments were carried out on AMD Opteron^{TM}244 1.75GHz processors with 1 MB L2-Cache and 2 GB RAM, running under Debian GNU/Linux.

The nearest-neighbor heuristic is used to generate initial solutions. The candidate list is set to size 40 and it is constructed with the quadrant nearest-neighbor strategy (Penky and Miller, 1994; Johnson and McGeoch, 1997). For each node i, a coordinate system is defined with origin in node i. The candidate list of node i then contains for each quadrant the 10 cities that are connected to i by the 10 edges of least cost. If fewer than 10 nodes are available in a quadrant, the list is filled with nodes from other quadrants. Each iterative improvement algorithm is run until it reaches a local optimum. The number of realizations in `2.5-opt-EEs` is set to 100. In order to highlight this fact, we denote the algorithm `2.5-opt-EEs-100`. We use Equation 4.1 for the post-evaluation of the best-so-far solutions found by each algorithm according to its evaluation procedure.

In addition to tables, we visualize the results using runtime development plots. These plots show how the cost of solutions develops over computation time. In these plots, the x-axis indicates computation time and the y-axis indicates the cost of the solutions found, averaged over 100 instances. For comparing several algorithms, one of them has been taken as a reference: for each instance, the computation time and the cost of the solutions of the algorithms are normalized by the average computation time and average cost of the local optima obtained by the reference algorithm. For convenience, the x-axis is in logarithmic scale. We report one such plot for each probability level under consideration.

4.4.1 Experiments on neighborhood reduction techniques

Before presenting the results of `2.5-opt-EEs`, we first show that the adoption of the 2.5-exchange neighborhood structure and the classical TSP neighborhood reduction techniques in the analytical computation framework leads to an effective analytical computation iterative improvement algorithm for the PTSP. We denote this new iterative improvement algorithm as `2.5-opt-ACs` where `AC` and `s` stand for analytical computation and speedup, respectively. The motivation behind these experiments is the following: if we compared `2.5-opt-EEs` with `2-p-opt` and `1-shift`, it would be difficult to clearly identify whether the observed differences are due to the estimation-based delta evaluation procedure or rather to the adoption of the 2.5-exchange neighborhood and the neighborhood reduction techniques. Therefore, we implemented an

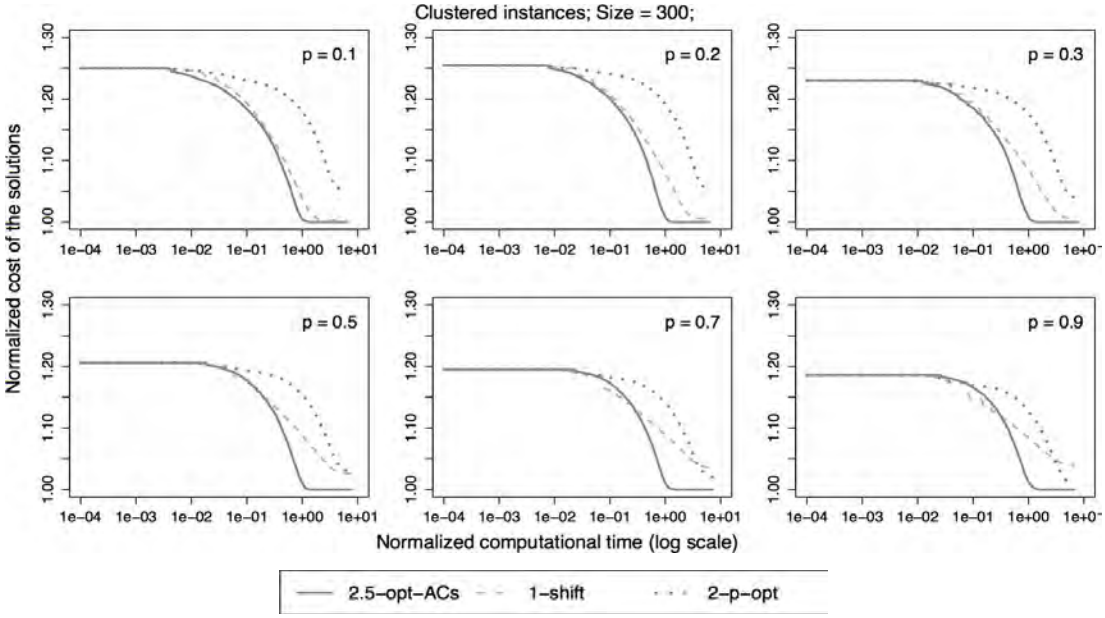


Figure 4.8: Experimental results on clustered homogeneous PTSP instances of size 300. The plots represent the average cost of the solutions obtained by `2-p-opt` and `1-shift` normalized by the one obtained by `2.5-opt-ACs`. Each algorithm is stopped when it reaches a local optimum. The normalization is done on an instance by instance basis for 100 instances; the normalized solution cost and the computation time are then aggregated. Note that the x-axis shows the computation time in logarithmic scale.

iterative improvement algorithm based on analytical computation that uses the 2.5-exchange neighborhood and the neighborhood reduction techniques, and compared its performance to `2-p-opt` and `1-shift`.

A difficulty in the implementation of `2.5-opt-ACs` is that, since the use of neighborhood reduction techniques prevents lexicographic exploration, the previously computed values cannot be reused. Therefore, the cost difference between two solutions is always computed from scratch. In order to compute the cost difference between 2.5-exchange neighboring solutions, we use the closed-form expressions proposed for the 2-exchange and the node-insertion neighborhood structures (Bianchi, 2006).

The results given in Figure 4.8 show that `2.5-opt-ACs` *dominates* `2-p-opt` and `1-shift` with the only exception being for the values of p ranging between 0.5 and 0.9: in the *early stages* of the search and for a very short time range, the average cost of the solutions obtained by `1-shift` is slightly lower than that of `2.5-opt-ACs`. Concerning the time required to reach local optima, irrespective of the probability value, `2.5-opt--ACs` is faster than `2-p-opt` by approximately a factor of four. In the case of `1-shift`, the same tendency holds when $p \geq 0.5$. However, for small values of p, the difference

in speed between 2.5-opt-ACs and 1-shift is small. Concerning the average cost of the local optima found, 2.5-opt-ACs is between 2% and 5% better than 2-p-opt. We can observe the same trend also in 1-shift; an exception is for $p \leq 0.3$, where the difference between the average cost of the local optima obtained by 2.5-opt-ACs and 1-shift is very small. For details, see Table 4.2 on page 63

In order to test that the observed differences between the cost of local optima are significant in a statistical sense, we use a t-test. The cost of the local optima obtained by 2.5-opt-ACs is significantly lower than that of 1-shift and 2-p-opt for all probability values, the only exception being $p \leq 0.2$, where the difference between the cost of the local optima obtained by 2.5-opt-ACs and 1-shift is not significant.

The increased speed of 2.5-opt-ACs also shows that the amount of computational time saved due to the use of neighborhood reduction techniques is much higher than the time that is lost in computing the cost difference from scratch. Regardless the values of p, with respect to the cost of the local optima and the computation time, 2.5-opt-ACs is better than—and in very few cases comparable with—1-shift and 2-p-opt. Therefore, in the following sections, we take 2.5-opt-ACs as a yardstick for measuring the effectiveness of 2.5-opt-EEs.

4.4.2 Experiments to assess the estimation-based approach

In this section, we compare 2.5-opt-EEs-100 with 2.5-opt-ACs. The two algorithms differ only in the delta evaluation procedure they adopt: empirical estimation versus analytical computation. The experimental results are illustrated using runtime development plots and are shown in Figure 4.9.

Concerning the average cost of local optima, the two algorithms are similar with the only exception of $p = 0.1$, where the average cost of the local optima obtained by 2.5-opt-EEs-100 is approximately 2% higher than that of 2.5-opt-ACs. Concerning the time required to reach local optima, irrespective of the probability value, 2.5--opt-EEs-100 is approximately 1.5 orders of magnitude faster than 2.5-opt-ACs. See Table 4.2 for the absolute values. The poorer solution cost of 2.5-opt-EEs-100 for $p = 0.1$ can be attributed to the fact that the number of realizations used to estimate the cost difference between two solutions is too small. Intuitively, the variance of the cost difference estimator with respect to the mean depends on p and M: the smaller the value of p, the higher the variance with respect to the mean. For $p = 0.1$ and $M = 100$, the variance with respect to the mean is very high, which eventually results in a misleading estimation of the cost difference between two solutions. As a consequence,

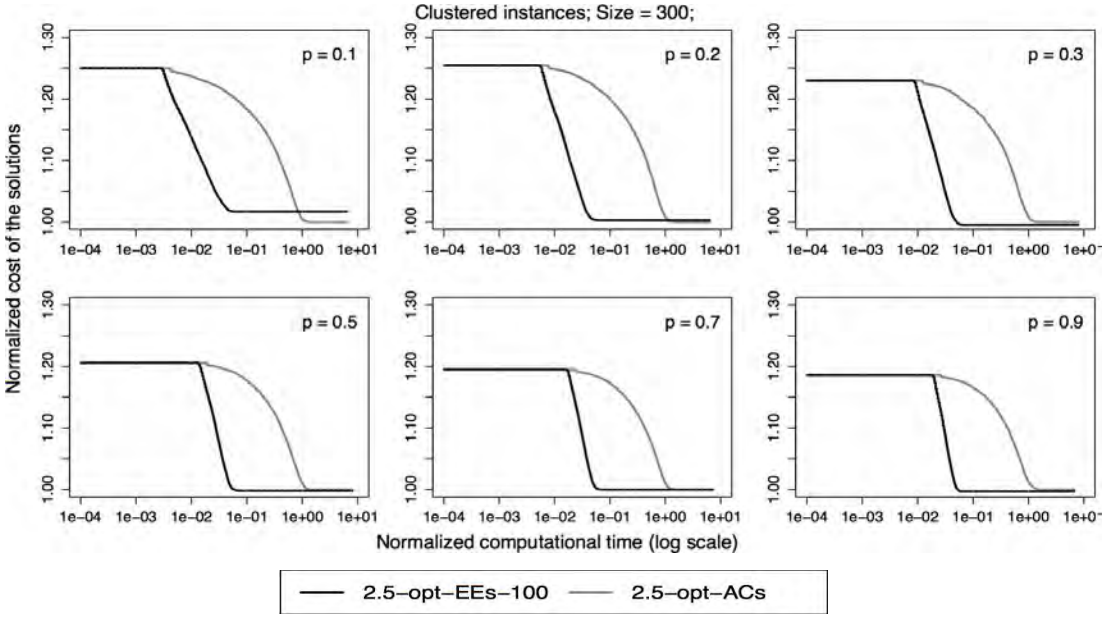


Figure 4.9: Experimental results on clustered homogeneous PTSP instances of size 300. The plots represent the average cost of the solutions obtained by `2.5-opt-EEs-100` normalized by the one obtained by `2.5-opt-ACs`. Each algorithm is stopped when it reaches a local optimum. The normalization is done on an instance by instance basis for 100 instances; the normalized solution cost and the computation time are then aggregated. Note that the x-axis shows the computation time in logarithmic scale.

`2.5-opt-EEs-100` stops prematurely.

In Table 4.3, we report the observed relative difference between the cost of the local optima obtained by the two algorithms and a 95% confidence interval of the relative difference (obtained from interval estimation through a t-test). For the sake of completeness, we also present these data for what concerns the comparison of `2.5-opt-EEs-100` with `1-shift` and `2-p-opt`.

Table 4.3 confirms that, concerning the average cost of the local optima found, `2.5-opt-EEs-100` is either slightly worse (for $p = 0.1$) or essentially equivalent to `2.5-opt-ACs` (for $p > 0.1$). To be more precise, for $p = 0.1$, `2.5-opt-EEs-100` has obtained solutions, the average cost of which is higher than the one of those obtained by `2.5-opt-ACs`. The difference is significant in a statistical sense but it is nonetheless relatively small: with a confidence of 95%, we can state that the expectation of the costs of the solutions obtained by `2.5-opt-EEs-100`, on the class of instances under analysis, is at most 1.98% higher than the one of `2.5-opt-ACs`.

For $p = 0.2$, the observed average cost obtained by `2.5-opt-EEs-100` is slightly higher than the one of `2.5-opt-ACs` but the difference is not significant in a statistical

Table 4.2: Experimental results for `2.5-opt-EEs-100`, `2.5-opt-ACs`, `2-p-opt` and `1-shift` on clustered instances of size 300. Each algorithm is allowed to run until it reaches a local optimum. The table gives mean and standard deviation (s.d.) of final solution cost and computation time in seconds. The results are obtained on 100 instances for each probability level.

	Algorithm	Solution Cost		Computation Time	
		mean	s.d.	mean	s.d.
$p = 0.1$	`2.5-opt-EEs-100`	2776865	456487	0.120	0.014
	`2.5-opt-ACs`	2730221	454321	6.453	1.067
	`1-shift`	2738026	450970	11.771	2.404
	`2-p-opt`	2870013	462383	22.440	5.831
$p = 0.2$	`2.5-opt-EEs-100`	3595283	467721	0.086	0.011
	`2.5-opt-ACs`	3585254	471967	3.413	0.540
	`1-shift`	3606878	467069	10.103	1.773
	`2-p-opt`	3775106	474269	13.848	3.254
$p = 0.3$	`2.5-opt-EEs-100`	4239788	499001	0.064	0.008
	`2.5-opt-ACs`	4259032	501810	2.214	0.399
	`1-shift`	4286461	481061	8.478	1.856
	`2-p-opt`	4429328	497857	9.842	2.462
$p = 0.5$	`2.5-opt-EEs-100`	5190835	537186	0.046	0.005
	`2.5-opt-ACs`	5201221	557895	1.421	0.230
	`1-shift`	5336979	544328	5.933	1.355
	`2-p-opt`	5352456	553028	5.978	1.616
$p = 0.7$	`2.5-opt-EEs-100`	5874044	579475	0.038	0.004
	`2.5-opt-ACs`	5875100	579015	1.127	0.209
	`1-shift`	6087249	597356	4.851	1.056
	`2-p-opt`	5993481	589339	4.776	1.146
$p = 0.9$	`2.5-opt-EEs-100`	6412335	602907	0.036	0.004
	`2.5-opt-ACs`	6428845	602878	1.020	0.220
	`1-shift`	6683735	628833	4.112	0.937
	`2-p-opt`	6491451	604388	4.093	0.954

sense: with 95% confidence, we can state that the expectation of the costs of the solutions obtained by `2.5-opt-EEs-100` is not more than 0.61% higher than the one of `2.5-opt-ACs`. For probabilities larger than 0.2, in our experiments, `2.5-opt-EEs-100` has obtained in average slightly better results, even if not in a statistically significant way. Should ever the expectation of the costs obtained by `2.5-opt-EEs-100`, on the class of instances under analysis, be larger than the one of the costs obtained by `2.5--opt-ACs`, their difference would be at most 0.31% for all values of probabilities larger than 0.2.

Similar conclusions can be drawn for what concerns the comparison between `2.5--opt-EEs-100` and `1-shift`. For $p = 0.1$, `1-shift` obtains a lower average cost than that of `2.5-opt-EEs-100`. The difference is significant in a statistical sense but it is

Table 4.3: Comparison of the average cost obtained by `2.5-opt-EEs-100` and by `2.5-opt-ACs`, `1-shift`, and `2-p-opt`, on clustered instances of size 300. Each algorithm is allowed to run until it reaches a local optimum. For a given comparison A vs. B, the table reports the observed relative difference between the two algorithms A and B and a 95% confidence interval (CI) obtained through the t-test. If the value is positive, algorithm A obtained an average cost that is larger than the one obtained by algorithm B. In this case, the value is typeset in italics if it is significantly different from zero according to the t-test at a confidence level of 95%. If the value is negative, algorithm A obtained an average cost that is smaller than the one obtained by algorithm B. In this case, the value is typeset in boldface if it is significantly different from zero according to the t-test at a confidence level of 95%.

	`2.5-opt-EEs-100` vs. `2.5-opt-ACs`		`2.5-opt-EEs-100` vs. `1-shift`		`2.5-opt-EEs-100` vs. `2-p-opt`	
p	Difference	95% CI	Difference	95% CI	Difference	95% CI
0.1	$+\mathbf{1.71}\%$	$[+1.438, +1.98]\%$	$+\mathbf{1.42}\%$	$[+1.00, +1.84]\%$	$-\mathbf{3.25}\%$	$[-3.60, -2.90]\%$
0.2	$+0.28\%$	$[-0.054, +0.61]\%$	-0.32%	$[-0.84, +0.19]\%$	$-\mathbf{4.76}\%$	$[-5.22, -4.31]\%$
0.3	-0.45%	$[-0.938, +0.03]\%$	$-\mathbf{1.09}\%$	$[-1.65, -0.53]\%$	$-\mathbf{4.28}\%$	$[-4.71, -3.86]\%$
0.5	-0.20%	$[-0.672, +0.28]\%$	$-\mathbf{2.74}\%$	$[-3.25, -2.22]\%$	$-\mathbf{3.02}\%$	$[-3.50, -2.52]\%$
0.7	-0.02%	$[-0.367, +0.31]\%$	$-\mathbf{3.50}\%$	$[-4.05, -2.98]\%$	$-\mathbf{1.99}\%$	$[-2.43, -1.58]\%$
0.9	-0.26%	$[-0.660, +0.05]\%$	$-\mathbf{4.06}\%$	$[-4.58, -3.63]\%$	$-\mathbf{1.22}\%$	$[-1.65, -0.88]\%$

relatively small: the expectation of the costs obtained by `2.5-opt-EEs-100` is within a bound of 1.84% of the one of `1-shift`. For $p = 0.2$, the difference between the two algorithms is not significant and the expectation of the costs obtained by `2.5-opt-EEs-100` is not more than 0.19% higher than the one of `1-shift`. For probabilities larger than 0.2, `2.5-opt-EEs-100` performs significantly better than `1-shift`; the expectation of the costs of the solutions obtained by `2.5-opt-EEs-100` is between at least 0.53% (for $p = 0.3$) and at least 3.63% (for $p = 0.9$) lower that the one of `2.5-opt-ACs`.

Concerning the last comparison, `2.5-opt-EEs-100` is significantly better than `2-p-opt` across the whole range of probabilities. The expected improvement obtained by `2.5-opt-EEs-100` ranges roughly between 1% and 4%.

In order to highlight the impact of the speed factor of `2.5-opt-EEs-100` on the cost of the solutions, we can analyze the cost of the solutions obtained by `2.5-opt-ACs` in the time needed by `2.5-opt-EEs-100` to find the local optima: From the results, irrespective of the value of p, we can observe that the average cost of the solutions obtained by `2.5-opt-EEs-100` is between 16% and 18% lower than that of `2.5-opt-ACs`. Clearly, the speed factor gives `2.5-opt-EEs-100` a significant advantage over `2.5-opt-ACs`. Even though `2.5-opt-ACs` and `2.5-opt-EEs-100` adopt the same neighborhood exploration and neighborhood reduction techniques, `2.5-opt-EEs-100`

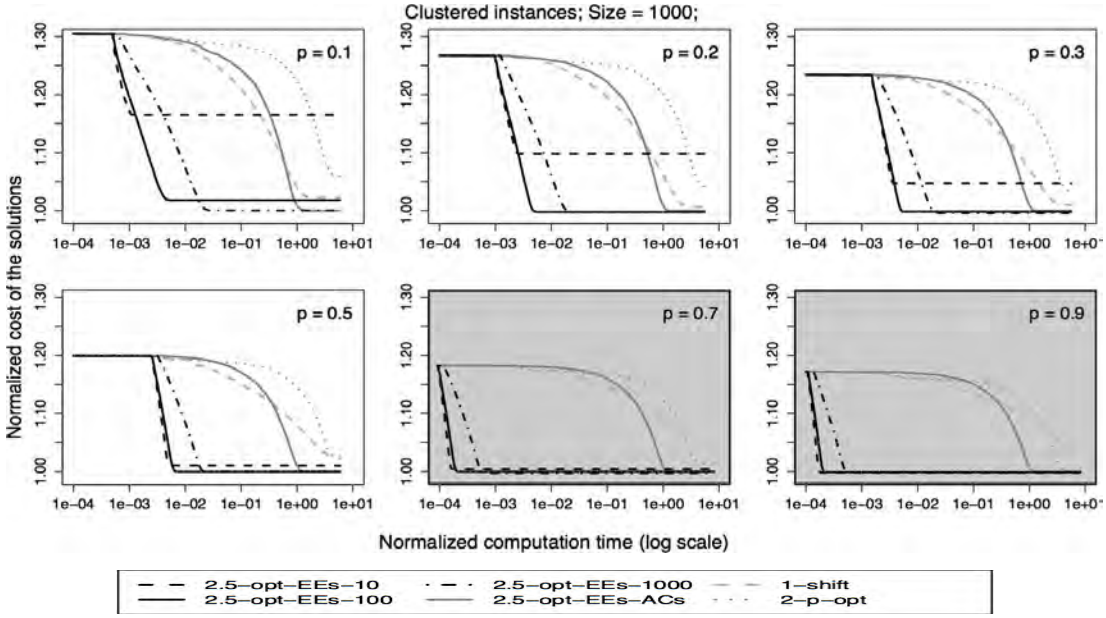


Figure 4.10: Experimental results on clustered homogeneous PTSP instances of size 1000. The plots represent the cost of the solutions obtained by `2.5-opt-EEs-10`, `2.5-opt-EEs-100`, `2.5-opt-EEs-1000`, `1-shift`, and `2-p-opt` normalized by the one obtained by `2.5-opt-ACs`. Each algorithm is stopped when it reaches a local optimum. The normalization is done on an instance by instance basis for 100 instances; the normalized solution cost and the computation time are then aggregated. Note that the x-axis shows the computation time in logarithmic scale. For $p > 0.5$ the algorithms based on the analytical computation use a library for arbitrary precision arithmetics. To emphasize this fact the background of the last two plots is gray.

is faster, due to the simplicity of the estimation-based delta evaluation procedure.

4.4.3 Experiments on large instances

In this section, we study the performance of `2.5-opt-EEs-100` when applied to large instances. For this purpose, we considered PTSP instances with 1000 nodes. In these experiments, for $p > 0.5$, `2.5-opt-ACs`, `1-shift`, and `2-p-opt` use MPFR (Fousse et al., 2007), a state-of-the-art library for arbitrary precision arithmetics to handle the numerical issue discussed in Section 4.2.2.

In the very same setting, we also study the impact on the performance of `2.5-opt-EEs` of the number of realizations considered. For this purpose, we consider samples of size 10, 100, and 1000 and we denote the algorithms `2.5-opt-EEs-10`, `2.5-opt-EEs-100`, and `2.5-opt-EEs-1000`. The results are given in Figure 4.10 and Table 4.4.

Let us first focus on the performance of `2.5-opt-EEs-100`. We discuss the results of `2.5-opt-EEs-10` and `2.5-opt-EEs-1000` at the end of this section. The percentage

difference between the average cost of the solutions obtained by `2.5-opt-EEs-100` and `2.5-opt-ACs` exhibits a trend similar to the one observed on instances of size 300, but the difference between the computation times of the two algorithms is much larger. Concerning the average cost of local optima, `2.5-opt-EEs-100` achieves an average cost similar to that of `2.5-opt-ACs` with the exception of $p = 0.1$, where the average cost of local optima obtained by `2.5-opt-EEs-100` is approximately 3% higher than that of `2.5-opt-ACs`. However, `2.5-opt-EEs-100` completely dominates `1-shift` and `2-p-opt`.

Regarding the time required to reach local optima, for $p \leq 0.5$, `2.5-opt-EEs-100` is approximately 2.3, 2.5 and 3 orders of magnitude faster than `2.5-opt-ACs`, `1-shift` and `2-p-opt`, respectively. For $p > 0.5$, `2.5-opt-EEs-100` is approximately 3.5, 4.5, and 4 orders of magnitude faster than `2.5-opt-ACs`, `1-shift` and `2-p-opt`, respectively. This very large speed difference—approximately 1.2, 2, and 1 order of magnitude more than the difference in speed between the algorithms for $p \leq 0.5$—can be attributed to the computational overhead involved in the adoption of the arbitrary precision arithmetics.

Concerning the impact of the sample size on the performance of `2.5-opt-EEs`, we can observe that the use of a large number of realizations, in our case $M = 1000$, is indeed very effective with respect to the cost of the local optima for low probability values. Even though this improvement is achieved at the expense of computation time, the total search time is relatively short when compared to analytical computation algorithms. On the other hand, the use of few realizations, in our case $M = 10$, is less effective and does not significantly reduce the computation time. Concerning the average computation time, `2.5-opt-EEs-10` is faster than `2.5-opt-EEs-100` approximately by a factor of two, while `2.5-opt-EEs-1000` is slower than `2.5-opt-EEs-100` by a factor of four. Nonetheless, an important observation is that for $p \leq 0.5$, `2.5-opt-EEs-1000` is approximately 1.5 orders of magnitude faster than `2.5-opt-ACs`. For $p > 0.5$, the adoption of the arbitrary precision arithmetics entails a major computational overhead: the former is approximately 3 orders of magnitude faster than the latter. Concerning the average cost of local optima, `2.5-opt-EEs-10` is worse than the algorithms that use 100 and 1000 realizations except for $p = 0.9$; `2.5-opt-EEs-1000` is similar to `2.5-opt-EEs-100` and `2.5-opt-ACs` with the exception of $p = 0.1$, where the average cost of the local optima obtained by `2.5-opt-EEs-1000` is approximately 3% lower than that of `2.5-opt-EEs-100` and it is comparable with the one of `2.5-opt-ACs`.

Table 4.4: Experimental results for `2.5-opt-EEs-10`, `2.5-opt-EEs-100`, `2.5-opt-EEs-1000`, `2.5-opt-ACs`, `2-p-opt` and `1-shift` on clustered instances of size 1000. Each algorithm is allowed to run until it reaches a local optimum. The table gives mean and standard deviation (s.d.) of final solution cost and computation time in seconds. The results are given for 100 instances at each probability level. Note that for $p > 0.5$ the algorithms based on the analytical computation techniques use a library for arbitrary precision arithmetics. To emphasize this fact the cells are colored in gray.

	Algorithm	Solution Cost		Computation Time	
		mean	s.d.	mean	s.d.
$p = 0.1$	`2.5-opt-EEs-10`	5909938	461029	0.462	0.030
	`2.5-opt-EEs-100`	5158215	448385	1.839	0.190
	`2.5-opt-EEs-1000`	5069560	424448	9.487	1.170
	`2.5-opt-ACs`	5068223	450709	443.952	70.934
	`1-shift`	5178144	469977	635.757	84.010
	`2-p-opt`	5365486	449318	1464.535	341.993
$p = 0.2$	`2.5-opt-EEs-10`	7364518	513937	0.524	0.032
	`2.5-opt-EEs-100`	6692459	486598	1.024	0.092
	`2.5-opt-EEs-1000`	6681179	475423	3.981	0.447
	`2.5-opt-ACs`	6697814	480609	229.288	33.165
	`1-shift`	6744906	494658	547.263	71.878
	`2-p-opt`	6978843	477590	859.276	159.102
$p = 0.3$	`2.5-opt-EEs-10`	8263425	554699	0.507	0.030
	`2.5-opt-EEs-100`	7894854	547385	0.722	0.051
	`2.5-opt-EEs-1000`	7875735	511413	2.658	0.306
	`2.5-opt-ACs`	7901717	524412	149.881	22.702
	`1-shift`	7982498	531787	451.773	58.575
	`2-p-opt`	8175022	547812	552.554	95.447
$p = 0.5$	`2.5-opt-EEs-10`	9693061	630069	0.422	0.020
	`2.5-opt-EEs-100`	9592605	623310	0.526	0.038
	`2.5-opt-EEs-1000`	9591076	635788	1.689	0.149
	`2.5-opt-ACs`	9597432	599270	89.272	14.155
	`1-shift`	9856073	579796	316.049	44.883
	`2-p-opt`	9799426	594452	338.203	63.679
$p = 0.7$	`2.5-opt-EEs-10`	10852736	686839	0.371	0.017
	`2.5-opt-EEs-100`	10803761	623940	0.448	0.025
	`2.5-opt-EEs-1000`	10776397	677248	1.281	0.112
	`2.5-opt-ACs`	10805384	669183	2377.355	296.572
	`1-shift`	11203988	666372	14909.760	1911.958
	`2-p-opt`	10955608	634087	8012.715	1339.187
$p = 0.9$	`2.5-opt-EEs-10`	11752749	701937	0.347	0.012
	`2.5-opt-EEs-100`	11764968	707582	0.407	0.018
	`2.5-opt-EEs-1000`	11763689	716438	1.015	0.073
	`2.5-opt-ACs`	11777782	716614	2062.499	192.265
	`1-shift`	12241504	701184	12351.744	1610.433
	`2-p-opt`	11792577	684387	7019.027	993.119

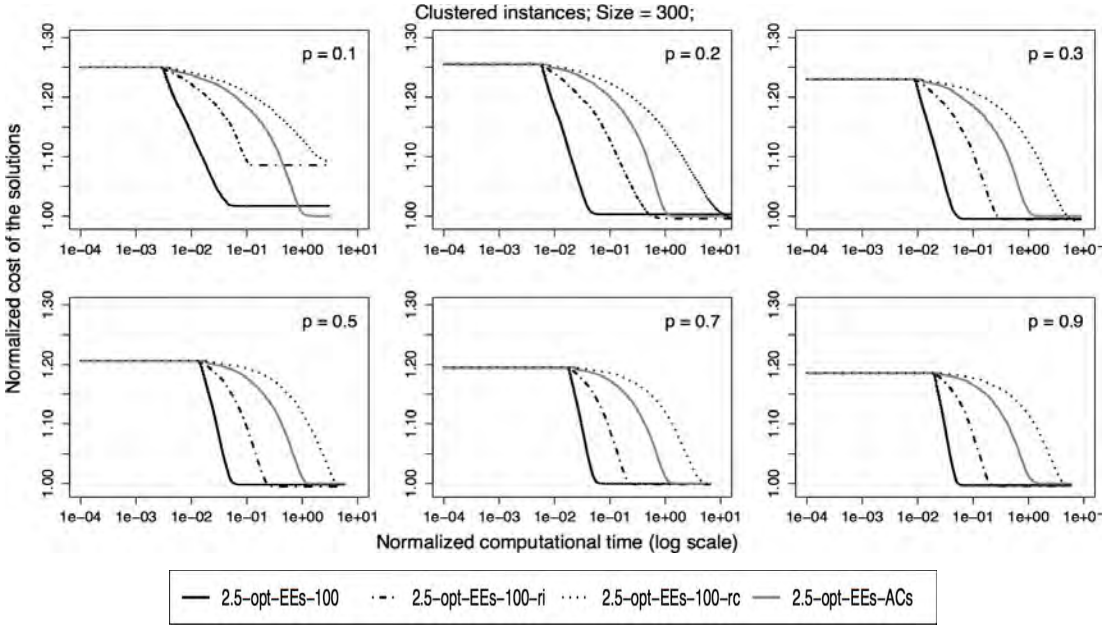


Figure 4.11: Experimental results on clustered homogeneous PTSP instances of size 300. The plots represent the average cost of the solutions obtained by `2.5-opt-EEs-100`, `2.5-opt-EEs-100-ri`, and `2.5-opt-EEs-100-rc` normalized by the one obtained by `2.5--opt-ACs`. Each algorithm is stopped when it reaches a local optimum. The normalization is done on an instance by instance basis for 100 instances; the normalized solution cost and the computation time are then aggregated. Note that the x-axis shows the computation time in logarithmic scale.

4.4.4 Experiments on sampling strategies

In this section, we present empirical results on several sampling strategies. For this study, we considered the following two alternatives in addition to the one adopted by `2.5-opt-EEs`, which consists of using the same set of M realizations for all steps of the iterative improvement algorithm: (i) a set of M realizations is sampled anew each time an improved solution is found; (ii) a set of M realizations is sampled anew for each comparison. We denote the former `2.5-opt-EEs-ri`, where `ri` stands for resampling for each improvement and the latter `2.5-opt-EEs-rc`, where `rc` stands for resampling for each comparison. Note that the sample size is set to 100. We compare `2.5-opt-EEs-100-ri` and `2.5-opt-EEs-100-rc` with `2.5-opt-EEs-100`. Moreover, `2.5-opt-ACs` is included in the analysis as a reference.

In `2.5-opt-EEs-100-ri` and `2.5-opt-EEs-100-rc`, for $p = 0.1$, the search cycles between solutions due to the high variance of the cost difference estimator with respect to the mean. To avoid this problem, we implemented a mechanism that for $p = 0.1$ memorizes moves in order to reject them in successive search steps. The results on

clustered instances with 300 nodes are given in Figure 4.11.

The results clearly show that, in our experimental setting, the strategies in which the set of realizations is changed for each improvement and for each comparison are less effective: `2.5-opt-EEs-100-ri` and `2.5-opt-EEs-100-rc` are dominated by `2.5-opt-EEs-100`. Concerning the time required to reach local optima, `2.5-opt-EEs-100` is by approximately 0.5 and 2 orders of magnitude faster than `2.5-opt-EEs-100-ri` and `2.5-opt-EEs-100-rc`, respectively. Moreover, `2.5-opt-EEs-100-rc` is slower than `2.5-opt-ACs` by a factor of approximately five. Concerning the average cost of local optima, `2.5-opt-EEs-100` is similar to `2.5-opt-EEs-100-rc` and `2.5-opt-EEs-100-ri`; an exception is $p = 0.1$, where the poor solution cost of `2.5-opt-EEs-100-ri` and `2.5-opt-EEs-100-rc` is due to the cycling problem and to the operations performed in order to avoid it.

4.5 Summary

In this chapter, we introduced `2.5-opt-EEs`, a new estimation-based iterative improvement algorithm for the PTSP. The main novelty of the proposed algorithm is the adoption of empirical estimation in delta evaluation. The algorithm adopts the 2.5-exchange neighborhood relation and it searches the neighborhood using a first-improvement rule. The effectiveness of the algorithm is further enhanced by exploiting three classical TSP neighborhood reduction techniques: fixed-radius search, candidate lists, and don't look bits.

Two sets of experiments were designed to evaluate the algorithmic components that we adopted. In the first set of experiments, we implemented a new analytical computation algorithm `2.5-opt-ACs` that uses the 2.5-exchange neighborhood relation and TSP neighborhood reduction techniques. We demonstrated that `2.5-opt-ACs` is more effective than the analytical computation algorithms `1-shift` and `2-p-opt`. Moreover, we identified a numerical precision problem in applying analytical computation algorithms to large instances. This issue needs to be addressed explicitly by resorting to arbitrary precision arithmetics and it has a major impact on computation time in the considered implementations. In the second set of experiments, we assessed the effectiveness of the estimation approach in delta evaluation. We compared the proposed algorithm `2.5-opt-EEs` using 100 realizations and `2.5-opt-ACs`, where the two algorithms differ only with respect to the evaluation procedure. The results showed that the main advantage of using the estimation approach is speed: `2.5-opt-EEs` is more than one order of magnitude faster to reach an average cost similar to that of `2.5-opt-ACs`.

We then went on to investigate several aspects of `2.5-opt-EEs`. We showed that the proposed algorithm is particularly effective for large instance sizes where it finds high quality solutions in a very short computation time. The results showed that on an instance size 1000, `2.5-opt-EEs` is more than two orders of magnitude faster than `2.5--opt-ACs`. To assess the impact of the sample size on the performance of `2.5-opt-EEs`, in addition to the default 100 realizations, we tested the algorithm with 10 and 1000 realizations. From the results, we observed that the adoption of 10 realizations does not produce a significant advantage over the adoption of 100 realizations. However, for instances with low probability values, the use of 1000 realizations allowed the algorithm to find high quality solutions. We observed that the use of 1000 realizations instead of 100 increases the time to reach the local optima by a factor of four. Finally, we illustrated that using a same set of realizations throughout the search in the proposed algorithm is more effective than the other alternatives such as changing realizations for each improvement and for each comparison.

Chapter 5

Adaptive Sample Size and Importance Sampling in Estimation-based Local Search for the PTSP

In Chapter 4, we discussed `2.5-opt-EEs`, an estimation-based local search algorithm for the PTSP. In this chapter, we improve the effectiveness of `2.5-opt-EEs` for instances with small node-probabilities by using two additional procedures. The first one is an adaptive sample size procedure that selects the appropriate number of realizations to be used in the estimation of the cost difference between two solutions; the second is a variance reduction technique called importance sampling that, when applied for the PTSP, artificially increases the probability values of the nodes such that they appear more frequently in realizations. We investigate several possible strategies for applying these procedures to the given problem and we identify the most effective one. Experimental results show that a particular heuristic customization of the two procedures increases significantly the effectiveness of `2.5-opt-EEs`. This chapter is organized as follows. In Section 5.1, we discuss the motivation behind the adoption of the two procedures and in Section 5.2 we describe their customization for `2.5-opt-EEs`; in Section 5.3, we study the effectiveness of using the two procedures in `2.5-opt-EEs`; in Section 5.4, we summarize the results.

5.1 Introduction

The adoption of the adaptive sample size procedure is motivated by the following observation: the variance of the estimator depends strongly on the probabilities associated with the nodes: the smaller the probability values, the higher the variance with respect to the mean. A high variance could be handled by increasing the sample size since averaging over a large number of realizations reduces the variance of the estimator. However, for instances with high node-probability values, using a large number of realizations is a waste of time. To address this issue, we adopt an adaptive sample size procedure that selects the appropriate number of realizations. The use of importance sampling is motivated by the fact that for instances with low node-probability values, the event that a node requires being visited in each realization is rare. The importance sampling procedure can be used to bias the nodes with low probability values to appear more frequently in realizations, which will eventually reduce the variance of the estimator with respect to the mean. The cost difference estimate obtained in this way is then corrected for the artificial bias.

There exists a number of prior publications where adaptive sample size and importance sampling have been studied in the context of stochastic combinatorial optimization. Alkhamis et al. (1999), Gutjahr (2004), Homem-de-Mello (2003), Pichitlamken and Nelson (2003), and Birattari et al. (2006a) investigated adaptive sample size procedures that make use of statistical tests to determine the number of samples to be chosen. The adoption of importance sampling to reduce the variance of the cost estimator has been investigated in Gutjahr et al. (2000a) and Gutjahr et al. (2000b). In all these works, the adaptive sample size and the importance sampling procedures have been used together with full evaluation, that is, the cost of each solution is estimated from scratch. The adoption of adaptive sample size and of importance sampling procedures in delta evaluation has never been investigated. We expect that the adoption of the two particular procedures will increase the effectiveness of `2.5-opt-EEs`. However, as we show in this chapter, the adoption is not trivial and a main contribution of the chapter consists in customizing the adaptive sample size and the importance sampling procedures for the delta evaluation applied to the PTSP. In particular, we investigate several ways of applying these procedures for the PTSP delta evaluation and we use a design of experiments approach to identify the most effective one.

5.2 Improvement procedures for 2.5-opt-EEs

In this section, we focus on the customization of adaptive sample size and importance sampling for the delta evaluation in 2.5-opt-EEs.

5.2.1 Adaptive sample size

The adaptive sample size procedure is realized using Student's t-test in the following way: given two neighboring *a priori* solutions, the cost difference between their corresponding *a posteriori* solutions is sequentially computed on a number of realizations. As soon as the t-test rejects the null hypothesis that the value of the estimated cost difference is equal to zero, the computation is stopped. If no statistical evidence is gathered, then the computation is continued until a maximum number M of realizations is considered, where M is a parameter of the procedure. In this case, that is, after M realizations are considered, the sign of the estimated difference determines the solution of lower cost. Note that the significance level of Student's t-test is also a parameter of the procedure.

The estimation-based iterative improvement algorithm, that adds the adaptive sample size procedure to 2.5-opt-EEs, will be called 2.5-opt-EEas.

5.2.2 Importance sampling

A difficulty in the adoption of the t-test is that for low probability values often the test statistic cannot be computed: since the nodes involved in the cost difference computation may not require being visited in the realizations considered, many cost differences between two *a posteriori* solutions are zero; therefore, the sample mean and the sample variance of the cost difference estimator are zero. Some degenerate cases in which this problem appears are the following. In a 2-exchange move that deletes the generic edges $\langle a, b\rangle$ and $\langle c, d\rangle$, and where no node between the nodes b and c (or between a and d) requires being visited, the difference between the two *a posteriori* solutions is zero—see Figure 4.6(a) on page 56 for an illustration. In particular, this case is very frequent when the number of nodes in the tour segment between b and c (or between the tour segment a and d) is small. In a node-insertion move, if the insertion node does not require being visited, the cost difference between the two *a posteriori* solutions is zero—see Figure 4.6(b) on page 56. A naïve strategy to handle this problem consists in postponing the t-test until non-zero sample mean and sample variance are obtained. However, this might increase the number of realizations needed for the cost

difference computation. The key idea to address this issue consists in forcing the nodes involved in the cost difference computation to appear frequently in the realizations. For this purpose, we use the variance reduction procedure known as importance sampling (Rubinstein, 1981).

In order to compute the cost difference between two *a posteriori* solutions, importance sampling, instead of using realizations of the given variable $\boldsymbol{\omega}$ parameterized by P, considers realizations of another variable $\boldsymbol{\omega}^*$ parameterized by P^*; this so-called biased distribution P^* biases the nodes involved in the cost difference computation to occur more frequently. This is achieved by choosing probabilities in P^* larger than the probabilities in P. The resulting biased cost difference between two *a posteriori* solutions for the r^{th} biased realization ω_r^* is then corrected for the adoption of the biased distribution: the correction is given by the likelihood ratio LR_r of the original distribution with respect to the biased distribution and it is obtained as

$$LR_r = \prod_{i=1}^{n} \frac{(p_i)^{\omega_r^*[i]} \cdot (1-p_i)^{1-\omega_r^*[i]}}{(p_i^*)^{\omega_r^*[i]} \cdot (1-p_i^*)^{1-\omega_r^*[i]}}, \tag{5.1}$$

where p_i and p_i^* are the original and biased probabilities of a node i, respectively. Finally, the unbiased cost difference is obtained as follows:

$$\hat{F}_M(x') - \hat{F}_M(x) = \frac{1}{M} \sum_{r=1}^{M} LR_r \cdot \Big(f(x', \omega_r^*) - f(x, \omega_r^*) \Big). \tag{5.2}$$

Recall that in delta evaluation, given a deleted edge $\langle i, j \rangle$, the algorithm needs to identify the corresponding *a posteriori* edge $\langle i^*, j^* \rangle$. In order to apply importance sampling in delta evaluation, only the nodes that are involved in finding the *a posteriori* edge, $\langle i^*, j^* \rangle$ are biased. Now, we discuss several ways of realizing this procedure.

Uniform biasing: This is a simple variant in which all the nodes involved in finding the *a posteriori* edge are biased with a probability p', where p' is a parameter. This variant does not take into account any problem-specific knowledge.

Geometric biasing: In this variant, the nodes involved in finding the *a posteriori* edge are biased with probabilities according to a geometric schedule, $\lambda^h \cdot p'$: the node i is biased with probability p', its h^{th} predecessor takes the biased probability value of $\lambda^h \cdot p'$. Similarly, the node j is biased with probability p', and its h^{th} successor takes the biased probability value of $\lambda^h \cdot p'$. Note that p' and $0 < \lambda < 1$ are parameters of this variant and, similar to the previous variant, this variant does not use any problem-

specific knowledge.

Strong greedy biasing: This variant uses k-exchange specific knowledge to bias the nodes. In a 2-exchange move, the cost difference computation involves four distinct nodes. If these four nodes require being visited in all the realizations, then there is no need for finding the predecessor and the successor of the starting nodes i and j, respectively. This variant is designed for biasing only those four nodes and their biased probability values are set to a same value p'. In the same way, for a node-insertion move only the five nodes that are involved in the cost difference computation are biased to appear in a given realization. Out of the five nodes, the biased probability of the insertion node is set to p'' and the biased probability values of the other four nodes are set to a same value p'. The reason for choosing a different value for the insertion node is that the appearance of the insertion node is more crucial than that of the other nodes, as illustrated in Figure 4.6(b) on page 56. Note that p' and p'' are parameters of this variant.

Weak greedy biasing: This variant differs from the strong greedy biasing with respect to the biasing scheme in the node-insertion move: the insertion node is biased with a value p'' and the other four nodes are not biased. This variant is designed for the following purpose: By comparing this variant with the previous one, we can study the necessity for biasing the four nodes with p'.

Heuristic biasing: This variant is similar to the weak greedy biasing except the fact that in the 2-exchange move the nodes involved in finding the *a posteriori* edge are biased. As illustrated in Figure 4.6(a) on page 56, in a 2-exchange move, the nodes in the shorter tour segment (either between b and c or between a and d) are more important than other nodes for the cost difference computation. Therefore, the nodes in the shorter segment are biased with a probability p', if the number of nodes in the shorter segment is less than $min_{is}\%$ of n, the number of nodes in the PTSP instance; min_{is} is a parameter of the procedure.

For PTSP instances with very low probability values, the computational results of the analytical computation and iterative improvement algorithms and also 2.5--opt-EEs show that the 2-exchange neighborhood is not very effective: often the improvements obtained with a 2-exchange move are rather small and therefore the time needed to reach a local optimum is high. The estimation-based local search might suffer from the aforementioned problem when all the nodes in the shorter segment are biased. In order to address this issue, the importance sampling in the 2-exchange move is used only occasionally: instead of biasing all the nodes in the shorter segment,

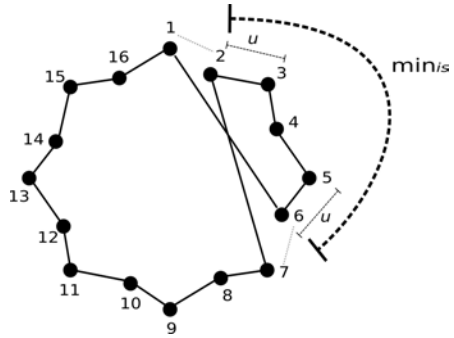


Figure 5.1: This figure illustrates heuristic biasing. In this example, a 2-exchange move is obtained by deleting the two edges $\langle 1, 2\rangle$ and $\langle 6, 7\rangle$ and by replacing them with $\langle 1, 6\rangle$ and $\langle 2, 7\rangle$. Assume that the parameter min_{is} is set to 50. Since the number of nodes in the segment $[2, \ldots, 6]$ is less than 50% of 16, that is eight, importance sampling is used to bias the nodes. However, instead of biasing all the nodes between 2 and 6, only a certain number of nodes that are close to them are biased. We assume u to be 40; therefore 40% of 5, that is, two nodes are biased on each side of the segment. The nodes that are biased are 2, 3, 5, and 6.

only a certain number of nodes, determined by another parameter u, are biased in the shorter segment. This parameter is used in the following way. Let us assume that $[b, b+1, b+2, \ldots, c+2, c+1, c]$ is the shorter segment; let us denote the number of nodes in this segment as *seg*; this variant biases u% of *seg* nodes, on each side of this segment. To give a concrete example, let us assume that *seg* is 50. If u is set to 2, then the nodes that are biased are b and c (2% of 50, that is, 1 node on each side of the segment is biased); if u is set to 4, then the nodes that are biased are $b, b+1$ and $c, c+1$, (4% of 50, that is, 2 nodes on each side of the segment are biased). In the same way, the nodes are biased if $[a, a+1, a+2, \ldots, d+2, d+1, d]$ is the shorter segment. The usage of min_{is} and u is illustrated in Figure 5.1. For the node-insertion move, only the insertion node is biased with a value p''. The parameters of this variant are p', p'', min_{is} and u.

General remarks on the importance sampling variants

The computation of the cost difference between two *a posteriori* solutions using any of the importance sampling variants proceeds as follows: Given a deleted edge $\langle i, j\rangle$, the nodes are biased with probabilities according to a selected variant. A biased realization is sampled with respect to the biased probabilities, on which the *a posteriori* edge $\langle i^*, j^*\rangle$ and the biased cost difference between two *a posteriori* solutions are obtained. The overall likelihood ratio for the biased realization is obtained by the product of the

likelihood ratio of each biased node l, which is used in finding the *a posteriori* edge $\langle i^*, j^* \rangle$. Note that the likelihood ratio of each biased node l is given by:

$$LR_r^l = \frac{(p_l)^{\omega_r^*[l]} \cdot (1 - p_l)^{1-\omega_r^*[l]}}{(p_l^*)^{\omega_r^*[l]} \cdot (1 - p_l^*)^{1-\omega_r^*[l]}}, \tag{5.3}$$

where p_l and p_l^* are the original and the biased probability of node l and $\omega_r^*[l]$ is sampled with the biased probability p_l^*. The unbiased cost difference between two *a posteriori* solutions is simply given by the product of the overall likelihood ratio and the biased cost difference between two *a posteriori* solutions.

The values of the biased probabilities of the aforementioned importance sampling variants are crucial for the variance reduction. In particular, if those values are inappropriate, then the adoption of importance sampling variants will increase the variance of the cost estimator. We address this issue using a parameter tuning algorithm in Section 5.3.

We denote `2.5-opt-EEais` the algorithm that adds to `2.5-opt-EEas` any of the described importance sampling variants.

Implementation-specific details

In order to implement `2.5-opt-EEais` efficiently, we use the same data structure as that of `2.5-opt-EEs`, which is composed of a doubly circularly linked list and some auxiliary arrays as described in Chapter 4. In `2.5-opt-EEais` with strong greedy, weak greedy and heuristic biasing, for each node three realization arrays, ω, ω', and ω'' are stored, each of size M, indexed from 1 to M. Element r of a realization array is either `1` or `0` indicating whether node i requires being visited or not in a realization and it is obtained as follows: first a random number between 0 and 1 is generated; if this number is less than or equal to p_i, p_i', or p_i'', node i requires being visited in realization ω_r, ω_r', or ω_r'', respectively. In `2.5-opt-EEais` with uniform biasing, each node has two realization arrays, ω, ω', each of size M. In `2.5-opt-EEais` with geometric biasing, given λ and p', it is possible to compute the set of all possible biased probability values that a node can take. Therefore, each node has a set of biased realizations, where each of them is sampled with respect to a possible biased probability. In all the variants, when the biased probability value of a node is less than the original probability, the former is set to the latter. Given p_i and the set of all biased probability values of a node i, the likelihood ratio is pre-computed and stored when the algorithm starts.

`2.5-opt-EEais` uses a same set of realizations for all iterative improvement steps.

However, for each 2-exchange and node-insertion move, the realizations are selected randomly from this set until the t-test rejects the null hypothesis.

The following techniques are used to speed up the computations involved in the t-test. The critical values of Student's t-distribution are pre-computed and stored in a lookup table and the sample mean and the sample variance of the cost difference estimator are computed recursively.

The implementation of `2.5-opt-EEas` is identical to `2.5-opt-EEais` except for the fact that the importance sampling procedure and the pre-computations required for `2.5-opt-EEais` are excluded.

5.3 Experimental analysis

In this section, we present the experimental setting considered and the empirical results. Our goal is to show that the integration of the adaptive sample size and the importance sampling procedures into the estimation-based local search increases significantly its effectiveness.

We generate PTSP instances with the DIMACS instance generator as described in Chapter 4. We use two classes of instances: homogeneous and heterogeneous PTSP instances. For the homogeneous instances, we consider probability values starting from 0.050 to 0.200 with an increment of 0.025 and from 0.3 to 0.9 with an increment of 0.1. The probability values in the heterogeneous instances are generated using a beta distribution as described in Bianchi (2006). In this scheme, the probability values of each instance are characterized by two parameters: the mean probability p_m and the percentage of maximum variance p_v: when an instance is generated with p_m and p_v, the expected value and the variance of the random variable $\boldsymbol{\omega}$ parameterized by P are p_m and $(p_v/100) \cdot p_m(1 - p_m)$, respectively. For the sake of convenience, we refer to the probability level of an instance as $p = p_m(p_v\%)$. We considered the values for p_m from 0.050 to 0.200 with increments of 0.025 and from 0.3 to 0.5 with increments of 0.1; for each value of p_m, we considered three values for p_v: $\{16, 50, 83\}$. We generate 50 instances for each combination of p_m and p_v. We present results obtained for probability values up to 0.200. The trends of the results obtained for the higher probability values are very similar to those presented here; we refer the reader to the following supplementary page (Balaprakash et al., 2008b) for the complete set of results.

`http://iridia.ulb.ac.be/supp/IridiaSupp2008-010/`

In `2.5-opt-EEas` and `2.5-opt-EEais`, the minimum number of realizations used

in the adaptive sample size procedure before applying the t-test is set to five. The null hypothesis is rejected at a significance level of 0.05. If the test statistic cannot be computed after five realizations, the cost difference computation is stopped and the algorithm considers the next neighbor solution. The maximum number M of realizations is set to one thousand. We use Equation 4.1 for the post-evaluation of the best-so-far solutions found by each algorithm according to its evaluation procedure.

For the homogeneous PTSP with $p \geq 0.1$, `2.5-opt-ACs` has already been shown to be more effective than `1-shift` and `2-p-opt` in Chapter 4. We carried out some preliminary experiments to verify that the same tendency holds also for low probability values, that is, for $p < 0.1$. In these experiments, `2.5-opt-ACs` outperforms both `1-shift` and `2-p-opt` and, therefore, we take `2.5-opt-ACs` as a yardstick for measuring the effectiveness of the proposed algorithms. The computational results for $p < 0.1$ are given in Balaprakash et al. (2007).

5.3.1 Parameter tuning

Finding appropriate values for the parameters—in particular the biased probability values—of the importance sampling variants adopted in `2.5-opt-EEais` is crucial for the variance reduction. For this purpose, we use a parameter tuning algorithm, Iterative F-Race, which we developed during our research, to identify suitable values for the parameters of each importance sampling variant. For a description of this algorithm, we refer the reader to Annex B. We tune each importance sampling variant separately for the homogeneous and the heterogeneous clustered instances of size 1000. We use 210 instances (7 levels of probability $\times$ 30 instances) for the homogeneous case and 210 instances (7 levels of probability $\times$ 3 levels of percentage of maximum variance $\times$ 10 instances) for the heterogeneous case. Table 5.1 shows, for each importance sampling variant, the range of each parameter given to the tuning algorithm and the selected value.

From the parameter values of strong greedy biasing, weak greedy biasing, and heuristic biasing, we can observe the following trend: low biased probability values are appropriate for the nodes involved in the 2-exchange moves, whereas high biased probability values are selected for the nodes involved in the node-insertion moves. In particular, under our experimental setting, the appropriate ranges for the biased probability values, p', in 2-exchange moves for homogeneous and heterogeneous PTSP instances are $[0.11, 0.2]$ and $[0.07, 0.12]$, respectively. This low range of values is due to the ineffectiveness of the 2-exchange neighborhood for low probability values as discussed in

Table 5.1: Parameter values considered for tuning the importance sampling variants in `2.5-opt-EEais` and the values selected by Iterative/F-Race.

variant	parameter	range	selected value	
			homogeneous	heterogeneous
uniform biasing	p'	$[0.0, 1.0]$	0.23	0.08
geometric biasing	p'	$[0.0, 1.0]$	0.39	0.18
	h	$[0.0, 1.0]$	0.25	0.12
strong greedy biasing	p'	$[0.0, 1.0]$	0.20	0.12
	p''	$[0.0, 1.0]$	0.76	0.60
weak greedy biasing	p'	$[0.0, 1.0]$	0.24	0.08
	p''	$[0.0, 1.0]$	0.79	0.64
heuristic biasing	p'	$[0.0, 1.0]$	0.11	0.07
	p''	$[0.0, 1.0]$	0.60	0.57
	min_{is}	$[0.0, 20.0]$	0.55	1.30
	u	$[0, 100]$	72.00	10.00

Section 5.2.2. The biased probability values for the nodes involved in the node-insertion moves are relatively high. In the current experimental setting, the appropriate range for the biased probability values, p'', are $[0.57, 0.79]$ for both homogeneous and heterogeneous PTSP instances. For what concerns uniform and geometric biasing, which do not use a separate biased probability value for 2-exchange and node-insertion moves, the tuning algorithm tries to find a good biased probability that is suitable for both types of moves. Eventually, this results in values that are higher than those for the 2-exchange moves and lower than those for the node-insertion moves in the greedy and heuristic biasing.

5.3.2 A study on the parameters of `2.5-opt-EEais`

In this section, we study the impact of the parameters of `2.5-opt-EEais` on solution quality and computation time. The main aim of this analysis is to identify an appropriate significance level for the adaptive sample size procedure and to determine the most promising importance sampling variant. Moreover, we also study the robustness of `2.5-opt-EEais` with respect to instance size and the way in which the nodes are distributed in the instances. For this purpose, we use the analysis of variance (ANOVA) technique. In ANOVA terminology, the parameters of `2.5-opt-EEais` are called factors and the solution quality and computation time are called response variables. In this analysis, we use `2.5-opt-EEs-1000` (`2.5-opt-EEs` that uses 1000 realizations without adaptive sample size and importance sampling) as a reference algorithm: we study the solution quality of a given algorithm as the percentage deviation from the cost of

`2.5-opt-EEs-1000`. The computation time of a given algorithm is normalized with respect to the computation time of `2.5-opt-EEs-1000`. We perform an ANOVA analysis for each of the response variables. In order to apply ANOVA, it is necessary to check three main assumptions on the distribution of the response variables, namely, normality, homogeneity of variance, and independence of residuals. Since grouping over all probability levels results in violation of the assumptions, the ANOVA analysis is done for each level. Even in this setting, there are a few probability levels for which the ANOVA assumptions are violated. In such cases, we use the non parametric Wilcoxon rank sum test to verify the results of the ANOVA analysis. For the complete ANOVA results, we refer the reader to Balaprakash et al. (2008b); here, we highlight some main results of the analysis.

5.3.2.1 Importance sampling variants

In this analysis, we study the effect of importance sampling variants in `2.5-opt-EE-ais` on solution quality and computation time. The results are shown as box plots in Figure 5.2. Note that the significance level in the adaptive sample size procedure is set to 0.05. For what concerns the homogeneous instances, the F-ratio and the p-values from the ANOVA table show that importance sampling has a significant impact on the solution quality for probability levels less than 0.150. The p-values from the pairwise t-test indicate that strong greedy, weak greedy and heuristic biasing are significantly better than the uniform and the geometric biasing. However, there is no significant difference among strong greedy, weak greedy and heuristic biasing. For what concerns the heterogeneous instances, the F-ratio and the p-values from the ANOVA table show that there is no significant difference among the different importance sampling variants. Nevertheless, the heuristic biasing obtains local optima whose average is slightly better than that of the other variants.

Concerning the computation time, the F-ratio and the p-values from the ANOVA table show the following general trend for the probability values less than 0.150: the heuristic biasing and the weak greedy biasing are significantly faster than other variants for both homogeneous and heterogeneous instances. Also note that, although there is no significant difference between heuristic biasing and weak greedy biasing, the computation time of the former is slightly lower than that of the latter.

Taking into account both solution quality and computation time, we select the heuristic biasing as the most promising variant and we use it as a basis for the following analyses.

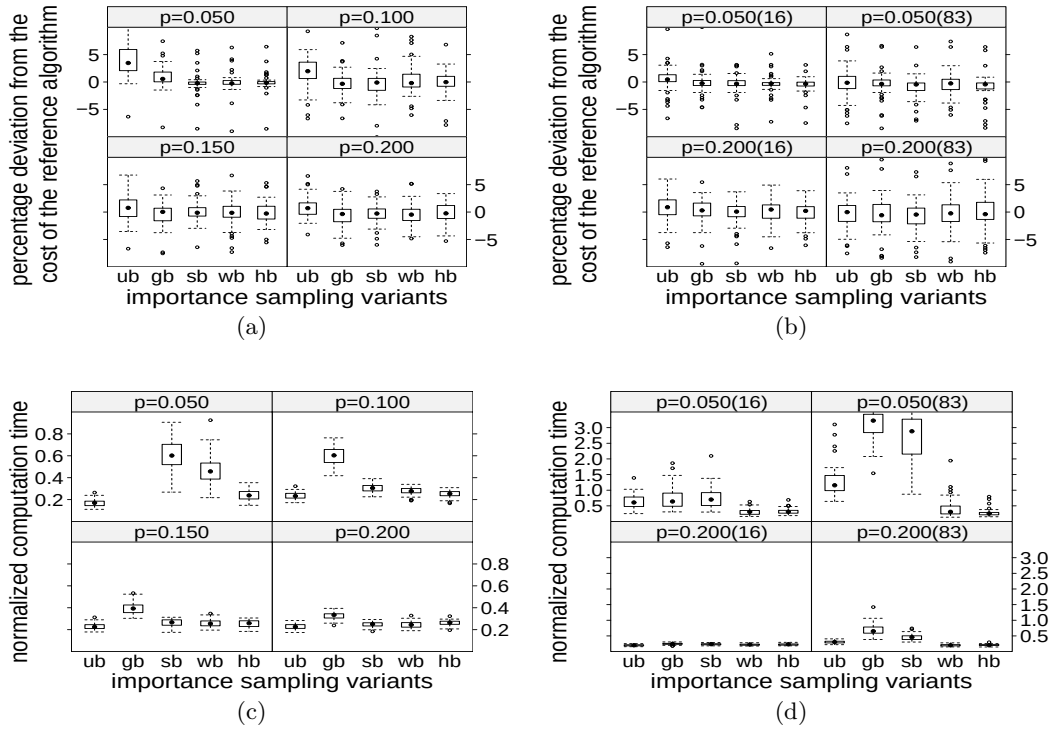


Figure 5.2: Analysis of different importance sampling variants on clustered PTSP instances of size 1000. Plots 5.2(a) and 5.2(b) show the cost of the solutions of `2.5-opt--EEais` with uniform biasing (ub), geometric biasing (gb), strong greedy biasing (sb), weak greedy (wb), and heuristic biasing (hb) as the percentage deviation from the cost of `2.5-opt-EEs-1000` on homogeneous and heterogeneous PTSP instances. Plots 5.2(c) and 5.2(d) show the normalized computation time of `2.5-opt-EEais` with different importance sampling variants, where the normalization is done with respect to the computation time of `2.5-opt-EEs-1000` for homogeneous and heterogeneous instances, respectively.

5.3.2.2 Significance level

In this analysis, we study the effect of the significance level used in the adaptive sample size procedure on the solution quality and on the computation time. We use `2.5-opt--EEais` with heuristic biasing for the experimental analysis. We consider 4 significance levels: $[0.01, 0.02, 0.05, 0.10]$. The F-ratio and the p-values from the ANOVA table show that the significance level does not have a significant impact on the solution quality; however, it has a significant impact on the computation time. The results on the computation time are shown as box plots in Figure 5.3: at significance levels 0.01 and 0.02, the algorithm needs more realizations, thus more computation time, to reject the null hypothesis at each step than for significance level 0.05. Nevertheless, the computation time of `2.5-opt-EEais` at significance level 0.10 is higher than for other

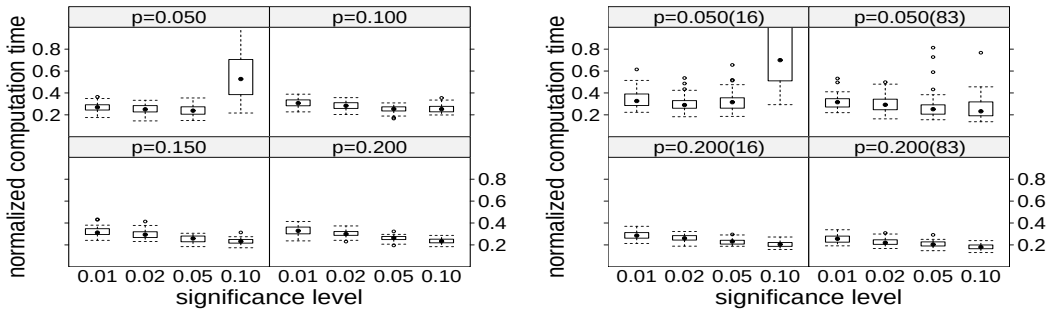


Figure 5.3: Analysis of different significance levels on clustered PTSP instances of size 1000. Plots 5.3(a) and 5.3(b) show the normalized computation time of `2.5-opt-EEais` with different significance levels, where the normalization is done with respect to the computation time of `2.5-opt-EEs-1000` for homogeneous and heterogeneous instances, respectively.

significance levels for low probability values. This can be attributed to the fact that the estimates of the cost differences are less precise for low probability values and as a consequence the algorithm with significance level 0.10 incorrectly moves to a number of non-improving neighbor solutions before reaching a local optimum. However, for high probability levels, where the variance of the cost estimate is low, the computation time of `2.5-opt-EEais` at significance level 0.10 is lower than for other significance levels. Taking into account both low and high probability values, we can see that the significance level 0.05 is appropriate for `2.5-opt-EEais`.

5.3.2.3 Instance size and distribution of nodes

In this analysis, we study the robustness of `2.5-opt-EEais` that adopts heuristic biasing with respect to instance size and nodes distribution. Note that the significance level in the adaptive sample size procedure is set to 0.05. We consider three levels $[100, 300, 1000]$ for instance size and two levels for nodes distribution, namely, uniform and clustered. The F-ratio and the p-values from the ANOVA table show that these factors do not have any significant impact on the relative solution quality and computation time. Note that the instance size will always have a significant impact on the absolute solution cost and computation time; however, recall that in this analysis, we always study the relative solution quality with respect to `2.5-opt-EEs-1000`.

Even though the parameter tuning for `2.5-opt-EEais` with heuristic biasing is performed only on the clustered instances of size 1000, it achieves good solutions also for other levels of instance size. This is mainly attributed to the ineffectiveness of 2-

exchange moves for small instances: min_{is} and u are the two parameters of `2.5-opt--EEais` that depend on the instance size; under the given parameter setting, for instance sizes 100 and 300, importance sampling is completely disabled for 2-exchange moves; from the results it seems that the usage of importance sampling in 2-exchange moves is not crucial for small instances.

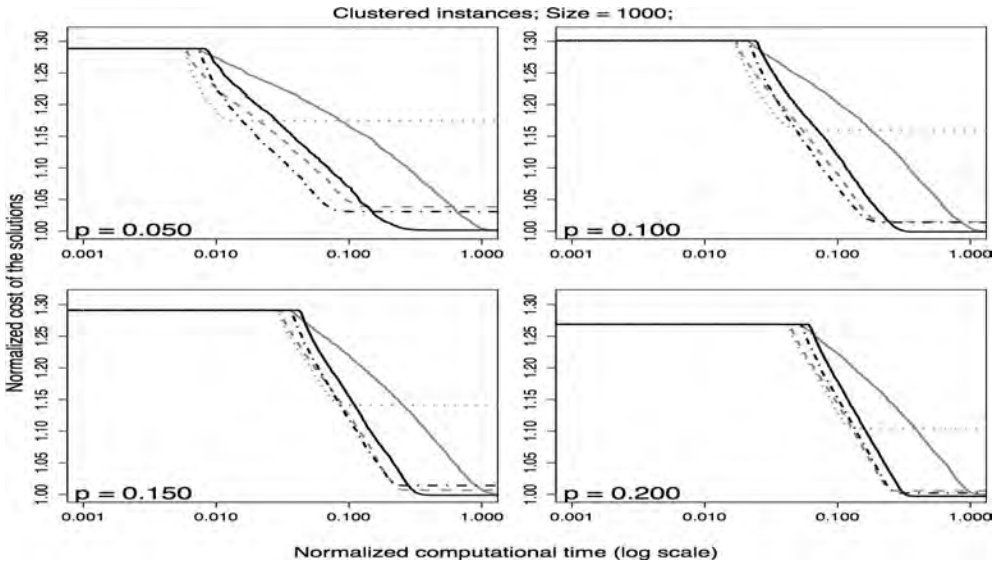
5.3.3 Experiments to assess variance reduction

In this section, we study the magnitude of reduction in variance obtained using heuristic biasing. For this purpose, we analyze the variance under two settings: `2.5-opt-EEs` that adopts a fixed sample size without importance sampling and `2.5-opt-EEs` that adopts a fixed sample size and heuristic biasing. We consider three sample sizes: 10, 100 and 1000. The two algorithms are then allowed to explore 1000 solutions. The variance of the cost difference estimator for each estimation in 2-exchange and node-insertion moves is recorded. The results for node-insertion moves are shown in Table 5.2. Since the computational results for 2-exchange moves show that the reduction in variance is rather small and less than 1%, we do not list the values in a table.

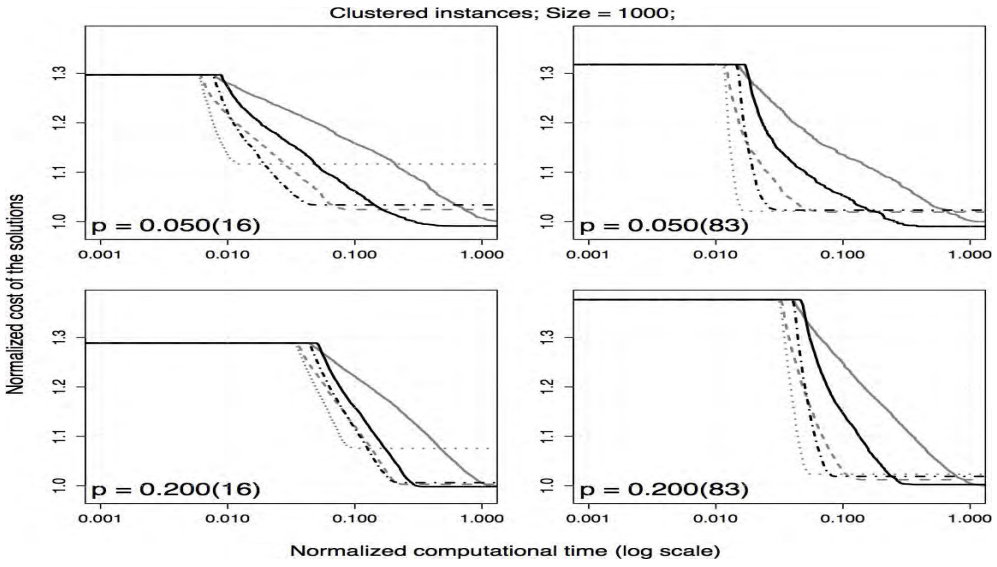
The magnitude of reduction in variance is very high for the node-insertion moves, in particular for low probability values: on average, the variance of the cost estimator when using heuristic biasing is between 14% and 97% less than when not using importance sampling. Another important observation is that a reduction in variance is achieved by increasing the size of the sample; however, the percentage reduction does not follow a same trend. For example, consider the case at $p_m = 0.200$ and $p_v = 0$: the average variance is reduced from $2.31e+07$ to $3.35e+06$ by increasing the realizations from 10 to 1000; however, the percentage reduction does not show a strictly decreasing trend as it goes from 76.72 to 68.71, and finally 70.50.

5.3.4 Experiments on estimation-based algorithms

In this section, we study the performance of `2.5-opt-EEas` and `2.5-opt-EEais` by comparing their solution cost and computation time to `2.5-opt-EEs`. In the case of `2.5-opt-EEs`, we consider samples of size 10, 100, and 1000; we denote these algorithms by `2.5-opt-EEs-10`, `2.5-opt-EEs-100`, and `2.5-opt-EEs-1000`, respectively. Note that these algorithms do not use the adaptive sample size and the importance sampling procedures. The results of the comparison of the five algorithms are given in Figure 5.4, where `2.5-opt-EEs-1000` is taken as a reference. Table 5.3 shows the absolute values.



(a) Homogeneous PTSP



(b) Heterogeneous PTSP

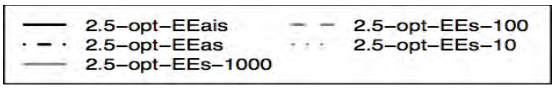


Figure 5.4: Experimental results on clustered PTSP instances of size 1000. The plots represent the cost of the solutions obtained by `2.5-opt-EEas`, `2.5-opt--EEais`, `2.5-opt-EEs-10`, and `2.5-opt-EEs-100` normalized by those obtained by `2.5-opt-EEs-1000`. Each algorithm is stopped when it reaches a local optimum. The normalization is done on an instance by instance basis for 50 instances; the normalized solution cost and the computation time are then aggregated.

Table 5.2: Experimental results on variance reduction by heuristic biasing on clustered instances of size 1000. The algorithm is allowed to explore 1000 solutions. The table gives, for each probability level, the average of the variances computed for 1000 delta estimations with and without importance sampling. The last column shows the percentage reduction of the cost estimator variance when using heuristic biasing (hb).

p_m	p_v	sample size	avg. of 1000 cost estimator variances		reduction in %
			without hb	with hb	
node-insertion move					
0.050	00	10	5.24e+08	1.09e+07	97.92
		100	9.51e+06	4.48e+05	95.29
		1000	6.94e+05	3.82e+04	94.50
	16	10	5.98e+08	6.71e+07	88.78
		100	2.38e+07	3.54e+06	85.13
		1000	3.86e+05	7.23e+04	81.25
	50	10	2.29e+08	7.27e+07	68.22
		100	2.89e+07	7.80e+06	72.99
		1000	2.07e+06	4.86e+05	76.55
	83	10	7.20e+07	4.13e+07	42.60
		100	5.90e+06	3.90e+06	33.86
		1000	1.57e+05	9.94e+04	36.81
0.200	00	10	9.90e+07	2.31e+07	76.72
		100	1.24e+07	3.87e+06	68.71
		1000	1.14e+06	3.35e+05	70.50
	16	10	1.29e+08	6.43e+07	50.29
		100	1.08e+07	6.36e+06	41.25
		1000	1.05e+06	5.90e+05	44.08
	50	10	1.02e+08	8.03e+07	21.04
		100	2.78e+06	2.09e+06	24.87
		1000	2.22e+05	1.46e+05	34.07
	83	10	7.51e+07	6.47e+07	13.85
		100	5.43e+06	4.48e+06	17.45
		1000	3.83e+05	3.29e+05	14.15

The computational results show that `2.5-opt-EEais` is more effective than the other algorithms, in particular, for low probability levels. For what concerns the comparison of `2.5-opt-EEais` and `2.5-opt-EEas`, the results show that the adoption of importance sampling allows the former to achieve high quality solutions for very low probability levels, that is, for p and $p_m < 0.2$: the average cost of the local optima obtained by `2.5-opt-EEais` is between 1% and 3% less than that of `2.5-opt-EEas`. The observed differences are significant in a statistical sense. The poor solution cost of `2.5-opt-EEas` can be mainly attributed to the following reason: Since this algorithm considers the next neighbor solution when the test statistic cannot be computed after five realizations (see Section 5.2.2), it rejects moves which are likely to be accepted.

Table 5.3: Experimental results for `2.5-opt-EEas`, `2.5-opt-EEais`, `2.5-opt-EEs-10`, `2.5-opt-EEs-100`, and `2.5-opt-EEs-1000` on clustered instances of size 1000. Each algorithm is allowed to run until it reaches a local optimum. The table gives, for each probability level, the mean and the standard deviation (s.d.) of the final solution cost and of the computation time in seconds over 50 instances.

	Algorithm	Solution Cost		Computation Time	
		mean	s.d.	mean	s.d.
Homogeneous PTSP					
$p = 0.050$	`2.5-opt-EEais`	4020433	437996	11.100	1.794
	`2.5-opt-EEas`	4137855	430945	2.970	0.497
	`2.5-opt-EEs-1000`	4014200	455410	41.104	6.821
	`2.5-opt-EEs-100`	4168788	434760	4.309	0.713
	`2.5-opt-EEs-10`	4713400	491452	0.482	0.035
$p = 0.100$	`2.5-opt-EEais`	5103869	508867	4.119	0.451
	`2.5-opt-EEas`	5179648	486450	2.230	0.249
	`2.5-opt-EEs-1000`	5108555	503921	14.096	1.972
	`2.5-opt-EEs-100`	5183844	470288	2.629	0.306
	`2.5-opt-EEs-10`	5922935	509627	0.591	0.053
$p = 0.150$	`2.5-opt-EEais`	5959120	496566	2.495	0.301
	`2.5-opt-EEas`	6050183	505229	1.702	0.161
	`2.5-opt-EEs-1000`	5966002	479174	8.104	1.172
	`2.5-opt-EEs-100`	6007125	500754	1.827	0.187
	`2.5-opt-EEs-10`	6808184	555047	0.648	0.045
$p = 0.200$	`2.5-opt-EEais`	6701562	545366	1.776	0.147
	`2.5-opt-EEas`	6734587	558760	1.407	0.120
	`2.5-opt-EEs-1000`	6720197	543464	5.596	0.574
	`2.5-opt-EEs-100`	6758117	563812	1.349	0.114
	`2.5-opt-EEs-10`	7416077	612763	0.661	0.053
Heterogeneous PTSP					
$p = 0.050(16)$	`2.5-opt-EEais`	3949356	409824	13.494	3.379
	`2.5-opt-EEas`	4119370	461810	1.473	0.207
	`2.5-opt-EEs-1000`	3984725	441295	38.069	7.455
	`2.5-opt-EEs-100`	4082128	435499	2.806	0.405
	`2.5-opt-EEs-10`	4449540	447232	0.418	0.028
$p = 0.050(83)$	`2.5-opt-EEais`	3876139	483153	6.251	2.457
	`2.5-opt-EEas`	4005424	550905	0.511	0.054
	`2.5-opt-EEs-1000`	3914341	522037	19.476	3.409
	`2.5-opt-EEs-100`	3990014	519289	1.033	0.163
	`2.5-opt-EEs-10`	3996970	531437	0.305	0.011
$p = 0.200(16)$	`2.5-opt-EEais`	6589342	537399	1.910	0.203
	`2.5-opt-EEas`	6641879	547318	1.283	0.118
	`2.5-opt-EEs-1000`	6601665	537940	6.674	0.890
	`2.5-opt-EEs-100`	6620425	553190	1.447	0.114
	`2.5-opt-EEs-10`	7100032	569439	0.579	0.049
$p = 0.200(83)$	`2.5-opt-EEais`	6244761	605180	1.991	0.284
	`2.5-opt-EEas`	6348075	607690	0.546	0.039
	`2.5-opt-EEs-1000`	6230550	602679	7.217	1.128
	`2.5-opt-EEs-100`	6306303	604002	0.811	0.090
	`2.5-opt-EEs-10`	6375420	613433	0.367	0.020

However, the adoption of importance sampling in 2.5-opt-EEais reduces the effect of this problem. For high probability levels, the average cost of the solutions and the computation time of 2.5-opt-EEais are comparable to those of 2.5-opt-EEas.

Concerning the comparison of 2.5-opt-EEais and 2.5-opt-EEs-1000, on average they have very similar costs. However, the advantage of 2.5-opt-EEais is the computation time: 2.5-opt-EEais is faster than 2.5-opt-EEs-1000 approximately by a factor of four.

Regarding the comparison of 2.5-opt-EEais and 2.5-opt-EEs-100, for low probability levels the average cost of the solutions obtained by the former is between 1% and 3% lower than that of 2.5-opt-EEs-100. This clearly shows that the adoption of 100 realizations is not sufficient for these probability levels. Note that these differences are significant according to the paired t-test. On the other hand, for high probability levels, the two algorithms are comparable to one another with respect to solution quality and computation time.

Although faster, 2.5-opt-EEs-10 achieves a very poor solution quality: the average cost of the solutions obtained by 2.5-opt-EEs-10 is between 17% and 2% higher than that of 2.5-opt-EEais.

The experimental results for $p > 0.5$ are reported in Balaprakash et al. (2007). These results show that the algorithms achieve equivalent results with respect to solution quality. Moreover, the results of 2.5-opt-EEs-10 show that a sample size of 10 is sufficient to tackle instances with $p > 0.5$. For what concerns computation time, 2.5-opt-EEais and 2.5-opt-EEs-100 are comparable to 2.5-opt-EEs-10. However, 2.5-opt-EEais is faster than 2.5-opt-EEs-1000 by a factor of three. Note that 2.5-opt-EEais and 2.5-opt-EEas are essentially the same for these probability levels.

Taking into account both the computation time and the cost of the solutions obtained, we can see that 2.5-opt-EEais emerges as a clear winner among the considered estimation-based algorithms.

5.3.5 Comparison with the analytical computation algorithm

In this section, we compare 2.5-opt-EEais with heuristic biasing to 2.5-opt-ACs. For this purpose, we generate 50 new instances for each probability level. The rationale behind the adoption of a new set of instances is the following: 2.5-opt-EEais and 2.5--opt-ACs are selected as winners from a set of 5 and 3 algorithms, respectively, where all of them are evaluated on a same set of instances. Basing the comparison of 2.5-opt--EEais and 2.5-opt-ACs on the same set of instances might possibly introduce a bias

Table 5.4: Comparison of the average cost obtained by `2.5-opt-EEais` and by `2.5-opt--ACs` on clustered instances of size 1000. See Table 4.3 on page 64 for an explanation of the contents and the typographic conventions adopted in the table.

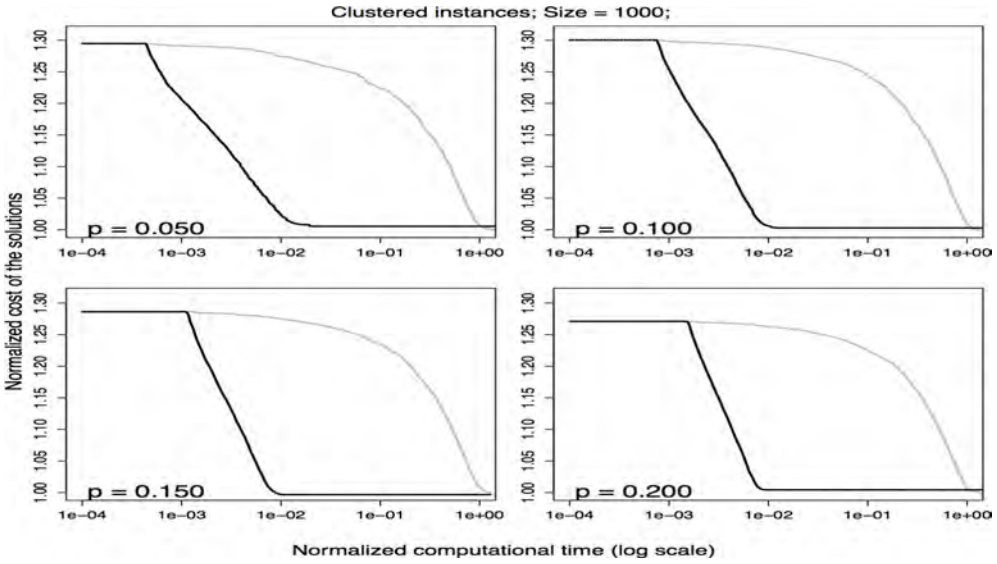
Homogeneous PTSP		
	`2.5-opt-EEais` vs. `2.5-opt-ACs`	
p	Difference	95% CI
0.050	+0.546%	[−0.157, +1.248]%
0.075	+0.232%	[−0.675, +1.139]%
0.100	+0.284%	[−0.645, +1.214]%
0.125	−0.333%	[−1.122, +0.456]%
0.150	−0.327%	[−1.132, +0.478]%
0.200	+0.422%	[−0.386, +1.231]%

Heterogeneous PTSP		
	`2.5-opt-EEais` vs. `2.5-opt-ACs`	
p	Difference	95% CI
0.050(16)	+0.472%	[−0.243, +1.187]%
0.050(50)	**+0.812**%	[+0.147, +1.478]%
0.050(83)	+0.390%	[−0.648, +1.428]%
0.075(16)	+0.998%	[−0.081, +2.077]%
0.075(50)	+0.191%	[−0.372, +0.754]%
0.075(83)	+0.780%	[−0.128, +1.688]%
0.100(16)	−0.037%	[−0.868, +0.795]%
0.100(50)	−0.199%	[−1.091, +0.693]%
0.100(83)	+0.352%	[−0.494, +1.199]%
0.200(16)	−0.514%	[−1.394, +0.366]%
0.200(50)	**−1.052**%	[−1.781, −0.323]%
0.200(83)	−0.086%	[−0.800, +0.629]%

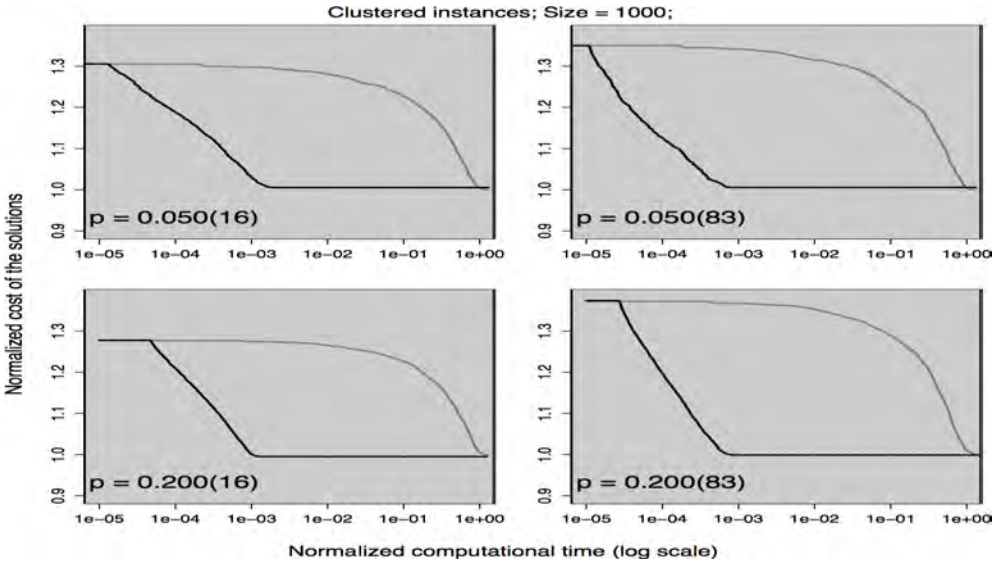
in favor of `2.5-opt-EEais`. This issue is known as over-tuning; we refer the reader to Birattari (2004) for further discussion.

The computational results given in Figure 5.5 show that `2.5-opt-EEais` is very competitive. Regarding the time required to reach local optima, irrespective of the probability levels, `2.5-opt-EEais` is approximately 2 orders and 3 orders of magnitude faster than `2.5-opt-ACs`, for homogeneous and heterogeneous instances, respectively. This very large speed difference in the heterogeneous case—approximately 1 order of magnitude more than the difference in speed between the algorithms for the homogeneous case—can be attributed to the computational overhead involved in the adoption of the arbitrary precision arithmetics. We refer the reader to Balaprakash et al. (2007) for the absolute values.

The average cost of local optima obtained by `2.5-opt-EEais` is comparable to the one of `2.5-opt-ACs`. In Table 5.4, we report the observed relative difference between the cost of the local optima obtained by the two algorithms and a 95% confidence bound on this relative difference. This bound is obtained through a two sided paired t-test. Table 5.4 confirms that, concerning the average cost of the local optima found, `2.5-opt-EEais` is essentially equivalent to `2.5-opt-ACs`. Nevertheless, with 95% confidence, under the current experimental setting, we can state that, should ever the



(a) Homogeneous PTSP



(b) Heterogeneous PTSP

Figure 5.5: Experimental results on clustered PTSP instances of size 1000. The plots represent the cost of the solutions obtained by `2.5-opt-EEais` normalized by those obtained by `2.5-opt-ACs`. Each algorithm is stopped when it reaches a local optimum. For the heterogeneous case, `2.5-opt-ACs` uses a library for arbitrary precision arithmetics. To emphasize this fact, the background of plots is gray. The normalization is done on an instance by instance basis for 50 instances; the normalized solution cost and the computation time are then aggregated.

average cost obtained by `2.5-opt-EEais` be higher than the one obtained by `2.5-opt--ACs`, the difference would be at most 1.2% and 2.1% for the homogeneous and the heterogeneous instances, respectively.

5.3.6 Experiments with iterated local search

In this section, we study the behavior of `2.5-opt-EEais` and `2.5-opt-EEs-100` integrated into iterated local search (ILS) (Lourenço et al., 2002), a metaheuristic on which many high performing algorithms for the TSP are based (Hoos and Stützle, 2005). We denote the two algorithms `ILS-2.5-opt-EEais` and `ILS-2.5-opt-EEs-100`. It is interesting to note that, for a given computation time, the two algorithms behave differently. `ILS-2.5-opt-EEais` obtains, for low probability values, at each iteration a high quality local optimum due to the adoption of `2.5-opt-EEais`; however, this high quality local optimum is obtained at the expense of higher computation time per local search. Given the same computation time, `ILS-2.5-opt-EEs-100` obtains at each iteration lower quality local optima than that of `ILS-2.5-opt-EEais`. Nevertheless, the former performs more iterations than that of the latter due to the adoption of the faster but less effective `2.5-opt-EEs-100`. The goal of this experiments is to determine which of the two algorithms is better for the PTSP.

We implemented standard ILS algorithms, in which new starting solutions for the subsequent local search are generated by perturbing the incumbent local optimum s^*. For the perturbation, we adopt a hybrid scheme that consists in first performing two random double-bridge moves and then changing the position of $ps\%$ of the nodes, where ps is a parameter. A change of the position is done by picking uniformly at random $ps\%$ of the nodes, removing them from the tour and then re-inserting them according to the farthest insertion heuristic. In our experiments, the parameter ps is set to 10. From the solution obtained after the perturbation, a new local search is started. If the newly identified local optimum has a lower cost than s^*, it is accepted as the new incumbent solution.

We include two more algorithms in the analysis: an ILS algorithm built on top of `2.5-opt-EEs`, which uses a sample size schedule proposed by Gutjahr (2004). In this schedule, the number of realizations is increased on the basis of the iteration counter. We denote this algorithm as `ILS-2.5-opt-EEs-sss`, where `sss` stands for sample size schedule. The second algorithm is `ILS-2.5-opt-EEs-1000`, which adopts `2.5-opt-EEs-1000`; this algorithm is used as a reference algorithm.

In `ILS-2.5-opt-EEais`, the acceptance criterion of ILS compares two local op-

tima by using the t-test with a maximum of 1000 realizations, which are unchanged throughout all the iterations. In ILS-2.5-opt-EEs-100 and ILS-2.5-opt-EEs-1000, the acceptance criterion compares two local optima based on 100 and 1000 realizations, respectively. In ILS-2.5-opt-EEs-sss, the sample size schedule determines the number of realizations for comparing the two local optima.

The stopping criterion for the considered algorithms is set to 100 seconds. We use 50 instances for each probability level. The results on clustered instances with 1000 nodes are given in Figure 5.6 and Table 5.5.

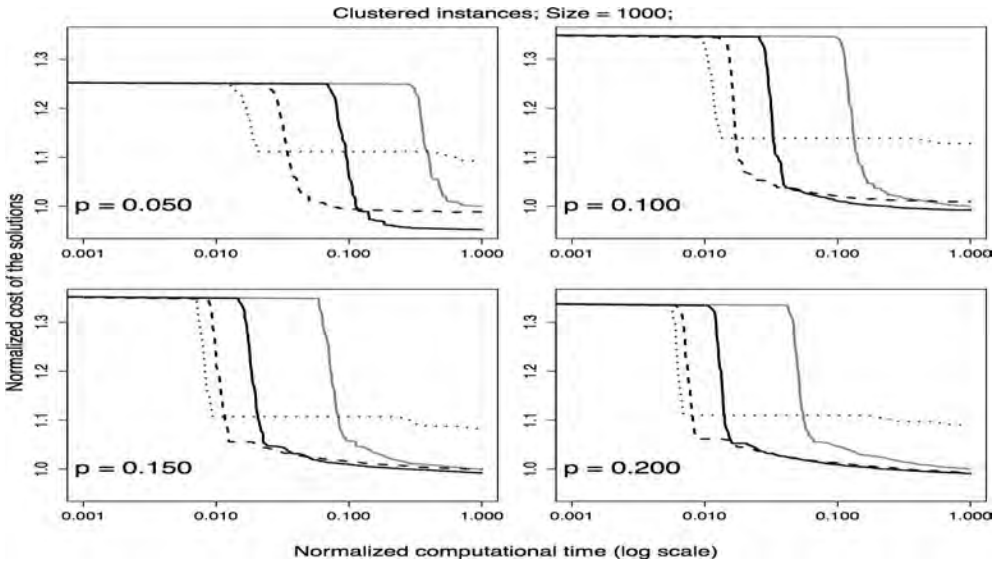
From the computational results, we can see that the ILS algorithm that uses 2.5-opt-EEais for each iteration is very effective. The average cost of the solutions obtained by ILS-2.5-opt-EEais is between 4% and 0.7% (homogeneous case), 2% and 0.8% (heterogeneous case), lower than ILS-2.5-opt-EEs-100. Note that the observed differences between the algorithms are statistically significant according to a t-test at a significance level of 0.05. For what concerns the comparison of ILS-2.5-opt-EEais with the reference algorithm ILS-2.5-opt-EEs-1000, the average solution cost of the former is between 4% and 0.09% (homogeneous case), and 6% and 0.1% (heterogeneous case), lower than 2.5-opt-EEs-1000. There is only one exception to this general trend: for $p_m = 0.200$ and $p_v = 16$, 2.5-opt-EEs-100 and 2.5-opt-EEs-1000 obtain average solution costs which are 0.7% and 0.4% lower than that of ILS-2.5-opt-EEais, respectively.

An interesting observation concerning the comparison of ILS-2.5-opt-EEs-100 and ILS-2.5-opt-EEs-1000 is that the average cost reached by the former is either better than or comparable to the latter. This is due to the fact that the use of 100 realizations instead of 1000 allows ILS-2.5-opt-EEs-100 to perform more iterations than ILS-2.5-opt-EEs-1000, which in turn results in solutions of higher quality.

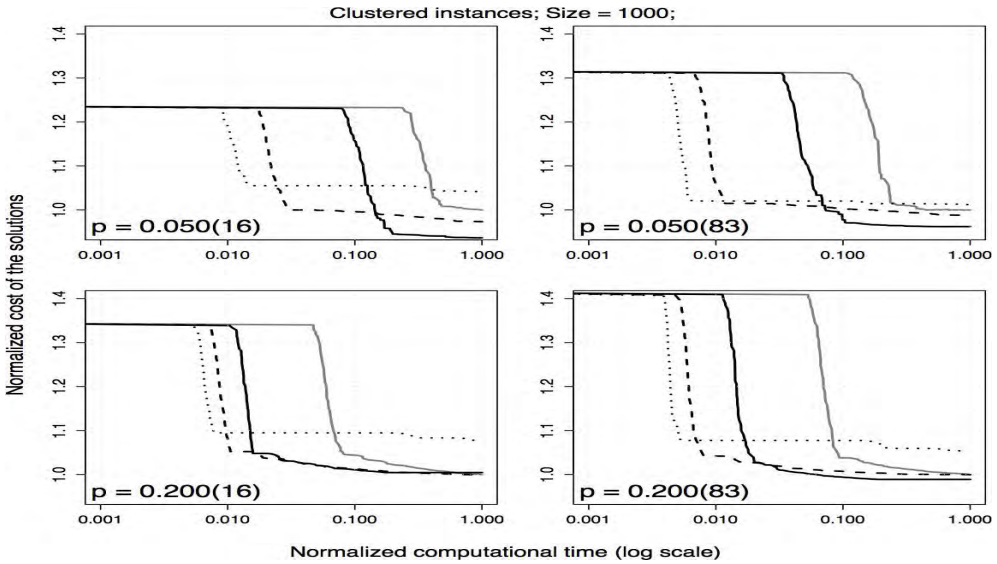
For what concerns the performance of ILS-2.5-opt-EEs-sss, the average solution cost is rather poor and significantly worse than that of all the other algorithms. This can be attributed to the fact that the particular sample size schedule is designed for a metaheuristic that is allowed to run for a relatively long computation time without an effective local search.

For the instances with high probability values ($p > 0.5$), the average cost obtained by ILS-2.5-opt-EEais is comparable to the one obtained by ILS-2.5-opt-EEs-100 and ILS-2.5-opt-EEs-1000.

Finally, we study the behavior of 2.5-opt-EEais with heuristic biasing and 2.5--opt-ACs integrated into ILS, namely, ILS-2.5-opt-EEais and ILS-2.5-opt-ACs. Since 2.5-opt-ACs is rather slow when compared to 2.5-opt-EEais, we use the fol-



(a) Homogeneous PTSP



(b) Heterogeneous PTSP

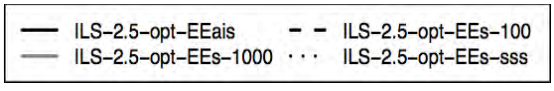


Figure 5.6: Experimental results on clustered PTSP instances of size 1000. The plots represent the cost of the solutions obtained by `ILS-2.5-opt-EEais`, `ILS-2.5-opt-EEs-100`, and `ILS-2.5-opt-EEs-sss` normalized by the one obtained by `ILS-2.5-opt-EEs-1000`. The normalization is done on an instance by instance basis for 50 instances; the normalized solution cost and the computation time are then aggregated.

Table 5.5: Experimental results for ILS-2.5-opt-EEais, ILS-2.5-opt-EEs-100, ILS--2.5-opt-EEs-1000, and ILS-2.5-opt-EEs-sss on clustered instances of size 1000. The table gives mean and standard deviation (s.d.) of final solution cost and computation time in seconds. The results are given for 50 instances at each probability level. Each algorithm is allowed to run for 100 CPU seconds.

	Algorithm	Solution Cost	
		mean	s.d.
Homogeneous PTSP			
$p = 0.050$	ILS-2.5-opt-EEais	3929167	391155
	ILS-2.5-opt-EEs-1000	4101042	525937
	ILS-2.5-opt-EEs-100	4124123	402049
	ILS-2.5-opt-EEs-sss	4591082	621302
$p = 0.100$	ILS-2.5-opt-EEais	4882334	416657
	ILS-2.5-opt-EEs-1000	4919269	420385
	ILS-2.5-opt-EEs-100	4983033	424849
	ILS-2.5-opt-EEs-sss	5550812	776072
$p = 0.150$	ILS-2.5-opt-EEais	5645487	438838
	ILS-2.5-opt-EEs-1000	5689150	446088
	ILS-2.5-opt-EEs-100	5724625	450742
	ILS-2.5-opt-EEs-sss	6148684	748337
$p = 0.200$	ILS-2.5-opt-EEais	6306761	461408
	ILS-2.5-opt-EEs-1000	6365820	469658
	ILS-2.5-opt-EEs-100	6356017	457958
	ILS-2.5-opt-EEs-sss	6939857	997612
Heterogeneous PTSP			
$p = 0.050(16)$	ILS-2.5-opt-EEais	3913474	356073
	ILS-2.5-opt-EEs-1000	4177331	658040
	ILS-2.5-opt-EEs-100	4026444	350354
	ILS-2.5-opt-EEs-sss	4333535	531310
$p = 0.050(83)$	ILS-2.5-opt-EEais	3829801	404242
	ILS-2.5-opt-EEs-1000	3962324	522985
	ILS-2.5-opt-EEs-100	3892286	361572
	ILS-2.5-opt-EEs-sss	4006539	452628
$p = 0.200(16)$	ILS-2.5-opt-EEais	6317297	404523
	ILS-2.5-opt-EEs-1000	6290327	387254
	ILS-2.5-opt-EEs-100	6270120	385406
	ILS-2.5-opt-EEs-sss	6766163	784251
$p = 0.200(83)$	ILS-2.5-opt-EEais	5933131	375063
	ILS-2.5-opt-EEs-1000	5999728	494003
	ILS-2.5-opt-EEs-100	5980738	382930
	ILS-2.5-opt-EEs-sss	6312369	709112

Table 5.6: Experimental results for `ILS-2.5-opt-EEais` and `ILS-2.5-opt-ACs` on clustered instances of size 1000. The table gives mean and standard deviation (s.d.) of final solution cost and computation time in seconds. The results are given for 10 instances at each probability level.

	Algorithm	Solution Cost		Computation Time	
		mean	s.d.	mean	s.d.
		Homogeneous PTSP			
$p = 0.050$	`ILS-2.5-opt-EEais`	3807580	461280	88.770	8.823
	`ILS-2.5-opt-ACs`	3822518	485976	8877.042	882.311
$p = 0.100$	`ILS-2.5-opt-EEais`	4768619	504847	32.824	2.340
	`ILS-2.5-opt-ACs`	4835552	538804	3282.372	233.997
$p = 0.150$	`ILS-2.5-opt-EEais`	5567761	516760	20.493	1.799
	`ILS-2.5-opt-ACs`	5609656	516846	2049.330	179.894
$p = 0.200$	`ILS-2.5-opt-EEais`	6240723	580072	15.660	1.572
	`ILS-2.5-opt-ACs`	6842928	751783	1566.023	157.196
		Heterogeneous PTSP			
p =0.050(16)	`ILS-2.5-opt-EEais`	3840549	339955	125.152	14.834
	`ILS-2.5-opt-ACs`	4216734	357264	12515.188	1483.376
p =0.050(83)	`ILS-2.5-opt-EEais`	3806451	372408	51.436	14.696
	`ILS-2.5-opt-ACs`	4911136	802057	5143.589	1469.553
p =0.200(16)	`ILS-2.5-opt-EEais`	6237114	386162	16.514	1.872
	`ILS-2.5-opt-ACs`	7840650	806949	1651.370	187.190
p =0.200(83)	`ILS-2.5-opt-EEais`	5794719	372376	16.870	1.313
	`ILS-2.5-opt-ACs`	7822102	652433	1687.035	131.296

lowing stopping criterion: `ILS-2.5-opt-EEais` is run until it performs 15 iterations and the time needed for completion is recorded. The time limit for `ILS-2.5-opt-ACs` is then set to 100 times the time taken by `ILS-2.5-opt-EEais`. We used 10 instances for each probability level. The results on clustered instances with 1000 nodes are given in Table 5.6. We refer the reader to Balaprakash et al. (2007) for the complete results.

The computational results obtained from the homogeneous and the heterogeneous instances show that `ILS-2.5-opt-EEais` is very effective with respect to both solution quality and computation time. In spite of the fact that `ILS-2.5-opt-EEais` is allowed to run for a computation time, which is two orders of magnitude less than the one of `ILS-2.5-opt-ACs`, the average cost of the solutions obtained by the former is between 0.4% to 8% and 0.8% to 26% lower than that of the latter, for the homogeneous and heterogeneous cases, respectively. For heterogeneous instances, `ILS-2.5-opt-ACs` could not even finish one complete iteration due to adoption of arbitrary precision arithmetics in `2.5-opt-ACs`, which eventually resulted in a large difference (up to 26%) in the final solution cost.

5.4 Summary

In this chapter, we integrated an adaptive sample size and an importance sampling procedure into the estimation-based local search to tackle the PTSP. The adaptive sample size procedure for delta evaluation is based on the t-test and the implementation is straightforward. The adoption of the importance sampling procedure is not a trivial task. Indeed, the main novelty of this chapter consists in customizing the importance sampling procedure for the delta evaluation.

We implemented five importance sampling variants, which mainly differ from each other in the way they use the neighborhood and problem specific knowledge. We followed a systematic methodology to assess these variants. First, we used an offline parameter tuning algorithm to fine tune the parameters of these variants. Based on the fine tuned parameter values, we provided some general hints on the range of probability values of the biased probability distribution used in the importance sampling procedure. Second, we used ANOVA to study the impact of using each importance sampling variant in `2.5-opt-EEs`. The results showed that a customized heuristic variant, which exploits neighborhood and problem specific knowledge, is very effective. We also used ANOVA to explore other important aspects of the heuristic variant: to identify an appropriate significance level for the adaptive sample size procedure and to study the robustness of the heuristic variant with respect to instance size and the distribution of nodes within a given instance. We also assessed the magnitude of reduction in variance by `2.5-opt-EEais`.

We carried out three sets of experiments. In the first set, we compared all the estimation-based algorithms. The results from the comparison between `2.5-opt-EEais` and `2.5-opt-EEas` suggested that the adoption of the importance sampling procedure in delta evaluation is very effective. The comparison of `2.5-opt-EEais` with three variants of `2.5-opt-EEs` (10, 100, and 1000 realizations) showed that the adoption of adaptive sample size and importance sampling is worthwhile as it significantly increases the solution quality especially for low node probability values and that it does so at reduced computation times when compared to using a large fixed sample size. In the second set, we compared `2.5-opt-EEais` with `2.5-opt-ACs` on instances with low probability values. The results showed that `2.5-opt-EEais` obtained a similar average solution cost of `2.5-opt-ACs` in two to three orders of magnitude less computation time. Recall that in Chapter 4 `2.5-opt-EEs` with 100 realizations obtained poorer solution cost than that of `2.5-opt-ACs` for instances with low probability values. In the third set, we performed some preliminary experiments with iterated local search,

where various iterative improvement algorithms are used as underlying local search. For a fixed computation time, the iterated local search variant that uses `2.5-opt-EE-ais` achieved average solution costs which are significantly less than that of the variants that use `2.5-opt-EEs` with 100 and 1000 realizations. These results further justified the adoption of the adaptive sample size and the importance sampling procedures.

Chapter 6

Estimation-based Metaheuristics for the PTSP

In this chapter, we develop estimation-based metaheuristics to tackle the PTSP. Before the start of our research, the best performing PTSP metaheuristic was pACS+1-shift (Bianchi, 2006; Bianchi and Gambardella, 2007), a tailor made ant colony optimization algorithm that uses `1-shift` as local search. Since `2.5-opt-EEais` obtained significant performance gains over `1-shift`, our hypothesis is that `2.5-opt-EEais` can be used to devise highly effective metaheuristics for the PTSP. In particular, we focus on three metaheuristics: iterated local search, memetic, and ant colony optimization algorithms. While iterated local search and memetic algorithms are chosen because of their high performance for the TSP, ant colony optimization is adopted due to its state-of-the-art performance on the PTSP. We also consider a random restart local search for providing a base-line reference for the three metaheuristics. The customization of these metaheuristics for the PTSP consists in adopting an estimation approach to evaluate the solution cost, `2.5-opt-EEais` as the local search, and tuning the metaheuristics parameters. We present an experimental study of the estimation-based metaheuristic algorithms on a number of instance classes. The results show that the proposed algorithms are highly effective and that they define a new state-of-the-art for the PTSP. The chapter is organized as follows. In Section 6.1, we discuss the proposed estimation-based metaheuristics. In Section 6.2, we evaluate their performances. Finally, in Section 6.3, we summarize the obtained results.

6.1 Estimation-based metaheuristics

A straightforward approach to make a metaheuristic estimation-based is to estimate the cost of solutions using Equation 4.2 given in page 52. In particular, for each solution x^i, an *unbiased* estimator $\hat{F}_{M_i}(x^i)$ of $F(x^i)$ is obtained through M_i independent realizations of $\boldsymbol{\omega}$. We use, as for `2.5-opt-EEais`, two procedures to increase the effectiveness of the estimation approach. First, the method of common random numbers to reduce the variance of the cost estimate. Second, the adaptive sample size procedure to select the most appropriate number of realizations needed for each estimation. Recall that in `2.5-opt-EEais`, the adaptive sample size procedure is implemented using the sequential application of a parametric statistical test, Student's t-test. This is also appropriate for comparing two solutions at each iteration in algorithms such as iterated local search (ILS). However, since in memetic algorithms (MAs) and ant colony optimization algorithms (ACOs) more than two solutions are compared at each iteration, we use a parametric statistical test based on analysis of variance (ANOVA) (Fisher, 1925) and Tukeys' honestly significant differences (HSD) test (Tukey, 1949) for multiple comparison. We implemented the adaptive sample size procedure as a racing algorithm (Birattari, 2004, 2009): at each iteration, the *a posteriori* solution cost of each *a priori* solution is computed sequentially on realizations. Once M_{min} realizations have been used, where M_{min} is a parameter, ANOVA is applied on a realization-by-realization basis to test the null hypothesis that the cost estimates of all solutions are equal. The rejection of the null hypothesis indicates that there is at least one solution whose cost estimate is significantly worse than the one with best cost estimate. This particular worse solution is identified using Tukeys' HSD and is eliminated from further evaluation. The procedure terminates when a single solution remains or when a maximum number M of realizations is used. If more than one solution survives at the end, the solution with the best cost estimate is selected as the best solution. We denote this procedure ANOVA-Race.

Random restart local search

RRLS consists in applying a local search algorithm a number of times, starting each time from a new initial solution, which is generated independently of the previously found local optima (see also Section 2.1 of Chapter 2). For the PTSP, we implemented an RRLS algorithm that at each iteration generates a new starting solution by using the nearest neighbor heuristic and then applies `2.5-opt-EEais`. Once each of the n possible nearest neighbor solutions has been generated, the algorithm considers a

random solution as the starting point. In order to compare the current local optimum to the best-so-far local optimum, the algorithm uses the adaptive sample size procedure with the t-test. We denote this algorithm as RRLS-EE, where EE refers to empirical estimation.

Iterated local search

ILS consists in a sequence of runs of a local search algorithm, where the initial solution of each run is obtained by a perturbation of the incumbent local optimum (see also Section 2.1 of Chapter 2). The implementation of ILS for the PTSP is a straightforward extension of TSP-specific ILS algorithms. It starts from a nearest neighbor solution and uses `2.5-opt-EEais` as the underlying local search algorithm. The perturbation consists of applying n_{db} random double-bridge moves and changing the position of $ps\%$ of n nodes, where ps and n_{db} are parameters and n is the size of the instance. This change of the position is done by picking uniformly at random $ps\%$ of n nodes, removing them from the solution and then re-inserting them according to the farthest insertion heuristic. The adoption of this hybrid perturbation is inspired by the observation that node insertion moves used in `1-shift` are very effective when probability values associated with the nodes are small—see Chapters 4, 5, Bianchi et al. (2005), and Bianchi and Campbell (2007). Hence, the proposed hybrid scheme is suitable for a wide range of probability values. The acceptance criterion compares two local optima using the adaptive sample size procedure with the t-test. The algorithm is restarted from a new nearest neighbor solution when no improvement is obtained for $rst_{it} \cdot n$ iterations, where $rst_{it} \in [0, 1]$ is a parameter. We denote this algorithm ILS-EE.

Memetic algorithms

MAs are iterative procedures that start with an initial population of solutions, which is then repeatedly improved by applying a series of genetic operators and local search (see also Section 2.1 of Chapter 2). As a starting point, we choose MAGX (Merz and Freisleben, 2001), one of the most effective memetic algorithms for the TSP. In this algorithm, the initialization phase consists in generating a number of solutions using a randomized variant of the greedy construction heuristic and applying a local search to each of them. The number of solutions is given by a parameter *pop_size*. At each iteration, *off_frac* $\times$ *pop_size* offsprings are produced, where *off_frac* $\in (0, 1]$ is a parameter. Each offspring is generated from two parent solutions using the following three step greedy recombination operator: first, all edges that are common to the

parents are copied to the offspring; second, a number (determined by a parameter p_n) of new short edges that are not common to the parents are added to the offspring; third, a number (determined by a parameter p_c) of low cost edges from the parents are copied to the offspring. A random double bridge move is used for mutating the individuals and the candidates for mutation are chosen at random. Local search is applied on any new solution that is generated by mutation or recombination. The customization of this algorithm to the PTSP consists in using the ILS composite perturbation mechanism parameterized by n_{db} and ps (see Section 6.1) as the mutation operator, ANOVA-Race at each iteration to compare the cost of the solutions, and `2.5-opt-EEais` as the local search. The mutation is performed when all solutions survive the race at a given iteration. We denote this algorithm MAGX-EE.

Ant colony optimization

We studied several ACO algorithms namely, ant colony system, $\mathcal{MAX}$–$\mathcal{MIN}$ ant system, rank-based ant system, and best-worst ant system to solve the PTSP. All these algorithms use `2.5-opt-EEais` as the local search and ANOVA-Race to compare the cost of the solutions. This study is reported in Appendix A. The results showed that all of them can be used to effectively tackle the PTSP provided that their parameter values are fine tuned. Given that the best analytical computation algorithm pACS+1-shift is based on ant colony system (ACS) algorithm (Dorigo and Gambardella, 1997), we also choose ACS. In this algorithm, at each iteration, m ants, where m is a parameter, construct solutions in the following way. Initially, each ant is placed at a randomly selected node; the choice of the ant to move from the current node i to a next node j depends on q, a random variable uniformly distributed over $[0, 1]$, and a parameter q_0. If $q \leq q_0$, then the ant chooses a node j that maximizes the product $\tau_{ij}\eta_{ij}^{\beta}$; otherwise a node j is chosen with probability $p_{ij}^{k} = \tau_{ij}\eta_{ij}^{\beta} / \sum_{l \in N_i^k} \tau_{il}\eta_{il}^{\beta}$ as the next node. The terms τ_{ij} and $\eta_{ij} = 1/c_{ij}$ are the pheromone value and the heuristic value associated with edge $\langle i, j \rangle$, respectively; β is a parameter that determines the relative influence of the heuristic information; N_i^k is the set of feasible nodes to move from node i. ACS updates pheromone in two phases. The first phase takes place when an ant moves from node i to node j: the pheromone value associated with the edge $\langle i, j \rangle$ is updated to $\tau_{ij} = (1-\varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0$. Typically, φ is set to 0.1, and τ_0, the initial value of the pheromone, is set to $1/(n \times C^{nn})$, where C^{nn} is the TSP cost of a nearest neighbor solution. The second phase takes place at the end of each iteration: the pheromone value associated with each edge $\langle i, j \rangle$ of the best-so-far solution is updated

to $\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij}^{best}$, where $\rho \in (0, 1]$ is a parameter and $\Delta\tau_{ij}^{best} = 1/C^{best}$. The value of C^{best} is set to the cost of the best-so-far solution. The PTSP-specific customization consists of using ANOVA-Race to evaluate the cost of solutions produced at each iteration and adopting `2.5-opt-EEais` as the local search, which is applied to all solutions constructed by the ants prior to the pheromone update.

6.2 Experimental analysis

In this section, we present the experimental setting and the empirical results. The goal of the experiments is to assess the performance of the proposed metaheuristics and to compare them to the state-of-the-art analytical computation algorithms for the PTSP.

6.2.1 Experimental setup

The PTSP instances used for the experiments are obtained as described in Section 5.3 on page 78 of Chapter 5. We considered the values for p_m from 0.050 to 0.200 with increments of 0.025 and from 0.3 to 0.5 with increments of 0.1; for each value of p_m, we considered four values for p_v: $\{0, 16, 50, 83\}$. We generated 50 instances for each probability level, each with 1000 nodes arranged as a number of clusters in a $10^6 \times 10^6$ square. The generated instances are grouped into three classes according to p_m: $\{0.050, 0.075, 0.100\}$ (Class I), $\{0.150, 0.175, 0.200\}$ (Class II), $\{0.300, 0.400, 0.500\}$ (Class III). This resulted in 12 levels (3 levels of p_m times 4 levels of p_v) per instance class. This grouping is based on the study on `2.5-opt-EEais` in Chapter 5, where we found that on our hardware setting, on clustered instances of size 1000, `2.5-opt-EEais` reaches local optima in approximately 6, 2, and 1 CPU second(s) on the instances grouped under Class I, Class II, and Class III, respectively.

All algorithms are implemented in `C` and compiled with `gcc`, version 3.3. The implementation of ACS-EE is based on ACOTSP (Stützle, 2002). Experiments are carried out on AMD Opteron^{TM}244 processors running at 1.75 GHz with 1 MB L2-Cache and 2 GB RAM under Rocks Cluster GNU/Linux.

We use 100 and 1000 CPU seconds as stopping criteria for each algorithm. This setup allows the algorithms to perform a relatively small and large number of iterations and it enables us to test the relative performance of the algorithms under different application scenarios, in particular, short and long computation time.

In RRLS-EE, ILS-EE, and MAGX-EE, the nearest-neighbor heuristic is used to generate initial solutions. In ACS-EE, the size of the candidate list for solution construction is set to 40 and it is generated with the quadrant nearest-neighbor strategy (Penky and

Miller, 1994; Johnson and McGeoch, 1997). The minimum number of realizations used in the adaptive sampling procedure before applying the t-test/ANOVA-Race is set to five. The null hypothesis is rejected at a significance level of 0.05. The maximum number M of realizations is set to 1000 in all algorithms. The critical values of the t-test, ANOVA, and Tukey tests are pre-computed and stored in a lookup table. Each algorithm uses a same set of realizations for all iterations. In the context of the PTSP, this strategy is more effective than changing realizations for each iteration—see Chapter 4. However, the realizations are selected randomly from the given set at each full iteration of the estimation-based metaheuristics. This is done to avoid the bias due to the order in which the realizations are generated—if the order of the realizations is the same in all iterations, then a solution whose cost estimate is better than that of other solutions only on the first few realizations will always be selected as the best due to the sequential application of the t-test/ANOVA-Race. Equation 4.1 is used for the post-evaluation of the best-so-far solutions found by all estimation-based metaheuristics.

We present only the results obtained on certain instance sets. The general trends of the results on other instances are consistent with the results presented here. The complete results are given in an online supplementary document (Balaprakash et al., 2008a):

`http://iridia.ulb.ac.be/supp/IridiaSupp2008-019/`

We also conducted a comparison with the aggregation approach (Campbell, 2006) and the progressive approximation approach (Tang and Miller-Hooks, 2004), proposed for the PTSP. Although these two approaches do not belong to the class of metaheuristics, they are considered to be viable alternatives to tackle the PTSP (Campbell and Thomas, 2008b). However, the results showed that our iterative improvement algorithm `2.5-opt-EEais` usually dominates the two methods. We report the detailed results in a technical report (Balaprakash et al., 2009b).

6.2.2 Parameter tuning

A major PTSP-specific customization of the estimation-based metaheuristics consists in finding appropriate values for their parameters. For this purpose, we used the parameter tuning algorithm, Iterative F-Race. For each of the three instance classes, we generated 120 instances (12 probability levels times 10 instances). The parameter tuning is done in two phases: in the first phase, the parameters of `2.5-opt-EEais` are fine tuned on each instance class. The obtained parameter values are reported in Table 6.1. Note that these values are different from those given in Table 5.1 of Chapter 5,

Table 6.1: Fine tuned parameter values for `2.5-opt-EEais`. This table gives the parameters considered for tuning, the range of each parameter given to Iterative F-Race, and the chosen values for each instance class.

algorithm	parameters	range	selected value		
			Class-I	Class-II	Class-III
2.5-opt-EEais	min_{is}	[0.0, 50.0]	42.0	46.0	2.40
	u	[0.0, 20.0]	13.0	16.0	5.80
	p'	[0.0, 1.0]	0.003	0.47	0.70
	p''	[0.0, 1.0]	0.92	0.67	0.95

where parameters of `2.5-opt-EEais` were fine-tuned separately on homogeneous and heterogeneous instances.

In the second phase, we tuned the parameters of the metaheuristics on each instance class for two stopping criteria: 100 and 1000 CPU seconds. In total, Iterative F-Race is run 18 times (3 metaheuristics times 3 instance classes times 2 stopping criteria), each with a computational budget of 1000 metaheuristic runs. Note that RRLS-EE is not included in the tuning because it does not have any parameters apart from the ones of `2.5-opt-EEais`. The tuning with Iterative F-Race was repeated 10 times. This was done to ensure that the observed trends in the results are not an artifact of the stochastic nature of Iterative F-Race, which is itself a stochastic algorithm. Consequently, for each instance class and stopping criterion combination, we have a set of 10 fine tuned parameter configurations for each metaheuristic. We report all the obtained parameter configurations in Balaprakash et al. (2008a).

6.2.3 Comparison between estimation-based metaheuristics

In this section, we compare the cost of the solutions obtained by ILS-EE, MAGX-EE, ACS-EE, and RRLS-EE in 100 and 1000 CPU seconds. To quantify the effectiveness of each algorithm, we study the expected solution cost of a metaheuristic, where the expectation is taken with respect to the set of 10 parameter configurations and the set of all test instances. In order to group the results obtained on different instances on each instance class, the cost of the solutions obtained by ILS-EE, MAGX-EE, and ACS-EE are normalized by the final solution cost reached by RRLS-EE. The normalization is done on an instance-by-instance basis for 50 instances for each probability level.

Figure 6.1 shows an exemplary run time development plot that characterizes the development of the solution cost of the algorithms over time up to 100 CPU seconds. The observed trends are very similar for all probability levels. From the plot, we can

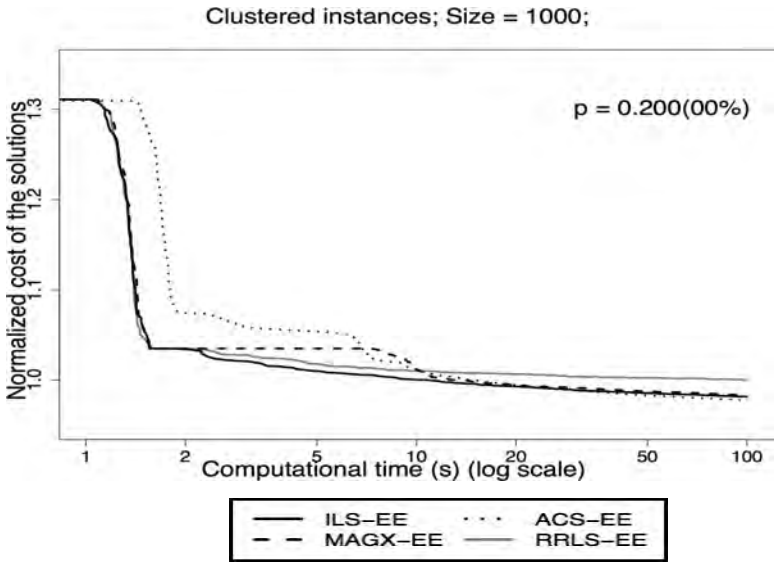
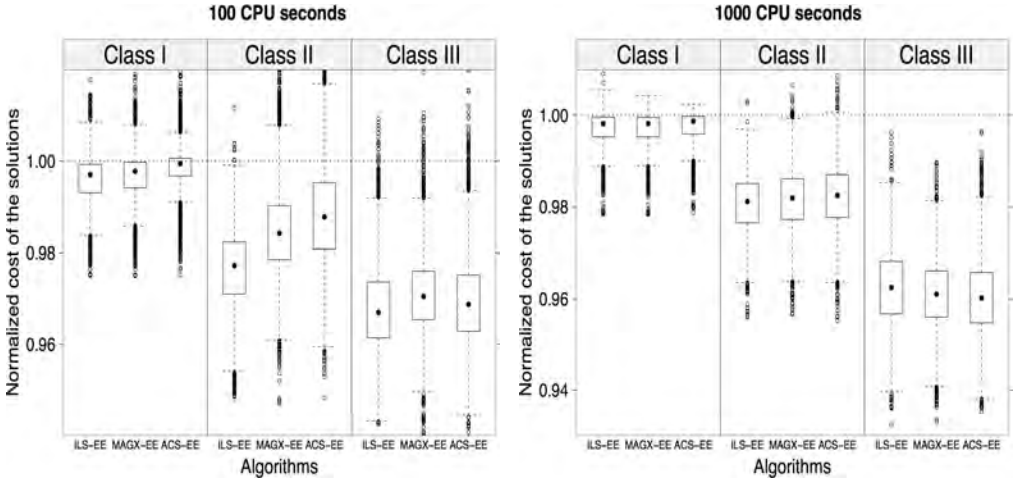


Figure 6.1: An exemplary run time development plot on clustered PTSP instances of 1000 nodes for 100 CPU seconds. The plot represents the cost of the solutions obtained by ILS-EE, MAGX-EE, ACS-EE, and RRLS-EE. The obtained solution costs of the algorithms are normalized by the final solution cost reached by RRLS-EE. The normalization is done on an instance-by-instance basis for 50 instances; the normalized solution cost is then aggregated.

observe the following general trend: the initial solution is improved by 30% to 40% in a very short computation time of 10 CPU seconds. The traces of the algorithms show that this large improvement is achieved in the very first iteration. The reason for this behavior is that the first run of `2.5-opt-EEais` on the initial solution allows each algorithm to obtain a large improvement. Note that in the case of population-based algorithms, the plots take into account the improvement incurred by the first local search applied to an individual of the population. Further improvements in the following iterations are considerably smaller than that in the first iteration. When going from 100 to 1000 CPU seconds, all four algorithms achieved an average solution cost that is less than that for 100 CPU seconds—see absolute values reported in Balaprakash et al. (2008a). The improvements in solution quality for this one order of magnitude increase in computation time are up to 3%. In particular, MAGX-EE and ACS-EE highly profit from the longer computation time.

Figure 6.2 shows the box plots of the solution cost of the algorithms after 100 and 1000 CPU seconds. From the plots, we can observe that the solution cost obtained by the baseline algorithm RRLS-EE is significantly worse than that of all other algorithms across most probability levels. The poor relative performance of RRLS-EE is ascribed to the fact that it does not exploit the solution components from the best-so-far local



(a) (b)

Figure 6.2: Experimental results on clustered PTSP instances of 1000 nodes. The box plots represent the cost of the solutions obtained by ILS-EE, MAGX-EE, and ACS-EE. The obtained solution costs of the algorithms are normalized by the final solution cost reached by RRLS-EE. The normalization is done on an instance-by-instance basis for 50 instances; the normalized solution cost is then aggregated. The dotted horizontal line denotes therefore the final cost of RRLS-EE.

optimum. An interesting observation from the plot is that the relative difference in the solution cost between RRLS-EE and the other algorithms increases with an increase of the mean probability value p_m. (Recall that p_m increases from Class I to Class III). This shows that for instances with small p_m, it is feasible to find high quality solutions by simply restarting `2.5-opt-EEais` with different nearest neighbor solutions. However, for instances with large values of p_m, besides `2.5-opt-EEais`, the use of sophisticated metaheuristics is crucial to find high quality solutions.

Table 6.2 reports the observed relative difference between the solution costs obtained by the algorithms for 100 CPU seconds with a 95% confidence bound obtained through a t-test. The results show that ILS-EE is more effective than the other algorithms. The average cost of the solutions obtained by ILS-EE is up to 0.87% and 1.27% less than that of MAGX-EE and ACS-EE, respectively. The observed differences are significant according to a t-test except for a few probability levels, where the observed differences between the algorithms are not significant or ILS-EE obtains solution costs that are slightly worse than those of ACS-EE.

The results for 1000 CPU seconds are given in Table 6.3. The results show that ILS-EE and MAGX-EE are particularly effective on instances of Class I and Class II. On Class I instances, ILS-EE and MAGX-EE obtain solutions whose averages are 0.04%

less than that of ACS-EE, respectively. On Class II instances, ILS-EE is more effective than MAGX-EE and ACS-EE: the average solution cost obtained by ILS-EE is 0.09% and 0.16% less than that of ACS-EE and MAGX-EE, respectively. However, on Class III instances, ACS-EE achieves an average solution cost which is between 0.22% and 0.05% less than that of ILS-EE and MAGX-EE, respectively.

The effectiveness of ILS-EE under short computation time can be explained as follows: since `2.5-opt-EEais` is applied on a single solution at each iteration, the computation time per iteration is lower than that of MAGX-EE and ACS-EE. As a consequence, ILS-EE can do more iterations and finds high quality solutions. However, it seems that the high computation time per iteration due to the adoption of population of solutions is not a major issue under long computation time. While ACS-EE has been shown to be effective for the PTSP—see Annex A, the reason for high performance of MAGX-EE can be ascribed to the fact that it operates exclusively on a population of high quality local optima obtained by `2.5-opt-EEais`. Also note that MAGX-EE is derived from MAs of Merz and Freisleben (2001), which is one of the most effective algorithm for the related TSP (Hoos and Stützle, 2005).

We also compared ILS-EE, MAGX-EE, and ACS-EE to previously proposed estimation-based simulated annealing algorithms. We implemented two simulated annealing algorithms built on top of `2.5-opt-EEais`. The first algorithm uses an acceptance criterion as in Bowler et al. (2003). The second algorithm adopts a sample size scheme and the acceptance criterion as in a general purpose stochastic simulated annealing algorithm of Gutjahr and Pflug (1996) and Gutjahr (2004). The results showed that ILS-EE, MAGX-EE, and ACS-EE completely outperform the two estimation-based simulated annealing algorithms. We refer the reader to Balaprakash et al. (2009b) for the complete results.

Table 6.2: Comparison of the average cost obtained by ILS-EE, MAGX-EE, and ACS-EE on clustered instances with 1000 nodes for 100 CPU seconds. See Footnote 1 for an explanation of the contents and the typographic conventions adopted in the table.

		ILS-EE vs. MAGX-EE		ILS-EE vs. ACS-EE		MAGX-EE vs. ACS-EE	
	p	d	CI	d	CI	d	CI
Class I	0.050(00%)	**−0.06**	[−0.10, −0.03]	**−0.50**	[−0.59, −0.40]	**−0.43**	[−0.53, −0.33]
	0.050(16%)	**−0.07**	[−0.11, −0.02]	**−0.20**	[−0.25, −0.15]	**−0.13**	[−0.19, −0.08]
	0.050(50%)	−0.03	[−0.06, +0.00]	−0.02	[−0.05, +0.01]	+0.01	[−0.02, +0.04]
	0.050(83%)	−0.01	[−0.03, +0.01]	−0.00	[−0.02, +0.02]	+0.01	[−0.02, +0.03]
	0.075(00%)	**−0.09**	[−0.12, −0.06]	**−0.36**	[−0.40, −0.33]	**−0.27**	[−0.31, −0.24]
	0.075(16%)	**−0.11**	[−0.14, −0.08]	**−0.32**	[−0.35, −0.29]	**−0.22**	[−0.25, −0.18]
	0.075(50%)	**−0.10**	[−0.13, −0.07]	**−0.24**	[−0.26, −0.21]	**−0.14**	[−0.17, −0.10]
	0.075(83%)	**−0.03**	[−0.05, −0.01]	**−0.07**	[−0.09, −0.05]	**−0.04**	[−0.06, −0.02]
	0.100(00%)	**−0.18**	[−0.21, −0.14]	**−0.64**	[−0.68, −0.60]	**−0.46**	[−0.51, −0.42]
	0.100(16%)	**−0.16**	[−0.19, −0.13]	**−0.63**	[−0.66, −0.59]	**−0.47**	[−0.51, −0.43]
	0.100(50%)	**−0.16**	[−0.19, −0.12]	**−0.41**	[−0.44, −0.37]	**−0.25**	[−0.29, −0.21]
	0.100(83%)	**−0.06**	[−0.09, −0.03]	**−0.28**	[−0.32, −0.24]	**−0.22**	[−0.25, −0.19]
	overall	**−0.09**	[−0.10, −0.08]	**−0.31**	[−0.32, −0.29]	**−0.22**	[−0.23, −0.21]
Class II	0.150(00%)	**−0.23**	[−0.26, −0.20]	**−0.44**	[−0.48, −0.40]	**−0.21**	[−0.26, −0.17]
	0.150(16%)	**−0.45**	[−0.48, −0.41]	**−0.77**	[−0.82, −0.72]	**−0.33**	[−0.38, −0.27]
	0.150(50%)	**−1.28**	[−1.35, −1.21]	**−1.84**	[−1.92, −1.75]	**−0.57**	[−0.67, −0.46]
	0.150(83%)	**−1.79**	[−1.89, −1.69]	**−2.67**	[−2.82, −2.52]	**−0.90**	[−1.07, −0.72]
	0.175(00%)	**−0.19**	[−0.23, −0.15]	**−0.38**	[−0.43, −0.33]	**−0.19**	[−0.24, −0.15]
	0.175(16%)	**−0.44**	[−0.49, −0.40]	**−0.74**	[−0.79, −0.69]	**−0.30**	[−0.35, −0.24]
	0.175(50%)	**−1.12**	[−1.18, −1.06]	**−1.57**	[−1.63, −1.50]	**−0.45**	[−0.53, −0.37]
	0.175(83%)	**−1.59**	[−1.67, −1.51]	**−2.30**	[−2.41, −2.19]	**−0.72**	[−0.85, −0.59]
	0.200(00%)	**−0.25**	[−0.29, −0.21]	**−0.32**	[−0.36, −0.27]	**−0.06**	[−0.11, −0.02]
	0.200(16%)	**−0.41**	[−0.45, −0.36]	**−0.62**	[−0.67, −0.56]	**−0.21**	[−0.26, −0.15]
	0.200(50%)	**−1.04**	[−1.09, −0.98]	**−1.43**	[−1.49, −1.37]	**−0.40**	[−0.48, −0.32]
	0.200(83%)	**−1.57**	[−1.64, −1.50]	**−2.14**	[−2.21, −2.06]	**−0.57**	[−0.66, −0.48]
	overall	**−0.86**	[−0.89, −0.84]	**−1.27**	[−1.30, −1.24]	**−0.41**	[−0.44, −0.38]
Class III	0.300(00%)	**−0.17**	[−0.23, −0.11]	+0.02	[−0.04, +0.07]	*+0.19*	[+0.14, +0.24]
	0.300(16%)	**−0.17**	[−0.23, −0.11]	−0.03	[−0.09, +0.03]	*+0.13*	[+0.08, +0.19]
	0.300(50%)	**−0.43**	[−0.51, −0.36]	**−0.70**	[−0.78, −0.63]	**−0.27**	[−0.34, −0.20]
	0.300(83%)	**−0.77**	[−0.85, −0.69]	**−1.36**	[−1.45, −1.27]	**−0.59**	[−0.69, −0.50]
	0.400(00%)	**−0.16**	[−0.22, −0.10]	*+0.16*	[+0.09, +0.22]	*+0.32*	[+0.27, +0.37]
	0.400(16%)	**−0.18**	[−0.25, −0.10]	*+0.15*	[+0.08, +0.23]	*+0.33*	[+0.28, +0.38]
	0.400(50%)	**−0.29**	[−0.36, −0.21]	**−0.14**	[−0.22, −0.07]	*+0.15*	[+0.09, +0.21]
	0.400(83%)	**−0.48**	[−0.56, −0.41]	**−0.65**	[−0.74, −0.57]	**−0.17**	[−0.25, −0.09]
	0.500(00%)	**−0.18**	[−0.25, −0.11]	*+0.19*	[+0.11, +0.26]	*+0.37*	[+0.32, +0.42]
	0.500(16%)	**−0.20**	[−0.28, −0.13]	*+0.21*	[+0.13, +0.28]	*+0.41*	[+0.36, +0.46]
	0.500(50%)	**−0.14**	[−0.22, −0.06]	*+0.23*	[+0.15, +0.31]	*+0.37*	[+0.31, +0.43]
	0.500(83%)	**−0.54**	[−0.62, −0.46]	**−0.28**	[−0.36, −0.20]	*+0.27*	[+0.20, +0.33]
	overall	**−0.31**	[−0.33, −0.29]	**−0.19**	[−0.21, −0.16]	*+0.12*	[+0.10, +0.14]

[1] For a given comparison A vs. B, the table reports the observed relative difference d between the two algorithms A and B and the 95% confidence interval CI obtained through the t-test. If the value is positive, algorithm A obtained an average cost that is larger than the one obtained by algorithm B. In this case, the value is typeset in italics if it is significantly different from zero according to the t-test at a confidence level of 95%. If the value is negative, algorithm A obtained an average cost that is smaller than the one obtained by algorithm B. In this case, the value is typeset in boldface if it is significantly different from zero according to the t-test, at a confidence level of 95%.

Table 6.3: Comparison of the average cost obtained by ILS-EE, MAGX-EE, and ACS-EE on clustered instances with 1000 nodes for 1000 CPU seconds. Typographic conventions are the same as in Table 6.2.

		ILS-EE vs. MAGX-EE		ILS-EE vs. ACS-EE		MAGX-EE vs. ACS-EE	
	p	d	CI	d	CI	d	CI
Class I	0.050(00%)	+0.00	[−0.01, +0.01]	**−0.06**	[−0.06, −0.05]	**−0.06**	[−0.06, −0.05]
	0.050(16%)	*+0.01*	[+0.00, +0.01]	**−0.02**	[−0.03, −0.01]	**−0.03**	[−0.03, −0.02]
	0.050(50%)	*+0.01*	[+0.00, +0.01]	−0.00	[−0.01, +0.00]	**−0.01**	[−0.01, −0.01]
	0.050(83%)	*+0.01*	[+0.00, +0.01]	+0.00	[−0.00, +0.01]	**−0.01**	[−0.01, −0.00]
	0.075(00%)	*+0.02*	[+0.01, +0.03]	**−0.07**	[−0.08, −0.06]	**−0.08**	[−0.09, −0.08]
	0.075(16%)	+0.00	[−0.01, +0.01]	**−0.06**	[−0.07, −0.06]	**−0.06**	[−0.07, −0.06]
	0.075(50%)	+0.00	[−0.00, +0.01]	**−0.02**	[−0.03, −0.02]	**−0.03**	[−0.03, −0.02]
	0.075(83%)	*+0.01*	[+0.00, +0.01]	−0.00	[−0.01, +0.00]	**−0.01**	[−0.01, −0.00]
	0.100(00%)	**−0.02**	[−0.03, −0.01]	**−0.08**	[−0.10, −0.06]	**−0.06**	[−0.07, −0.04]
	0.100(16%)	**−0.02**	[−0.02, −0.01]	**−0.11**	[−0.12, −0.09]	**−0.09**	[−0.10, −0.08]
	0.100(50%)	**−0.01**	[−0.01, −0.00]	**−0.05**	[−0.06, −0.05]	**−0.05**	[−0.06, −0.04]
	0.100(83%)	+0.00	[−0.00, +0.01]	**−0.02**	[−0.02, −0.01]	**−0.02**	[−0.02, −0.02]
	overall	+0.00	[−0.00, +0.00]	**−0.04**	[−0.04, −0.04]	**−0.04**	[−0.04, −0.04]
Class II	0.150(00%)	*+0.04*	[+0.02, +0.06]	*+0.05*	[+0.03, +0.07]	+0.01	[−0.01, +0.03]
	0.150(16%)	−0.02	[−0.04, +0.00]	**−0.06**	[−0.08, −0.04]	**−0.04**	[−0.06, −0.02]
	0.150(50%)	**−0.17**	[−0.19, −0.15]	**−0.33**	[−0.35, −0.30]	**−0.16**	[−0.18, −0.13]
	0.150(83%)	**−0.22**	[−0.25, −0.19]	**−0.40**	[−0.43, −0.38]	**−0.18**	[−0.22, −0.15]
	0.175(00%)	*+0.05*	[+0.02, +0.08]	*+0.06*	[+0.03, +0.09]	+0.01	[−0.02, +0.04]
	0.175(16%)	**−0.04**	[−0.07, −0.02]	**−0.05**	[−0.07, −0.02]	−0.00	[−0.03, +0.03]
	0.175(50%)	**−0.18**	[−0.20, −0.15]	**−0.22**	[−0.24, −0.19]	**−0.04**	[−0.07, −0.01]
	0.175(83%)	**−0.22**	[−0.24, −0.19]	**−0.38**	[−0.41, −0.35]	**−0.16**	[−0.19, −0.13]
	0.200(00%)	*+0.07*	[+0.04, +0.10]	*+0.06*	[+0.03, +0.09]	−0.01	[−0.03, +0.02]
	0.200(16%)	+0.00	[−0.03, +0.03]	−0.01	[−0.04, +0.02]	−0.01	[−0.03, +0.02]
	0.200(50%)	**−0.17**	[−0.19, −0.14]	**−0.20**	[−0.23, −0.18]	**−0.04**	[−0.06, −0.01]
	0.200(83%)	**−0.24**	[−0.26, −0.21]	**−0.41**	[−0.44, −0.39]	**−0.18**	[−0.21, −0.15]
	overall	**−0.09**	[−0.10, −0.08]	**−0.16**	[−0.17, −0.15]	**−0.07**	[−0.07, −0.06]
Class III	0.300(00%)	*+0.18*	[+0.14, +0.23]	*+0.28*	[+0.23, +0.32]	*+0.09*	[+0.06, +0.12]
	0.300(16%)	*+0.14*	[+0.10, +0.18]	*+0.20*	[+0.16, +0.24]	*+0.06*	[+0.03, +0.09]
	0.300(50%)	+0.03	[−0.01, +0.07]	**−0.07**	[−0.11, −0.02]	**−0.10**	[−0.13, −0.06]
	0.300(83%)	**−0.09**	[−0.13, −0.05]	**−0.50**	[−0.55, −0.45]	**−0.41**	[−0.46, −0.36]
	0.400(00%)	*+0.22*	[+0.17, +0.27]	*+0.43*	[+0.38, +0.48]	*+0.21*	[+0.18, +0.24]
	0.400(16%)	*+0.23*	[+0.18, +0.28]	*+0.40*	[+0.34, +0.45]	*+0.17*	[+0.13, +0.20]
	0.400(50%)	*+0.18*	[+0.13, +0.22]	*+0.23*	[+0.18, +0.28]	*+0.05*	[+0.02, +0.08]
	0.400(83%)	*+0.10*	[+0.05, +0.15]	−0.02	[−0.08, +0.03]	**−0.12**	[−0.16, −0.08]
	0.500(00%)	*+0.29*	[+0.24, +0.35]	*+0.53*	[+0.47, +0.59]	*+0.23*	[+0.20, +0.27]
	0.500(16%)	*+0.27*	[+0.21, +0.32]	*+0.49*	[+0.43, +0.55]	*+0.22*	[+0.19, +0.25]
	0.500(50%)	*+0.28*	[+0.22, +0.33]	*+0.47*	[+0.41, +0.53]	*+0.19*	[+0.16, +0.22]
	0.500(83%)	*+0.14*	[+0.09, +0.19]	*+0.21*	[+0.15, +0.26]	*+0.07*	[+0.03, +0.10]
	overall	*+0.16*	[+0.15, +0.18]	*+0.22*	[+0.20, +0.23]	*+0.05*	[+0.04, +0.07]

Table 6.4: Fine tuned parameter values for each algorithm in 1000 CPU seconds. This table gives the parameters considered for tuning, the range of each parameter given to Iterative F-Race, and the chosen values for each instance class. For an explanation of the parameters, see Section 6.1.

algorithm	parameters	range	selected value		
			Class-I	Class-II	Class-III
ILS-EE	ps	$[1, 90]$	1	1	1
	n_{db}	$[1, 50]$	1	1	1
	rst_{itr}	$[0, 1]$	0.089	0.34	0.69
MAGX-EE	pop_size	$[3, 15]$	4	3	8
	off_frac	$[0.1, 1.0]$	0.24	0.67	0.16
	ps	$[1, 90]$	9	2	1
	n_{db}	$[1, 50]$	1	1	1
	p_n	$[0.1, 1.0]$	0.33	0.11	0.29
	p_c	$[0.1, 1.0]$	0.92	0.97	0.72
ACS-EE	m	$[3, 15]$	10	8	11
	q_0	$[0.0, 1.0]$	1.0	1.0	0.97
	β	$[0.0, 5.0]$	0.18	0.00	0.83
	ρ	$[0.001, 1.0]$	0.31	0.30	0.61

6.2.4 Comparison with analytical computation algorithms

Currently, the best performing analytical computation algorithms are pACS+1-shift (Bianchi and Gambardella, 2007) and HybMSPSO (Marinakis and Marinaki, 2010). In this section, we compare ILS-EE, MAGX-EE and ACS-EE to these two algorithms.

First, we focus on the comparison with pACS+1-shift. The algorithms are evaluated on the instances adopted by Bianchi (2006) to show the effectiveness of pACS+1-shift. These instances are generated from the well-known TSPLIB instances, `kroA100`, `eil101`, `ch150`, `d198`, `lin318`, `att532`, and `rat783`; probabilities associated with the nodes are generated by the beta distribution using the same parameters as described for the previous set of experiments. We used the stopping criterion suggested by Bianchi and Gambardella (2007) and Bianchi (2006): each algorithm is allowed to run for a computation time of $n^2/100$ CPU seconds. This stopping criterion allows the algorithms to run for a very long computation time. Concerning the parameter configuration for pACS+1-shift, we use the one given in Bianchi and Gambardella (2007) and Bianchi (2006). For ILS-EE, MAGX-EE, and ACS-EE, we choose the parameter values that produced the lowest average cost across the 10 tuning runs for 1000 CPU seconds in Section 6.2.3. They are listed in Table 6.4.

A problem in pACS+1-shift is that the underlying local search, `1-shift`, as discussed in Chapter 4, suffers from numerical precision problems for large instances.

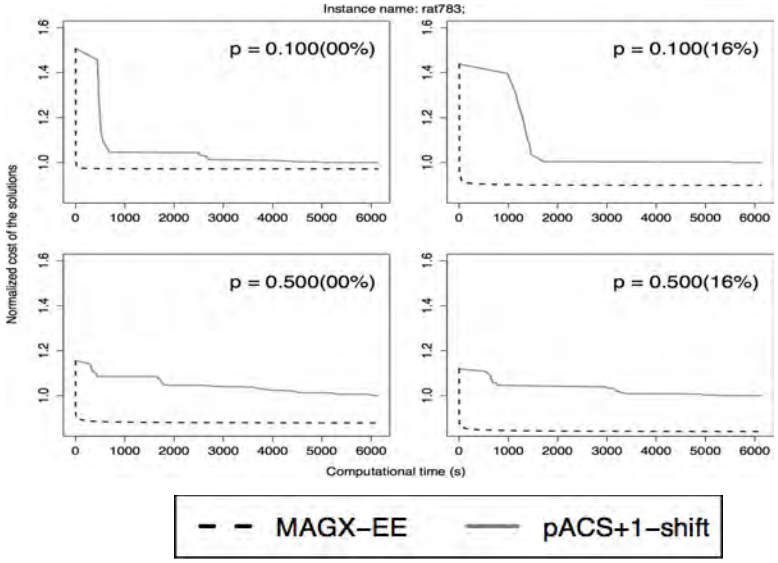


Figure 6.3: Experimental results on the instance `rat783`. The plots represent the development of the solution cost over time for MAGX-EE and pACS+1-shift. The obtained solution costs of the two algorithms are normalized by the final solution cost reached by pACS+1-shift. The normalization is performed on a run-by-run basis for 10 runs; the normalized solution cost is then aggregated.

This problem can been addressed by resorting to computationally expensive arbitrary precision arithmetics. However, for the given stopping criterion, the usage of the arbitrary precision arithmetics in pACS+1-shift does not even allow the algorithm to complete the first iteration. Therefore, we compared the solution costs obtained for the probability levels at which the numerical problems do not occur.

Figure 6.3 shows exemplary run time development plots on PTSP instances generated from `rat783`. Since there is no recognizable visual difference among the estimation-based algorithms, only MAGX-EE is chosen for the plots. We can see that MAGX-EE (and also ILS-EE and ACS-EE) obtains high quality solutions in a very short time. More precisely, the estimation-based algorithms reached the average solution cost of pACS+1-shift in approximately two to three orders of magnitude less CPU time. The trend is quite similar for other instances; the plots are shown in Balaprakash et al. (2008a).

Table 6.5 reports the average difference between the final solution costs obtained by MAGX-EE and pACS+1-shift on instances generated from `rat783`, `d198`, and `kroA100`. The results of ILS-EE and ACS-EE are quite similar—see Balaprakash et al. (2008a). On all probability levels for `rat783`, the estimation-based algorithms achieve average solution costs that are significantly less than that of pACS+1-shift. The average im-

provements are up to 4.90% (MAGX-EE), 7.35% (MAGX-EE), and 12.39% (ACS-EE) on Class I, II, III instances, respectively. We can observe a similar trend but smaller improvements for `d198` and `kroA100` instances. There are a few exceptions at some probability levels. These results also show a general trend in which the differences between the average solution costs of the estimation-based algorithms and pACS+1-shift increase with an increase in the instance size and also with an increase in the probability level (the only exception is on Class II instances of `kroA100`).

The high performance of the estimation-based algorithms is mainly due to the adoption of the effective iterative improvement algorithm as local search: since, for the given computation time, `2.5-opt-EEais` is much faster than `1-shift`, the estimation-based algorithms perform a much larger number of local searches than pACS+1-shift. The average number of local searches performed by each algorithm is shown in Table 6.6.

For the comparison between the estimation-based algorithms and HybMSPSO, we adopt the instances used by Marinakis and Marinaki (2010), on which the authors showed that HybMSPSO is more effective than pACS+1-shift for high probability values. These are homogeneous PTSP instances generated from the TSP instances `kroA100`, `eil101`, `ch150`, `d198`, and `rat783` with some probability values between 0.1 and 0.9. On these instances, we made a direct comparison of the cost of the solutions obtained by the estimation-based algorithms to the ones of HybMSPSO reported in Marinakis and Marinaki (2010). Note that we compare the average cost of the estimation-based algorithms to the reported best solution cost of HybMSPSO. This might possibly introduce a bias in favor of HybMSPSO. Table 6.7 highlights the results from the comparison on instances `kroA100`, `d198`, and `rat783`. The trend is similar for all other instances. Absolute values are reported in Balaprakash et al. (2008a).

The results show that the estimation-based algorithms are more effective than HybMSPSO and that they obtain average solution costs which are significantly less than the cost of the best solution obtained by HybMSPSO on a wide range of instance sizes and probability levels. On the largest instance `rat783`, the observed average difference ranges between 0.32% and 10.23%. On the instance `d198` and `kroA100`, for most probability levels, the observed differences are significant and the improvements are up to 1.03% and 2.58%, respectively.

Table 6.5: Comparison of the average cost obtained by MAGX-EE and pACS+1-shift on `rat783`, `d198`, and `kroA100`. The results are obtained over 10 independent runs. For certain probability levels in large instances, pACS+1-shift suffers from numerical problems, where the comparison is not meaningful. Those cases are marked as −. Typographic conventions are the same as in Table 6.2.

		rat783		d198		kroA100	
	p	d	CI	d	CI	d	CI
Class I	0.050(00%)	+0.04	[−0.08, +0.16]	*+0.07*	[+0.05, +0.08]	*+0.08*	[+0.06, +0.11]
	0.050(16%)	**−7.57**	[−9.91, −5.23]	**−0.16**	[−0.23, −0.09]	**−0.29**	[−0.42, −0.16]
	0.050(50%)	-	-	**−0.62**	[−1.09, −0.15]	**−0.99**	[−1.52, −0.46]
	0.050(83%)	-	-	**−1.84**	[−2.87, −0.81]	−1.34	[−2.84, +0.17]
	0.075(00%)	**−2.33**	[−3.47, −1.19]	*+0.06*	[+0.05, +0.07]	*+0.08*	[+0.05, +0.10]
	0.075(16%)	**−6.81**	[−9.98, −3.64]	**−0.02**	[−0.03, −0.01]	**−0.03**	[−0.06, −0.01]
	0.075(50%)	-	-	**−0.27**	[−0.41, −0.12]	**−0.36**	[−0.48, −0.24]
	0.075(83%)	-	-	**−1.92**	[−2.65, −1.19]	**−0.42**	[−0.60, −0.24]
	0.100(00%)	**−2.82**	[−3.49, −2.14]	*+0.08*	[+0.07, +0.10]	*+0.08*	[+0.06, +0.10]
	0.100(16%)	**−9.91**	[−12.98, −6.85]	*+0.03*	[+0.02, +0.04]	+0.00	[−0.01, +0.02]
	0.100(50%)	-	-	**−0.16**	[−0.23, −0.09]	**−0.23**	[−0.31, −0.15]
	0.100(83%)	-	-	**−3.85**	[−5.03, −2.66]	**−0.32**	[−0.41, −0.23]
	overall	**−4.90**	[−6.06, −3.74]	**−0.72**	[−0.96, −0.47]	**−0.31**	[−0.45, −0.17]
Class II	0.150(00%)	**−4.89**	[−6.13, −3.64]	*+0.05*	[+0.04, +0.07]	*+0.03*	[+0.02, +0.05]
	0.150(16%)	**−8.82**	[−10.17, −7.47]	*+0.04*	[+0.02, +0.07]	*+0.02*	[+0.01, +0.03]
	0.150(50%)	-	-	**−0.06**	[−0.08, −0.04]	**−0.09**	[−0.11, −0.06]
	0.150(83%)	-	-	**−4.85**	[−6.11, −3.59]	**−0.23**	[−0.31, −0.15]
	0.175(00%)	**−5.34**	[−6.21, −4.46]	*+0.05*	[+0.03, +0.07]	*+0.03*	[+0.01, +0.04]
	0.175(16%)	**−8.96**	[−10.61, −7.31]	+0.02	[−0.02, +0.06]	*+0.02*	[+0.00, +0.04]
	0.175(50%)	-	-	**−0.06**	[−0.10, −0.01]	**−0.05**	[−0.07, −0.02]
	0.175(83%)	-	-	**−6.33**	[−8.05, −4.62]	**−0.18**	[−0.23, −0.13]
	0.200(00%)	**−5.64**	[−6.24, −5.04]	*+0.06*	[+0.03, +0.09]	*+0.02*	[+0.01, +0.04]
	0.200(16%)	**−10.46**	[−11.92, −8.99]	+0.02	[−0.00, +0.04]	*+0.02*	[+0.01, +0.03]
	0.200(50%)	-	-	−1.29	[−3.22, +0.64]	**−0.04**	[−0.07, −0.00]
	0.200(83%)	-	-	**−6.53**	[−8.16, −4.90]	**−0.52**	[−0.89, −0.15]
	overall	**−7.35**	[−8.05, −6.65]	**−1.57**	[−2.09, −1.06]	**−0.08**	[−0.12, −0.04]
Class III	0.300(00%)	**−9.69**	[−10.63, −8.74]	+0.04	[−0.00, +0.09]	*+0.05*	[+0.02, +0.07]
	0.300(16%)	**−12.32**	[−13.54, −11.09]	−0.08	[−0.19, +0.03]	*+0.01*	[+0.00, +0.02]
	0.300(50%)	-	-	−1.14	[−2.78, +0.50]	**−0.02**	[−0.02, −0.01]
	0.300(83%)	-	-	**−7.57**	[−9.07, −6.07]	**−1.26**	[−2.14, −0.38]
	0.400(00%)	**−10.28**	[−11.60, −8.95]	**−0.23**	[−0.43, −0.02]	+0.00	[−0.00, +0.01]
	0.400(16%)	**−13.44**	[−14.35, −12.52]	**−0.69**	[−1.21, −0.18]	+0.01	[−0.00, +0.02]
	0.400(50%)	-	-	**−4.95**	[−7.27, −2.63]	−0.36	[−0.82, +0.09]
	0.400(83%)	-	-	**−8.37**	[−9.69, −7.05]	**−2.40**	[−3.78, −1.02]
	0.500(00%)	**−12.09**	[−13.19, −11.00]	**−0.34**	[−0.61, −0.07]	+0.00	[−0.00, +0.00]
	0.500(16%)	**−15.91**	[−16.88, −14.93]	**−1.23**	[−1.75, −0.71]	+0.00	[−0.00, +0.01]
	0.500(50%)	-	-	**−6.18**	[−7.72, −4.63]	**−1.12**	[−1.81, −0.42]
	0.500(83%)	-	-	**−8.64**	[−10.42, −6.86]	**−4.82**	[−7.03, −2.61]
	overall	**−12.29**	[−12.94, −11.63]	**−3.28**	[−3.97, −2.60]	**−0.83**	[−1.15, −0.50]

Table 6.6: The average number of local searches performed by the estimation-based algorithms and pACS+1-shift over 10 independent runs on instance `rat783`.

p	ILS-EE	MAGX-EE	ACS-EE	pACS+1-shift
0.050(00%)	576	521	468	14
0.050(16%)	553	573	613	5
0.075(00%)	2302	1903	2323	15
0.075(16%)	929	928	896	5
0.100(00%)	4714	3827	6286	16
0.100(16%)	1505	1530	1457	6
0.150(00%)	7790	7709	13914	24
0.150(16%)	2712	2712	3347	10
0.175(00%)	11191	11353	21458	26
0.175(16%)	4378	4376	6441	10
0.200(00%)	15692	15878	27183	26
0.200(16%)	6310	6435	10232	10
0.300(00%)	24777	23924	34864	25
0.300(16%)	12831	12958	15803	12
0.400(00%)	41546	35626	49138	29
0.400(16%)	27886	26479	33784	15
0.500(00%)	77195	53900	81400	34
0.500(16%)	50423	41128	58192	16

Table 6.7: Comparison of the average cost obtained by ILS-EE, MAGX-EE, ACS-EE, and HybMSPSO on instances `kroA100`, `d198`, and `rat783`. For ILS-EE, MAGX-EE, and ACS-EE the results are obtained over 10 independent runs. For HybMSPSO, the values are taken directly from Marinakis and Marinaki (2010). Typographic conventions are the same as in Table 6.2.

		ILS-EE vs. HybMSPSO		MAGX-EE vs. HybMSPSO		ACS-EE vs. HybMSPSO	
	p	d	[95% CI]	d	[95% CI]	d	[95% CI]
kroA100	0.100(00%)	**−0.36**	[−0.39, −0.32]	**−0.37**	[−0.40, −0.35]	**−0.40**	[−0.42, −0.38]
	0.200(00%)	**−0.07**	[−0.10, −0.05]	**−0.08**	[−0.10, −0.07]	+0.04	[−0.11, +0.20]
	0.300(00%)	*+0.06*	[+0.03, +0.10]	+0.01	[−0.01, +0.04]	−0.01	[−0.03, +0.02]
	0.400(00%)	**−0.92**	[−0.99, −0.85]	**−1.00**	[−1.00, −0.99]	**−1.00**	[−1.00, −0.99]
	0.500(00%)	−0.03	[−0.10, +0.04]	**−0.08**	[−0.08, −0.07]	**−0.08**	[−0.08, −0.07]
	0.600(00%)	**−1.91**	[−1.92, −1.90]	**−1.91**	[−1.92, −1.91]	**−1.91**	[−1.92, −1.91]
	0.800(00%)	**−2.57**	[−2.58, −2.56]	**−2.58**	[−2.59, −2.57]	**−2.57**	[−2.59, −2.56]
	0.900(00%)	0.00	[0.00, 0.00]	0.00	[0.00, 0.00]	0.00	[0.00, 0.00]
d198	0.100(00%)	**−0.82**	[−0.85, −0.79]	**−0.82**	[−0.84, −0.81]	**−0.85**	[−0.86, −0.84]
	0.200(00%)	**−0.97**	[−1.01, −0.94]	**−1.03**	[−1.06, −1.01]	**−1.00**	[−1.05, −0.95]
	0.500(00%)	**−0.83**	[−0.86, −0.79]	**−0.85**	[−0.87, −0.82]	**−0.84**	[−0.86, −0.82]
	0.900(00%)	−0.02	[−0.05, +0.01]	**−0.07**	[−0.09, −0.04]	**−0.03**	[−0.05, −0.01]
rat783	0.100(00%)	**−10.14**	[−10.24, −10.05]	**−10.23**	[−10.32, −10.15]	**−10.20**	[−10.26, −10.14]
	0.200(00%)	**−4.60**	[−4.77, −4.42]	**−4.64**	[−4.75, −4.54]	**−4.31**	[−4.57, −4.05]
	0.500(00%)	**−3.35**	[−3.47, −3.23]	**−3.51**	[−3.61, −3.42]	**−3.35**	[−3.53, −3.17]
	0.900(00%)	**−0.32**	[−0.38, −0.26]	**−0.32**	[−0.52, −0.12]	**−0.49**	[−0.62, −0.36]

6.2.5 TSP approximation and the aggregation approach in estimation-based algorithms

A very different approach to the PTSP is to completely forget about its stochastic component and to tackle a PTSP instance as if it were a TSP instance. This approach makes it possible to exploit the state-of-the-art in TSP solving for generating *a priori* solutions for the PTSP. In fact, this approach is, according to the PTSP literature surprisingly effective: Bianchi (2006) and Bianchi and Gambardella (2007) benchmarked pACS+1-shift against the state-of-the-art exact TSP solver, Concorde (Applegate et al., 2001) and they established a critical mean probability of 0.5, above which the latter is more effective than the former. Note that Concorde is an exact algorithm that finds the optimal solution for a given TSP instance. In this section, we repeat the same experiments but now using ILS-EE, ACS-EE, and MAGX-EE as the PTSP solver (using the version that is tuned for 1000 CPU seconds). We use the same instances and a $n^2/100$ CPU seconds stopping criterion as described by Bianchi (2006), Bianchi and Gambardella (2007).

The results on PTSP instances derived from the TSP instances `kroA100`, `d198`, and `rat783` are shown in Table 6.8. The estimation-based algorithms ILS-EE, ACS-EE, and MAGX-EE obtain solution costs that are significantly lower than those of Concorde up to probability 0.9. The advantage of the estimation-based algorithms is quite strong on instances with low probability values and the observed average difference increases with instance size reaching more than 10% for the largest instance `rat783`.

We also investigated whether the effectiveness of estimation-based algorithms can be improved by using optimal tours for the TSP or the tours returned by the aggregation approach as initial solutions under short computation time. However, we could not observe a significant improvement in obtained solution cost with the two approaches. We refer the reader to Balaprakash et al. (2009b) for complete results.

6.3 Summary

In this chapter, we customized iterated local search, memetic, and ant colony optimization algorithms to tackle the PTSP. The proposed customization consists in adopting an estimation-based approach to evaluate the solution cost and the state-of-the-art iterative improvement algorithm, `2.5-opt-EEais`, as local search. We presented an experimental comparison of the estimation-based algorithms using short and long computation times as stopping criteria. First, we performed a rigorous parameter tuning

Table 6.8: Comparison of the average cost obtained by ILS-EE, MAGX-EE, ACS-EE, and Concorde over 10 independent runs on instance `kroA100`, `d198`, and `rat783`. Typographic conventions are the same as in Table 6.2.

		ILS-EE vs. Concorde		MAGX-EE vs. Concorde		ACS-EE vs. Concorde	
	p	d	[95% CI]	d	[95% CI]	d	[95% CI]
kroA100	0.100(00%)	**−0.41**	[−0.44, −0.38]	**−0.43**	[−0.45, −0.40]	**−0.45**	[−0.47, −0.43]
	0.200(00%)	**−0.41**	[−0.43, −0.38]	**−0.41**	[−0.43, −0.40]	**−0.29**	[−0.44, −0.14]
	0.300(00%)	**−0.22**	[−0.26, −0.18]	**−0.27**	[−0.30, −0.24]	**−0.29**	[−0.32, −0.26]
	0.400(00%)	**−0.08**	[−0.15, −0.01]	**−0.16**	[−0.16, −0.15]	**−0.16**	[−0.16, −0.15]
	0.500(00%)	−0.05	[−0.12, +0.02]	**−0.09**	[−0.09, −0.09]	**−0.09**	[−0.09, −0.09]
	0.600(00%)	**−0.07**	[−0.08, −0.06]	**−0.07**	[−0.08, −0.07]	**−0.07**	[−0.08, −0.07]
	0.700(00%)	**−0.16**	[−0.17, −0.14]	**−0.16**	[+0.00, +0.00]	**−0.16**	[+0.00, +0.00]
	0.800(00%)	**−0.03**	[−0.04, −0.02]	**−0.04**	[−0.05, −0.03]	**−0.03**	[−0.05, −0.02]
	0.900(00%)	**−0.00**	[+0.00, +0.00]	**−0.00**	[+0.00, +0.00]	−0.00	[−0.01, +0.00]
	1.000(00%)	0.00	[0.00, 0.00]	0.00	[0.00, 0.00]	0.00	[0.00, 0.00]
d198	0.100(00%)	**−1.08**	[−1.11, −1.05]	**−1.09**	[−1.10, −1.07]	**−1.12**	[−1.13, −1.11]
	0.200(00%)	**−1.22**	[−1.26, −1.19]	**−1.28**	[−1.31, −1.26]	**−1.25**	[−1.30, −1.21]
	0.300(00%)	**−1.44**	[−1.47, −1.40]	**−1.48**	[−1.51, −1.45]	**−1.49**	[−1.53, −1.44]
	0.400(00%)	**−1.25**	[−1.28, −1.21]	**−1.28**	[−1.30, −1.25]	**−1.32**	[−1.33, −1.31]
	0.500(00%)	**−0.92**	[−0.95, −0.88]	**−0.94**	[−0.96, −0.91]	**−0.93**	[−0.95, −0.91]
	0.600(00%)	**−0.51**	[−0.57, −0.44]	**−0.54**	[−0.59, −0.50]	**−0.55**	[−0.59, −0.50]
	0.700(00%)	**−0.41**	[−0.45, −0.38]	**−0.47**	[−0.47, −0.47]	**−0.47**	[−0.47, −0.47]
	0.800(00%)	**−0.22**	[−0.27, −0.18]	**−0.27**	[−0.27, −0.27]	**−0.27**	[−0.27, −0.27]
	0.900(00%)	**−0.08**	[−0.11, −0.05]	**−0.12**	[−0.14, −0.10]	**−0.09**	[−0.11, −0.06]
	1.000(00%)	*+0.05*	[+0.03, +0.07]	*+0.03*	[+0.01, +0.05]	+0.02	[−0.00, +0.04]
rat783	0.100(00%)	**−10.18**	[−10.27, −10.09]	**−10.27**	[−10.36, −10.19]	**−10.24**	[−10.29, −10.18]
	0.200(00%)	**−8.39**	[−8.57, −8.22]	**−8.44**	[−8.54, −8.34]	**−8.12**	[−8.37, −7.87]
	0.300(00%)	**−6.25**	[−6.34, −6.15]	**−6.32**	[−6.46, −6.17]	**−6.43**	[−6.59, −6.28]
	0.400(00%)	**−4.72**	[−4.85, −4.58]	**−4.64**	[−4.78, −4.49]	**−4.79**	[−4.90, −4.67]
	0.500(00%)	**−3.39**	[−3.51, −3.27]	**−3.56**	[−3.65, −3.46]	**−3.39**	[−3.57, −3.21]
	0.600(00%)	**−2.10**	[−2.21, −1.99]	**−2.26**	[−2.38, −2.13]	**−2.28**	[−2.45, −2.12]
	0.700(00%)	**−1.17**	[−1.32, −1.03]	**−1.45**	[−1.59, −1.32]	**−1.39**	[−1.47, −1.31]
	0.800(00%)	**−0.81**	[−0.90, −0.72]	**−0.82**	[−0.93, −0.71]	**−0.92**	[−1.08, −0.75]
	0.900(00%)	**−0.56**	[−0.62, −0.50]	**−0.56**	[−0.76, −0.37]	**−0.73**	[−0.86, −0.61]
	1.000(00%)	*+0.81*	[+0.71, +0.91]	*+0.76*	[+0.61, +0.91]	*+0.49*	[+0.34, +0.63]

of all the estimation-based algorithms to avoid any bias due to the tuning procedure. Using the fine tuned parameter values, we evaluated the solution cost obtained by the estimation-based algorithms on three instance classes. For the short computation time, the iterated local search is most effective. For long computation time, both the iterated local search and the memetic algorithms outperform ant colony system on instances with low average probability values (up to 0.2). Nevertheless, ant colony system emerges as the best algorithm on instances with average probability values between 0.3 and 0.5.

The main contribution of the chapter is the development of new state-of-the-art metaheuristic algorithms for the PTSP. Compared to the previously proposed analytical computation algorithms, the estimation-based metaheuristic algorithms obtain high quality solutions in a very short computation time. The advantage in speed and solution quality is primarily due to the adoption of `2.5-opt-EEais` as the local search. Moreover, in the literature, it has been shown that the PTSP instances with average probability value greater than 0.5 can be solved more effectively by an exact TSP algorithm than the analytical computation PTSP algorithm. Here, we showed that the estimation-based algorithms can push this probability limit to much larger values up to 0.9. In a nutshell, we showed that the estimation-based approach is an effective replacement for analytical computation in metaheuristics for the PTSP.

Chapter 7

Estimation-based Metaheuristics for the VRPSDC

In this chapter, we customize the estimation-based metaheuristics developed for the PTSP to tackle the vehicle routing problem with stochastic demands and customers (VRPSDC). The VRPSDC is an extension of the PTSP: a capacitated vehicle has to serve a set of customers where each customer has a probability of requiring being visited and a stochastic demand. Besides the usual difficulty of finding an optimal solution in a large search space, the VRPSDC has a computationally expensive cost function. For large instances, the adoption of this cost function in metaheuristics is a primary bottleneck for finding high quality solutions in a reasonable computation time. We tackle this issue by using an empirical estimation approach to evaluate the VRPSDC solution cost. We also exploit the effectiveness of `2.5-opt-EEais` for the VRPSDC by using it as an underlying local search algorithm inside metaheuristics. We present an experimental study of several estimation-based metaheuristics on a number of problem instances. We show that the proposed estimation-based algorithms outperform the current best algorithm tailored to solve the given problem. Among the estimation-based algorithms, we show that an ant colony optimization algorithm is particularly effective.

This chapter is organized as follows. In Section 7.1, we describe the VRPSDC and its solutions approaches. In Section 7.2, we describe estimation-based metaheuristics for the VRPSDC. In Section 7.3, we present an experimental study of the proposed algorithms to show their effectiveness. We summarize the results in Section 7.4.

7.1 The VRPSDC

The VRPSDC (Bertsimas, 1988) is concerned with minimizing the cost involved in routing a capacitated vehicle that collects goods from a set of customers. Each customer has a probability of requiring being visited and a stochastic demand. This problem combines two classical stochastic routing problems: the probabilistic traveling salesman problem (PTSP) and the well-known vehicle routing problem with stochastic demands (VRPSD). The VRPSDC is an $\mathcal{NP}$-hard problem that models a number of practical problems in the areas of less-than-truckload operations and package delivery systems (Bertsimas, 1988, 1992).

Formally, an instance of the VRPSDC can be defined on a graph G with the following elements:

- a set $V = \{1, 2, ..., n\}$ of nodes that represent customers with node 1 being the depot;
- a set $A = \{\langle i, j\rangle : i, j \in V, i \neq j\}$ of edges, where an edge $\langle i, j\rangle$ connects nodes i and j;
- a set $C = \{c_{ij} : \langle i, j\rangle \in A\}$ of travel costs, where c_{ij} is the cost of using edge $\langle i, j\rangle \in A$;
- a set $P = \{p_i : i \in V, i \neq 1, 0 \leq p_i \leq 1\}$ of probabilities, where p_i is the probability that a node i requires being visited;
- a set $\xi = \{\xi_i : i \in V, i \neq 1\}$ of random variables, where ξ_i describes the stochastic demand of node i;
- a vehicle of capacity Q.

It is assumed that (i) the travel costs are symmetric, that is, for all pairs of nodes i, j we have $c_{ij} = c_{ji}$; (ii) the events that two distinct nodes i and j require being visited and their demands are assumed to be independent; (iii) node demands are discrete; (iv) the information that a node i does not require being visited is revealed before the vehicle leaves the predecessor of node i; (v) the exact demand of a node i is revealed only when the vehicle reaches node i; and (vi) the maximum demand of any node does not exceed the vehicle capacity Q.

The VRPSDC is usually tackled using *a priori* optimization, which consists in finding an *a priori* solution that minimizes the expected cost of the *a posteriori* solution. The *a priori* solution is a Hamiltonian tour, which is found before knowing which nodes

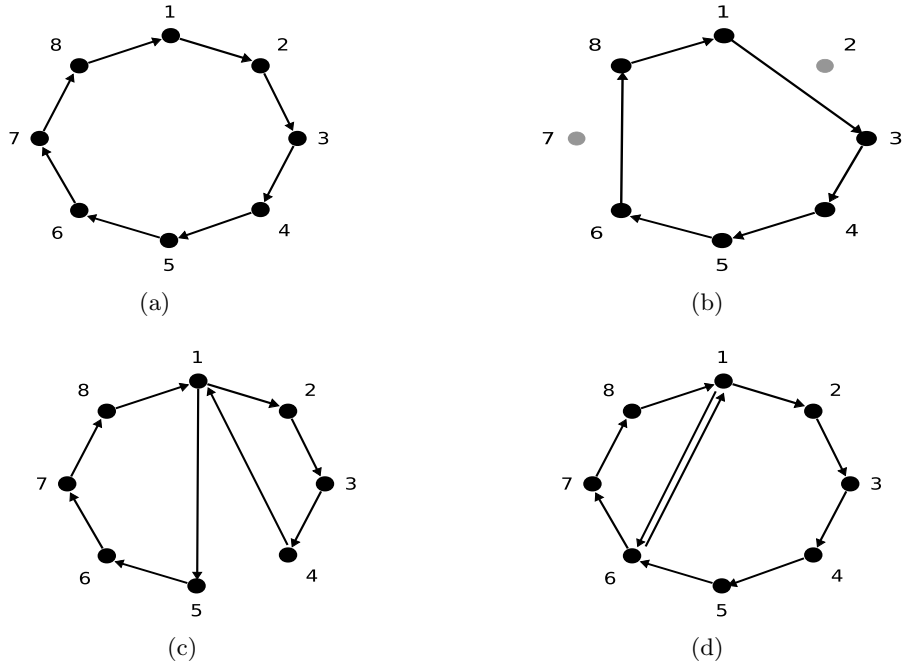


Figure 7.1: Plot 7.1(a) shows an *a priori* solution for a VRPSDC instance with 8 nodes, where node 1 is the depot and the nodes are visited in the following order: 1, 2, 3, 4, 5, 6, 7, 8, and 1. Plot 7.1(b) shows a recourse action for skipping nodes 2 and 7 that do not require being visited assuming that the vehicle capacity is not exceeded and recourse action due to the vehicle capacity is not performed. Plot 7.1(c) shows a recourse action at node 4, where the capacity of the vehicle is exactly attained after collecting goods. The vehicle goes back to the depot 1 and it resumes the collection from the next node 5 that requires being visited. Plot 7.1(d) shows a recourse action at node 6, where the space available in the vehicle is not enough to collect all goods. The vehicle accommodates a part of the goods up to its capacity, returns to the depot 1 to unload, and goes back to the same node 6 to collect the remaining goods.

require being visited and their respective demands; the *a posteriori* solution is obtained by following the nodes in the order of the *a priori* solution with the following recourse actions.

- Similar to the PTSP, nodes that do not require being visited are skipped.
- If the capacity of the vehicle is exactly attained after collecting goods at a node, the vehicle goes back to the depot and resumes the collection from the next node that requires being visited.
- If the space available in the vehicle is not enough to collect all goods from a node, the vehicle accommodates a part of the goods up to its capacity, returns to the depot to unload, and goes back to the same node to collect the remaining goods.

See Figure 7.1 for an illustration of the three recourse actions.

Bertsimas (1988), Séguin (1994), and Gendreau et al. (1995) derived closed-form expressions to compute the exact cost of a VRPSDC solution. Let $x = (\pi(1), \pi(2), \ldots, \pi(n), \pi(n+1) = \pi(1))$ be an *a priori* solution for the VRPSDC, where π is a permutation of the set V. The cost $F(x)$ of the *a priori* solution x can be obtained as follows:

$$F(x) = \sum_{i=1}^{n} \sum_{j=i+1}^{n+1} c_{\pi(i)\pi(j)} \overline{p}_{\pi(i)\pi(j)} + \gamma_2\big(Q\big), \tag{7.1}$$

where

$$\overline{p}_{\pi(i)\pi(j)} = \begin{cases} p_{\pi(i)} p_{\pi(j)}, & j = i+1, \\ p_{\pi(i)} p_{\pi(j)} \prod_{h=i+1}^{j-1} (1 - p_{\pi(h)}), & j > i+1, \end{cases}$$

$$\gamma_n\big(g\big) = p_{\pi(n)}(c_{\pi(n)\pi(1)} + c_{\pi(1)\pi(n)}) \sum_{l|\xi^l_{\pi(n)} > g} p^l_{\pi(n)}$$

$$(1 \le g \le Q).$$

$$\begin{aligned} \gamma_i\big(g\big) &= (1 - p_{\pi(i)})\gamma_{i+1}(g) \\ &\quad + p_{\pi(i)}\Big[\sum_{l|\xi_{\pi(i)} > g} p^l_{\pi(i)} (\gamma_{i+1}(Q - \xi^l_{\pi(i)}) + c_{\pi(i)\pi(1)} + c_{\pi(1)\pi(i)}) \\ &\quad + \sum_{l|\xi_{\pi(i)} < g} p^l_{\pi(i)} \gamma_{i+1}(g - \xi^l_{\pi(i)}) \Big] \\ &\quad + P(\xi_{\pi(i)} = g | \pi(i) \text{ is present}) \\ &\quad \Big[p_{\pi(i)} \gamma_{i+1}(Q) + \sum_{j=i+1}^{n} \overline{p}_{\pi(i)\pi(j)} (c_{\pi(i)\pi(1)} + c_{\pi(1)\pi(j)} - c_{\pi(i)\pi(j)}) \Big] \end{aligned}$$

$$(i = 2, \ldots, n-1; 1 \leq g \leq Q),$$

where $p^l_{\pi(i)}$ is the probability of the l^{th} discrete demand level $\xi^l_{\pi(i)}$ of $\xi_{\pi(i)}$. The time complexity of this expression is $O(n^2 + n\bar{l}Q)$, where $\bar{l}$ is the maximum number of different demand levels for any node (Gendreau et al., 1995). It should be noted that in Equation 7.1, an undirected solution must be evaluated twice—once in the clockwise direction and once in the anti-clockwise direction. This is due to the fact that a vehicle with a finite capacity Q might have different recourse actions depending on whether it visits the nodes in the clockwise or the anti-clockwise direction.

The VRPSDC has received less attention than other stochastic routing problems and, in particular, than the PTSP. Séguin (1994) and Gendreau et al. (1995) first proposed an exact algorithm based on the Integer L-shaped method. This algorithm is tested on instances with up to 70 nodes. The authors showed that instances with a large number of probabilistic nodes cannot be solved in a reasonable computation time. For example, to solve an instance with 10 probabilistic nodes and 1 depot, the Integer L-shaped method needed up to 53903 CPU seconds on a Silicon Graphics computer with a 33 Mhz processor.

Séguin (1994) and Gendreau et al. (1996b) tackled this problem using a tabu search algorithm, TABUSTOCH. This algorithm uses a randomized node-insertion neighborhood, where the neighborhood of the incumbent solution is a set of solutions obtained by deleting q nodes and inserting each of them before or after its nn nearest neighbors, where q and nn are parameters. Concerning the tabu length, if the incumbent solution at iteration v is obtained by deleting a node i between two nodes, its reinsertion between the same two nodes is tabu until iteration $v + \theta$, where θ is a random value in the interval $[n-5, n]$. At each iteration, the algorithm explores all neighboring solutions of the incumbent solution. Since computing the exact cost of all neighboring solutions with Equation 7.1 is computationally expensive, the cost differences between neighboring solutions are computed using a delta evaluation that is based on an analytical approximation approach. A list is constructed in which the neighboring solutions are sorted in ascending order with respect to the approximate costs obtained from the proxy expression. Using this sorted list, the exact cost of solutions are computed with Equation 7.1 in a batch-by-batch fashion, where a batch comprises five solutions. As soon as the algorithm finds a non-tabu neighboring solution that has a lower cost than the incumbent solution, the exact cost computation is stopped and the lower cost neighboring solution is accepted as the incumbent solution. However, if the algorithm finds a lower cost neighboring solution in the first batch of the sorted list, then the solution

is accepted immediately even if it is obtained by a tabu move. Note that this algorithm can also handle multiple vehicles. For a complete description of the algorithm, we refer the reader to Gendreau et al. (1996b). To the best of our knowledge, TABUSTOCH is the only metaheuristic explicitly designed to tackle the VRPSDC.

The main novelty in TABUSTOCH is the adoption of the analytical approximation approach in delta evaluation. Gendreau et al. (1996b) systematically tested four analytical approximation schemes: a TSP delta evaluation scheme that considers edge costs but completely ignores node probabilities and demands; a PTSP delta evaluation that adopts edge costs weighted by node probabilities; two VRPSDC delta evaluation schemes that consider edge costs, node probabilities, and demands. After some preliminary experiments, the authors adopted the PTSP delta evaluation scheme, which they found to be more effective than the TSP and VRPSDC schemes.

It is worthwhile to focus on some implementation details of TABUSTOCH. Séguin (1994) and Gendreau et al. (1996b) developed TABUSTOCH to tackle the VRPSDC with K vehicles, where K is a parameter. A penalized cost function is used to force the algorithm to achieve K *a priori* routes. The approximation scheme is more sophisticated when a node from one *a priori* route is moved to another *a priori* route during randomized node-insertion moves. In the experimental analysis, TABUSTOCH is applied to solve the VRPSDC instances of size up to 46 with 2 vehicles. On the instance of size 11 with 10 probabilistic nodes, where the Integer L-shaped method needed up to 53903 CPU seconds, TABUSTOCH required only 15.46 CPU seconds to find the optimal solution on the same hardware. In this chapter, we focus on a single vehicle VRPSDC because the adoption of multiple vehicles might not allow us to exactly assess the computational overhead involved in using Equation 7.1. This is ascribed to the fact that, for a given instance with n nodes, the depth of recursion and the associated function-call overhead while using two vehicles will be much more than that of one vehicle—in particular for large instances.

There are two main shortcomings in the literature that motivated us to develop estimation-based algorithms for the VRPSDC. First, the currently available metaheuristic, TABUSTOCH, uses a computationally expensive analytical computation approach, which affects the performance of the algorithm for large instances. The effectiveness of using the alternative empirical estimation approach has never been studied within metaheuristics to tackle the VRPSDC. Second, there is only one tailor-made metaheuristic for the VRPSDC; the possibility of using other metaheuristics to tackle the VRPSDC has not been studied.

7.2 Estimation-based metaheuristics

As in the PTSP, the empirical estimation approach consists in estimating the cost $F(x)$ on the basis of sample costs $f(x, \omega_1), f(x, \omega_2), \ldots, f(x, \omega_M)$ of *a posteriori* solutions obtained from M independent realizations $\omega_1, \omega_2, \ldots, \omega_M$ of probabilities and stochastic demands of nodes. A realization comprises two components for each node i: the first component takes a value `'1'` with a probability p_i and a value `'0'` with a probability $1 - p_i$; the second component is a demand value sampled from the demand distribution ξ_i when the first component is `'1'`. The time complexity involved in evaluating the cost of a solution by the empirical estimation is $O(nM)$. Unlike Equation 7.1, the time complexity of this approach does not increase with the vehicle capacity, Q or with the maximum number of different demand levels for any node, $\overline{l}$. However, as discussed in previous chapters, the number M of realizations is crucial to the effectiveness of this approach.

We extend the estimation-based PTSP metaheuristics to tackle the VRPSDC. This is done in three steps.

- The VRPSDC-specific realizations are generated and used for cost estimation.
- The cost of an *a posteriori* solution is computed by considering the VRPSDC-specific recourse actions.
- As in TABUSTOCH, we consider solutions obtained by estimation-based metaheuristics to be undirected. Thus, given a number of solutions that need to be compared, each solution needs to be evaluated twice, once in clockwise direction and once in anti-clockwise direction. Consequently, in random restart local search and iterated local search, there are four solutions that need to be compared in each iteration. Hence, in all algorithms, at each iteration, we use ANOVA-Race with VRPSDC-specific realizations to decide which solution is better.

Concerning `2.5-opt-EEais`, a straightforward extension of the PTSP delta evaluation to the VRPSDC delta evaluation is not feasible because local modifications in a solution entail a global change in the cost of a solution. Therefore, we use `2.5-opt--EEais` as developed for the PTSP without any modification as a local search algorithm in all metaheuristics. Thus, in the local search phase, a PTSP approximation is used for the VRPSDC by ignoring the nodes demand and the vehicle capacity. This particular usage is aimed to exploit the speed advantage of `2.5-opt-EEais` during the local search phase. We expect that `2.5-opt-EEais` will be effective for the VRPSDC

because it has been shown that the stochasticity due to probabilistic nodes has more influence than the one due to the stochastic demands for solving the VRPSDC and that the PTSP approximation in TABUSTOCH is more effective than the problem-specific approximation (Séguin, 1994; Gendreau et al., 1995).

7.3 Experimental analysis

In this section, we present the experimental setting and the results from the empirical study. The goal of the experiments is to assess the effectiveness of the estimation-based metaheuristics on large instances.

7.3.1 Experimental setup

The VRPSDC instances used for the experiments are obtained as follows. First TSP instances are generated with the DIMACS instance generator (Johnson et al., 2001), where the nodes are arranged as a number of clusters in a $10^6 \times 10^6$ square. Four levels of instance size are considered: 30, 100, 300, and 1000. For each TSP instance, a same probability value p is assigned to all nodes except the first node, which is the depot. The considered values for p are from 0.050 to 0.200 with an increment of 0.025 and 0.3, 0.5, 0.8, and 1.0. The demand distributions for the nodes and the vehicle capacity are then assigned as described in Gendreau et al. (1996a): each node takes at random one of the three discrete uniform distributions in $\{1, \ldots, 9\}$, $\{5, \ldots, 15\}$, $\{10, \ldots, 20\}$. In this way, for a given instance, the expected demand of each node that requires being visited is equal to 10. The vehicle capacity Q is set to $10 \times \sum_{i=2}^{n}(p/fc)$, where fc is the so-called *filling coefficient*. We consider values $fc \in \{1.0, 2.0, 4.0, 8.0\}$. Note that the values of fc affect Q in the following way: when fc is set to 1.0, Q is equal to the total expected demand of the nodes; when fc is set to 8.0, Q is equal to the one eighth of the total expected demand of the nodes. The generated instances are grouped into three classes according to p: $\{0.050, 0.075, 0.100\}$ (Class I), $\{0.150, 0.175, 0.200\}$ (Class II), $\{0.300, 0.500, 0.800, 1.000\}$ (Class III).

The aforementioned instance generation scheme differs from the one described in Gendreau et al. (1996a) in the following way: in Gendreau et al. (1996a), each instance contained 1, $n - 1/2$, or $n - 1$ probabilistic nodes and the probability values of the nodes are randomly generated between 0 and 1. Although the authors showed that instances with $n - 1$ probabilistic nodes are difficult to solve, the impact of the values of node probabilities on the effectiveness of the algorithms has not been investigated. To address this issue, we use instances with homogeneous node probability. The values

of *fc* chosen by Gendreau et al. (1996a) are 0.25, 0.50, 0.75. The adoption of these values in our setting leads to a vehicle capacity which is always greater than the total expected nodes demands. In this case, recourse actions, in which the vehicle goes back to the depot, are unlikely to occur.

For the experimental analysis, we implemented TABUSTOCH as described in Gendreau et al. (1996a) but applied to VRPSDC instances with only one vehicle: we did not use the penalized cost function. In the approximation scheme, we did not use the auxiliary computations needed when a node from one *a priori* route is moved to another *a priori* route during randomized node-insertion moves. We use the same parameter values as suggested in Gendreau et al. (1996a).

The nearest-neighbor heuristic is used to generate the initial solution in TABUSTOCH, RRLS-EE, ILS-EE, and MAGX-EE. The minimum number of realizations used in the ANOVA-Race before applying ANOVA is set to five; the null hypothesis is rejected at a significance level of 0.05. All algorithms use a same set of M realizations for all iterations, however, realizations are selected randomly from this set for each iteration. The maximum number M of realizations is set to one thousand in all algorithms. The critical values of the F-distribution and Tukey HSD test are pre-computed and stored as a look up table. Equation 7.1 is used for the post-evaluation of the best-so-far solutions found by all algorithms. For `2.5-opt-EEais`, we use the same parameter values obtained for the PTSP–see Table 6.1 on page 105.

We present the exemplary results obtained from several instances. The trend of the results obtained on all other instances is similar. The complete results and the numerical values are given in an online supplementary document (Balaprakash et al., 2009a):

`http://iridia.ulb.ac.be/supp/IridiaSupp2009-008/`

7.3.2 Effectiveness of `2.5-opt-EEais`

In this section, we show that a simple random restart local search that uses `2.5-opt-EEais` is more effective than TABUSTOCH. Since RRLS-EE uses `2.5-opt-EEais` and ANOVA-Race, it will be difficult for us to attribute the effectiveness only to the adoption of `2.5-opt-EEais`. Therefore, we use RRLS-AC, a random restart local search similar to RRLS-EE with the exception that local optima are compared using Equation 7.1. We evaluate the two algorithms on instances with 30, 100, and 300 nodes.

The stopping criterion for the comparison for each instance is chosen as follows: TABUSTOCH is run until it performs 1000 iterations and the time needed for completion is

Table 7.1: The computation time (in CPU seconds) needed by TABUSTOCH to complete 1000 iterations. For each level of p and fc combination, the mean and the standard deviations (s.d.) are computed over 10 instances of size 30, 100, and 300.

	$n = 30$		$n = 100$		$n = 300$	
$p(fc)$	mean	s.d.	mean	s.d.	mean	s.d.
0.100(1.00)	38	1	502	10	5457	114
0.100(2.00)	40	1	509	9	5527	124
0.100(4.00)	40	1	522	13	5655	119
0.100(8.00)	40	1	526	13	5727	99
0.200(1.00)	37	1	487	11	5268	130
0.200(2.00)	38	1	484	9	5279	113
0.200(4.00)	39	1	499	11	5289	91
0.200(8.00)	40	1	521	8	5347	149
0.300(1.00)	37	0	473	8	4999	91
0.300(2.00)	37	0	476	8	5055	115
0.300(4.00)	39	0	483	11	5134	128
0.300(8.00)	40	1	494	11	5035	135
0.500(1.00)	36	0	464	10	4842	78
0.500(2.00)	37	1	465	4	4831	82
0.500(4.00)	38	1	467	6	4895	150
0.500(8.00)	40	0	477	8	4917	122
0.800(1.00)	36	1	457	6	4738	55
0.800(2.00)	36	1	460	6	4725	51
0.800(4.00)	37	0	463	4	4748	68
0.800(8.00)	38	0	466	5	4781	78
1.000(1.00)	36	1	456	5	4638	60
1.000(2.00)	36	1	458	6	4688	53
1.000(4.00)	37	1	460	4	4691	77
1.000(8.00)	37	0	464	5	4749	78

recorded. The time limit for RRLS-AC is then set to the time taken by TABUSTOCH. Table 7.1 shows the average computation time needed by TABUSTOCH to complete 1000 iterations. From the results, we can observe that the instance size has a strong impact on the computation time of TABUSTOCH. In particular, the computation time increases quite drastically (more than 2 orders of magnitude) by increasing the instance size from 30 to 300. However, there is no considerable difference in computation time for different values of fc. The observed trends are ascribed to the fact that the term n^2 in $O(n^2 + n\bar{l}Q)$ time complexity of Equation 7.1 clearly dominates over other terms.

Figure 7.2 shows exemplary run time development plots on instances of size 100. From these plots, we can observe that RRLS-AC completely outperforms TABUSTOCH. The poor performance of TABUSTOCH is due to the heavy usage of Equation 7.1: the algorithm moves from a solution x to a lower cost neighbor solution, where each move is obtained at an expense of high computation time because *at least* five neighbor solutions are evaluated with Equation 7.1. We can also observe from the plots that the improvement in TABUSTOCH by moving from a incumbent solution

to a neighbor solution is rather small. This is typical when an algorithm moves in the space of neighbor solutions. However, it should be noted that, for a fixed value of *fc*, the relative difference in the average solution cost between TABUSTOCH and RRLS-AC increases with an increase of p. This is due to the adoption of randomized node-insertion neighborhood, which becomes less effective for large values of p. Note that we observed a similar behavior in PTSP iterative improvement algorithms.

The high performance of RRLS-AC is due to the adoption of `2.5-opt-EEais`. This can be inferred from Figure 7.2, which shows that for RRLS-AC there is a large improvement within a short computation time (less than 5 to 6 seconds). The traces of RRLS-AC show that this large improvement is achieved in the very first iteration after the first run of `2.5-opt-EEais` on the initial solution.

For a given value of p, the relative difference in the average solution cost between the two algorithms decreases with an increase in the value of *fc*. This can be explained as follows. The number of recourse actions in which the vehicle has to return back to the depot for unloading increases with an increase in the value of *fc*. Since this recourse action is ignored by `2.5-opt-EEais`, the quality of the local optima found by `2.5-opt-EEais` decreases with an increase of *fc*.

Table 7.2 reports the observed relative difference between the final solution cost achieved by the two algorithms with a 95% confidence bound obtained through a t-test. For the absolute values, we refer the reader to Balaprakash et al. (2009a). The results show a general trend that RRLS-AC is better than TABUSTOCH across a wide range of instance sizes, node probabilities, and the vehicle capacity. For the instance size 300, the average cost of the solutions obtained by RRLS-AC is between 15.92% and 1.78% less than that of TABUSTOCH. The observed differences are significant according to the t-test. This trend is similar for instances of size 100, where RRLS-AC achieves an average solution cost that is between 15.04% to 1.85% less than that of TABUSTOCH. For the smallest instance, $n = 30$, the observed difference in the solution cost is between 12.63% to 1.93%. There are few exceptions where TABUSTOCH obtains solutions whose cost is slightly lower than that of RRLS-AC.

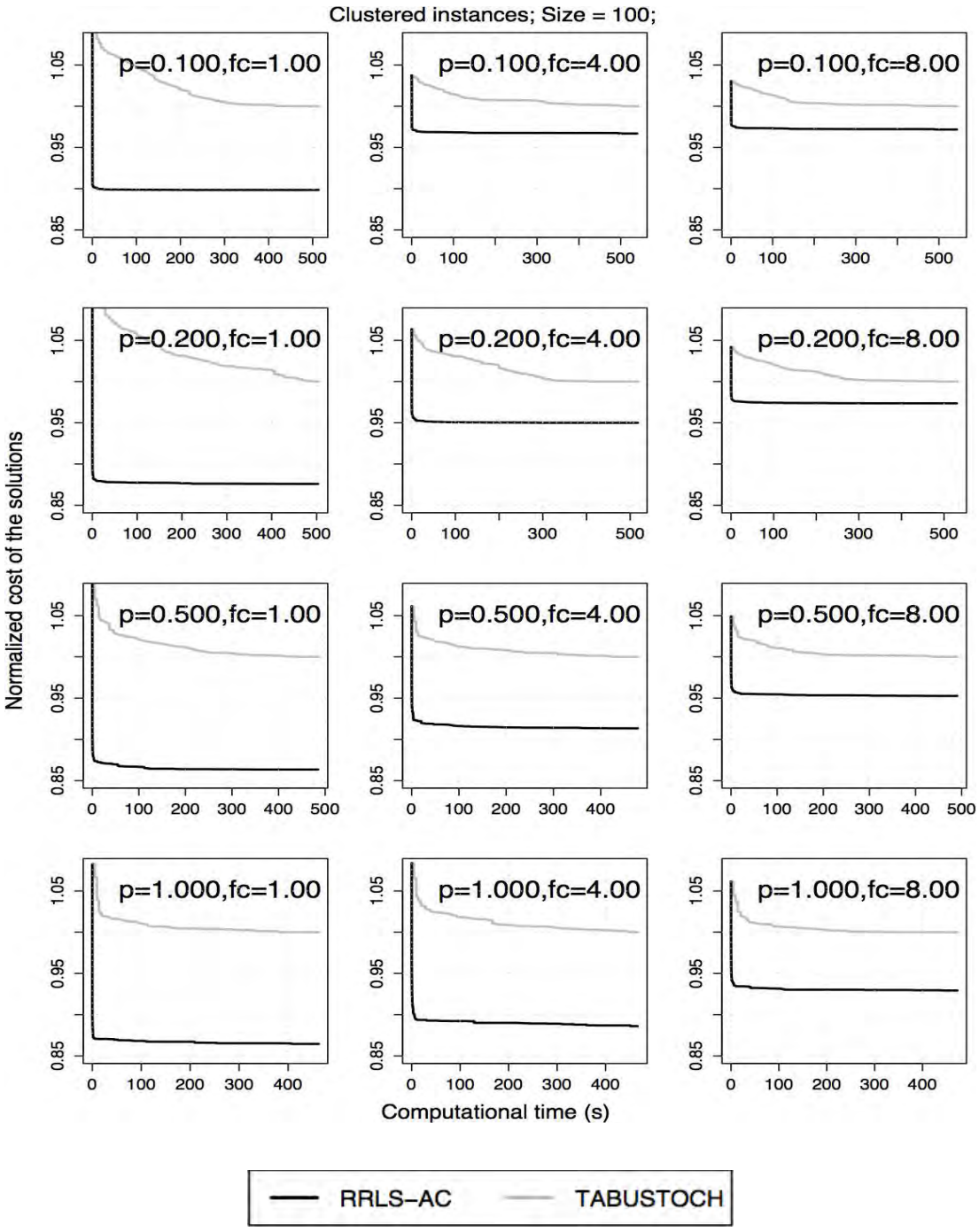


Figure 7.2: Experimental results on the clustered VRPSDC instances of size 100. The plots represent the cost of the solutions obtained by RRLS-AC normalized by those obtained by TABUSTOCH. The normalization is done on an instance-by-instance basis for 10 instances; the normalized solution cost is then aggregated.

Table 7.2: Comparison of the average cost obtained by RRLS-AC and TABUSTOCH over 10 instances of size 30, 100, and 300. See Footnote 1 for an explanation of the contents and the typographic conventions adopted in the table.

RRLS-AC vs. TABUSTOCH

	$n = 30$		$n = 100$		$n = 300$	
p	Difference	95% CI	Difference	95% CI	Difference	95% CI
0.100(1.00)	+0.04	[−1.53, +1.62]	**−10.32**	[−15.37, −5.27]	**−12.58**	[−14.37, −10.78]
0.100(2.00)	*+1.85*	[+0.58, +3.13]	**−6.14**	[−8.84, −3.43]	**−10.00**	[−12.88, −7.11]
0.100(4.00)	*+1.85*	[+0.58, +3.13]	**−3.56**	[−5.57, −1.55]	**−4.94**	[−6.66, −3.22]
0.100(8.00)	*+1.85*	[+0.58, +3.13]	**−2.86**	[−4.61, −1.11]	**−3.40**	[−4.39, −2.41]
0.200(1.00)	**−2.94**	[−5.34, −0.53]	**−12.67**	[−18.44, −6.91]	**−14.32**	[−17.21, −11.44]
0.200(2.00)	−0.78	[−2.24, +0.67]	**−10.84**	[−14.17, −7.51]	**−13.76**	[−17.23, −10.29]
0.200(4.00)	−0.21	[−2.38, +1.97]	**−5.30**	[−7.28, −3.32]	**−8.36**	[−9.44, −7.28]
0.200(8.00)	−0.21	[−2.38, +1.97]	**−2.80**	[−3.84, −1.77]	**−5.36**	[−6.35, −4.36]
0.300(1.00)	**−5.78**	[−10.31, −1.24]	**−14.89**	[−18.73, −11.05]	**−15.76**	[−18.48, −13.05]
0.300(2.00)	**−4.57**	[−7.29, −1.85]	**−11.50**	[−13.03, −9.97]	**−14.39**	[−17.82, −10.95]
0.300(4.00)	**−1.94**	[−3.68, −0.20]	**−7.36**	[−8.94, −5.77]	**−9.86**	[−11.39, −8.33]
0.300(8.00)	**−2.08**	[−3.46, −0.70]	**−4.68**	[−5.83, −3.53]	**−6.74**	[−7.53, −5.94]
0.500(1.00)	**−8.02**	[−13.34, −2.70]	**−13.90**	[−17.28, −10.51]	**−14.69**	[−18.57, −10.81]
0.500(2.00)	**−8.52**	[−12.17, −4.87]	**−13.14**	[−16.05, −10.23]	**−16.27**	[−20.41, −12.14]
0.500(4.00)	**−4.97**	[−6.99, −2.95]	**−8.67**	[−10.45, −6.90]	**−10.09**	[−11.53, −8.65]
0.500(8.00)	**−2.63**	[−3.70, −1.56]	**−5.07**	[−6.55, −3.59]	**−6.55**	[−7.76, −5.34]
0.800(1.00)	**−12.46**	[−18.41, −6.50]	**−13.74**	[−17.25, −10.23]	**−12.56**	[−15.64, −9.48]
0.800(2.00)	**−9.16**	[−14.19, −4.13]	**−13.30**	[−16.02, −10.57]	**−15.39**	[−18.91, −11.86]
0.800(4.00)	**−6.72**	[−9.60, −3.83]	**−8.96**	[−10.87, −7.05]	**−11.47**	[−12.93, −10.00]
0.800(8.00)	**−3.15**	[−4.58, −1.72]	**−5.95**	[−7.74, −4.15]	**−7.17**	[−8.51, −5.82]
1.000(1.00)	**−11.98**	[−16.36, −7.59]	**−14.08**	[−17.46, −10.70]	**−14.49**	[−17.41, −11.57]
1.000(2.00)	**−11.43**	[−17.35, −5.52]	**−14.06**	[−16.53, −11.58]	**−16.17**	[−18.72, −13.61]
1.000(4.00)	**−7.17**	[−10.55, −3.80]	**−11.41**	[−12.94, −9.89]	**−13.49**	[−16.33, −10.66]
1.000(8.00)	**−3.85**	[−5.38, −2.31]	**−6.99**	[−9.18, −4.81]	**−8.92**	[−11.06, −6.79]

[1] For a given comparison A vs. B, the table reports the observed relative difference d between the two algorithms A and B and the 95% confidence interval CI obtained through the t-test. If the value is positive, algorithm A obtained an average cost that is larger than the one obtained by algorithm B. In this case, the value is typeset in italics if it is significantly different from zero according to the t-test at a confidence level of 95%. If the value is negative, algorithm A obtained an average cost that is smaller than the one obtained by algorithm B. In this case, the value is typeset in boldface if it is significantly different from zero according to the t-test, at a confidence level of 95%.

7.3.3 Effectiveness of the estimation-based evaluation

In this section, we assess the effectiveness of the estimation-based evaluation procedure, ANOVA-Race, by comparing RRLS-EE to RRLS-AC. The two algorithms differ only with respect to the evaluation procedure. Given that the PTSP-specific `2.5-opt-EE-ais` obtained high quality solutions, we include in our analysis the RRLS-EE algorithm developed for the PTSP. We denote this algorithm RRLS-EE(PTSP). Note that RRLS-EE(PTSP) ignores node demands and thus uses an adaptive sample size procedure based on t-test to compare two local optima.

The three algorithms are compared on three levels of instance size: 100, 300, and 1000. For each level of instance size and for each p and fc combination, we use 10 instances for the comparison. Since the time limit used in Section 7.3.2 is rather high, we allow the three algorithms to run for n CPU seconds, where n is the size of the instance.

Figure 7.3 shows the run time development plots obtained on instances of size 1000. Note that in the plots the computation time is taken in log scale to highlight the speed of RRLS-EE. From the plots, we can observe that RRLS-EE obtains the average solution cost of RRLS-AC in approximately 0.5 to 2 orders of magnitude less CPU time. From the run time development plots reported in Balaprakash et al. (2009a) we can observe the following: RRLS-EE is faster than RRLS-AC by a factor of 1 to 2 on instances of size 300; we could not observe considerable speedup for instance size 100. Concerning RRLS-EE(PTSP), as expected, the algorithm behaves quite similar to RRLS-EE for $fc = 1.0$. However, as the value of fc increases, the algorithm accepts worse solutions during the search due to the adoption of the PTSP-specific evaluation procedure.

Table 7.3 shows the average number of iterations performed by the two algorithms for the given computation time. On instance size 1000, the average number of iterations increases with an increase in the value of p: It increases from 34 to 43 in RRLS-AC but in RRLS-EE, it increases from 330 to 3006. This is due to the effectiveness of ANOVA-Race, which needs only few realizations to select the best solution for large p values. Although the average number of iterations performed by RRLS-EE(PTSP) is higher than that of RRLS-EE, the observed differences are rather small. This shows that the VRPSDC-specific evaluation does not involve a large computational overhead for taking into account the stochastic demands. We can observe a similar trend for instance sizes 100 and 300.

Table 7.4 shows the difference in the average cost between the three algorithms on an instance size of 1000. The results confirm that the VRPSDC-specific estimation

approach is more effective than that of the PTSP-specific estimation approach and that of analytical computation. RRLS-EE obtains average solution costs which are between 0.23% and 1.90% less than that of RRLS-AC. Although the difference in the number of iterations between RRLS-EE and RRLS-AC is quite large, the difference in the final solution cost is rather small. This is due to the adoption of `2.5-opt-EEais`, which, as shown in Section 7.3.2, produces a large improvement at the first iteration. Concerning the comparison between RRLS-EE and RRLS-EE(PTSP), we can observe that the two algorithms achieve similar average costs for $fc = 1.0$, where the instances are similar to the PTSP instances. However, as fc increases, the PTSP-specific estimation approach becomes less effective. Note that for a given value of p, the difference in the average cost between RRLS-EE and RRLS-EE(PTSP) increases up to $fc = 4.0$; the observed difference for $fc = 8.0$ is less than that of $fc = 4.0$. This is because RRLS-EE becomes less effective on instances with $fc = 8.0$. The results on instances of size 100 and 300, which are given in Balaprakash et al. (2009a), exhibit a similar trend except that the difference in the average cost decreases with a decrease in instance size.

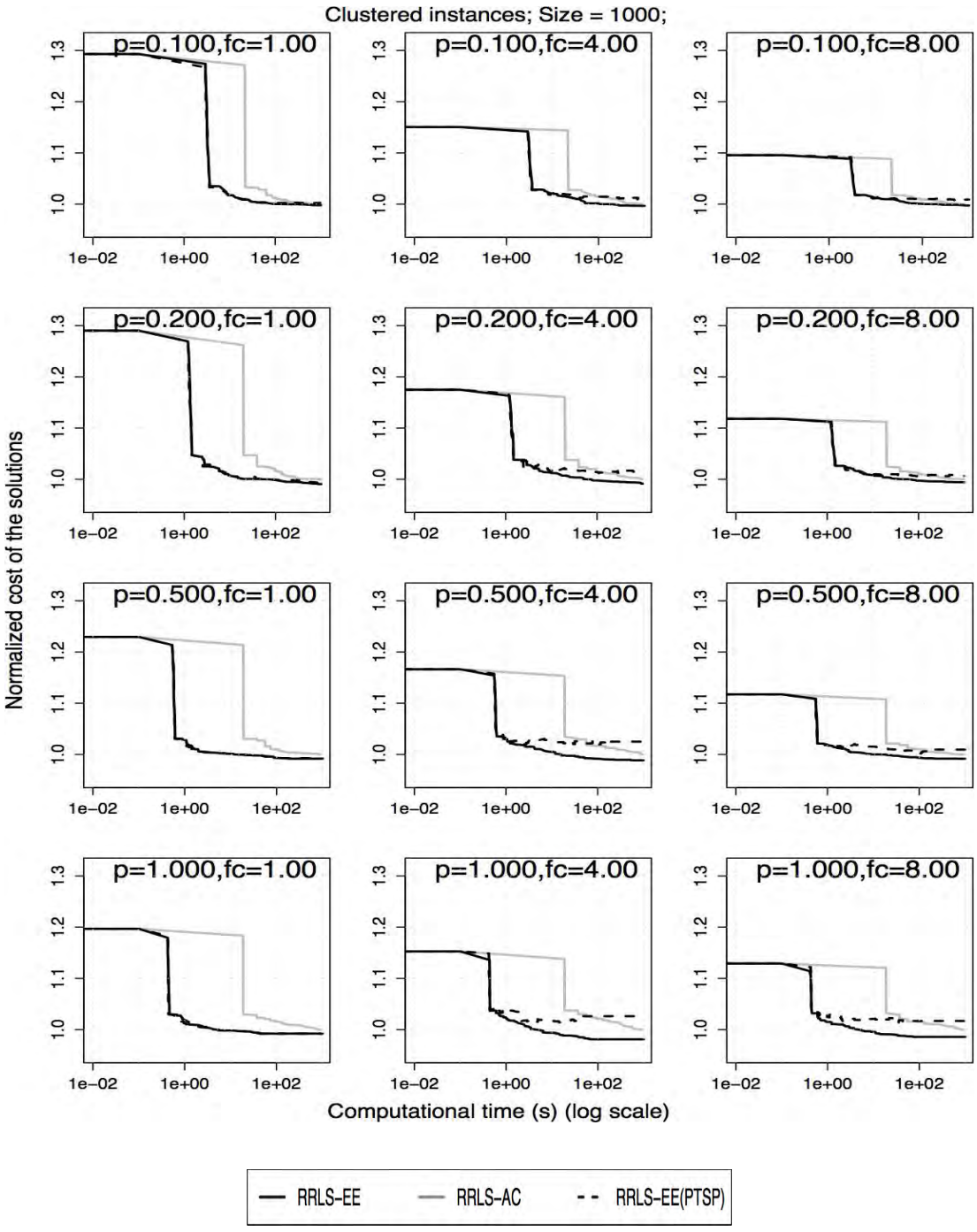


Figure 7.3: Experimental results on clustered VRPSDC instances of size 1000 for 1000 CPU seconds. The plots represent the cost of the solutions obtained by RRLS-EE normalized by the one obtained by RRLS-AC. The normalization is done on an instance-by-instance basis for 10 instances; the normalized solution cost is then aggregated.

Table 7.3: The average number of iterations performed by RRLS-EE and RRLS-AC. Each algorithm is allowed to run for n CPU seconds, where n is the size of the instance.

$p(fc)$	$n = 100$			$n = 300$			$n = 1000$		
	RRLS-EE	RRLS-AC	RRLS-EE (PTSP)	RRLS-EE	RRLS-AC	RRLS-EE (PTSP)	RRLS-EE	RRLS-AC	RRLS-EE (PTSP)
0.100(1.00)	269	220	278	322	150	334	330	34	340
0.100(2.00)	269	219	278	328	153	332	329	36	337
0.100(4.00)	268	217	277	321	150	332	330	34	340
0.100(8.00)	268	216	278	320	147	334	326	30	336
0.200(1.00)	529	364	556	559	223	568	973	45	995
0.200(2.00)	530	363	554	561	221	570	983	44	998
0.200(4.00)	522	359	552	559	220	568	969	43	998
0.200(8.00)	514	353	554	539	218	569	962	41	999
0.300(1.00)	797	464	825	774	251	789	1245	43	1260
0.300(2.00)	809	461	831	777	250	783	1245	43	1259
0.300(4.00)	789	458	828	777	249	784	1243	43	1254
0.300(8.00)	759	452	832	748	246	786	1238	48	1248
0.500(1.00)	1364	594	1408	1208	271	1216	1638	42	1643
0.500(2.00)	1391	592	1398	1202	271	1216	1637	46	1657
0.500(4.00)	1363	589	1399	1201	269	1218	1645	44	1662
0.500(8.00)	1302	581	1402	1182	268	1211	1650	46	1662
0.800(1.00)	2394	715	2445	1956	280	1966	2221	41	2255
0.800(2.00)	2422	716	2440	1944	280	1966	2226	41	2252
0.800(4.00)	2412	713	2452	1966	280	1974	2244	41	2239
0.800(8.00)	2383	705	2435	1959	280	1968	2249	41	2259
1.000(1.00)	3938	800	3976	3025	283	3052	3000	42	3027
1.000(2.00)	3942	800	3967	3034	283	3033	2994	44	3048
1.000(4.00)	3927	796	3962	3025	282	3052	2985	43	3038
1.000(8.00)	3940	791	3975	3021	282	3030	3006	43	3014

7.3.4 Comparison between estimation-based metaheuristics

In this section, we assess the effectiveness of using the three estimation-based metaheuristics ILS-EE, MAGX-EE, and ACS-EE to tackle the VRPSDC. We compare the three algorithms to RRLS-EE on a new set of instances with 100, 300, and 1000 nodes. We allow each algorithm to run for n CPU seconds, where n is the size of the instance.

First, we tuned the parameters of ILS-EE, MAGX-EE, and ACS-EE using Iterative F-Race. For the tuning task, we used a new set of clustered instances with 1000: we generated 120 instances (ten instances times three values of p times four values of fc) for instance Class I and Class II, respectively, and 160 instances (ten instances times four values of p times four values of fc) for instance Class III. Iterative F-Race is run nine times (three metaheuristics times three instance classes), each with a computational budget of 1000 metaheuristic runs. Each metaheuristic run is given a stopping criterion of n (=1000) CPU seconds. Similar to the parameter tuning of PTSP algorithms, the parameter tuning for the VRPSDC algorithms with Iterative F-Race is repeated 10 times. For each metaheuristic and each instance class, we have a set of 10 fine tuned parameter configurations. The obtained parameter configurations are reported in Balaprakash et al. (2009a).

To study the cost of the solutions obtained by each algorithm, we use the expected solution cost of a metaheuristic, where the expectation is taken with respect to the set of 10 parameter configurations and the set of all test instances. The cost of the solutions obtained by ILS-EE, MAGX-EE, and ACS-EE are normalized by the final solution cost reached by RRLS-EE. The normalization is done on an instance-by-instance basis for 10 instances for each p and fc combination.

Figure 7.4 shows exemplary run time development plots of the four estimation-based algorithms over time up to 1000 CPU seconds on instance size 1000. For MAGX-EE and ACS-EE, the plots take into account the improvement incurred by the first local search applied to an individual of the population. Due to the adoption of `2.5-opt-EE-ais`, in all the algorithms the initial solution is improved by 10% to 40% in a very short computation time. Further improvements in the following iterations are considerably smaller than that in the first iteration.

Figure 7.5 shows the box plot of the solution cost of the algorithms on instance size 100, 300, and 1000. We can observe that ILS-EE, MAGX-EE, and ACS-EE outperform RRLS-EE across most probability levels. The difference in the solution cost between RRLS-EE and the other algorithms increases with an increase in instance size. For a given instance size, the observed differences in the solution cost between the four

Table 7.4: Comparison of the average cost obtained by RRLS-EE and RRLS-AC over 10 instances of size 1000. Typographic conventions are the same as in Table 7.2.

	$n = 1000$					
	RRLS-EE vs. RRLS-AC		RRLS-EE vs. RRLS-EE(PTSP)		RRLS-AC vs. RRLS-EE(PTSP)	
$p(fc)$	d	[95% CI]	d	[95% CI]	d	[95% CI]
0.100(1.00)	**−0.23**	[−0.40, −0.06]	−0.45	[−0.94, +0.04]	−0.22	[−0.65, +0.21]
0.100(2.00)	**−0.33**	[−0.52, −0.15]	**−3.83**	[−6.56, −1.10]	**−3.51**	[−6.24, −0.77]
0.100(4.00)	**−0.36**	[−0.67, −0.06]	**−1.68**	[−2.79, −0.57]	**−1.32**	[−2.43, −0.21]
0.100(8.00)	**−0.26**	[−0.43, −0.09]	**−1.14**	[−1.73, −0.55]	**−0.88**	[−1.46, −0.30]
0.200(1.00)	**−0.94**	[−1.29, −0.58]	−0.24	[−0.49, +0.01]	*+0.70*	[+0.23, +1.17]
0.200(2.00)	**−0.90**	[−1.26, −0.54]	**−3.89**	[−6.47, −1.32]	**−3.02**	[−5.50, −0.55]
0.200(4.00)	**−0.87**	[−1.45, −0.30]	**−2.17**	[−3.30, −1.05]	**−1.31**	[−2.44, −0.18]
0.200(8.00)	**−0.58**	[−0.81, −0.35]	**−1.21**	[−1.97, −0.46]	−0.64	[−1.47, +0.19]
0.300(1.00)	**−0.73**	[−1.16, −0.31]	**−0.24**	[−0.47, −0.01]	*+0.50*	[+0.02, +0.97]
0.300(2.00)	−1.05	[−2.18, +0.07]	**−3.69**	[−5.68, −1.69]	**−2.66**	[−4.62, −0.70]
0.300(4.00)	**−0.84**	[−1.29, −0.39]	**−2.40**	[−3.46, −1.35]	**−1.58**	[−2.83, −0.32]
0.300(8.00)	**−0.32**	[−0.48, −0.17]	**−1.22**	[−2.02, −0.41]	**−0.90**	[−1.73, −0.06]
0.500(1.00)	**−0.83**	[−1.18, −0.48]	−0.10	[−0.26, +0.06]	*+0.73*	[+0.28, +1.19]
0.500(2.00)	**−1.23**	[−1.97, −0.48]	**−3.61**	[−5.67, −1.55]	**−2.41**	[−4.24, −0.59]
0.500(4.00)	**−1.15**	[−1.94, −0.36]	**−3.56**	[−4.65, −2.46]	**−2.44**	[−3.52, −1.35]
0.500(8.00)	**−0.85**	[−1.20, −0.51]	**−1.78**	[−2.72, −0.84]	−0.94	[−1.95, +0.08]
0.800(1.00)	**−0.76**	[−1.11, −0.40]	−0.03	[−0.08, +0.02]	*+0.73*	[+0.36, +1.10]
0.800(2.00)	**−1.13**	[−1.75, −0.51]	**−2.53**	[−4.06, −0.99]	−1.41	[−2.99, +0.17]
0.800(4.00)	**−1.10**	[−1.74, −0.47]	**−2.91**	[−4.33, −1.49]	**−1.83**	[−2.93, −0.73]
0.800(8.00)	**−0.57**	[−1.05, −0.09]	**−2.19**	[−3.30, −1.09]	**−1.63**	[−2.75, −0.51]
1.000(1.00)	**−0.79**	[−1.18, −0.39]	0.00	[0.00, 0.00]	*+0.79*	[+0.40, +1.19]
1.000(2.00)	**−1.10**	[−1.97, −0.22]	**−3.60**	[−5.40, −1.80]	**−2.53**	[−4.46, −0.60]
1.000(4.00)	**−1.90**	[−3.10, −0.70]	**−4.40**	[−5.78, −3.01]	**−2.54**	[−3.90, −1.19]
1.000(8.00)	**−1.42**	[−2.04, −0.80]	**−3.06**	[−4.51, −1.61]	−1.66	[−3.36, +0.03]

estimation algorithms increases with an increase in the node probability p. This shows that for instances with small p values, it is rather easy to find high quality solutions by restarting `2.5-opt-EEais` a number of times. Nevertheless, for instances with large p values, in addition to `2.5-opt-EEais`, the use of sophisticated metaheuristics is crucial to find high quality solutions.

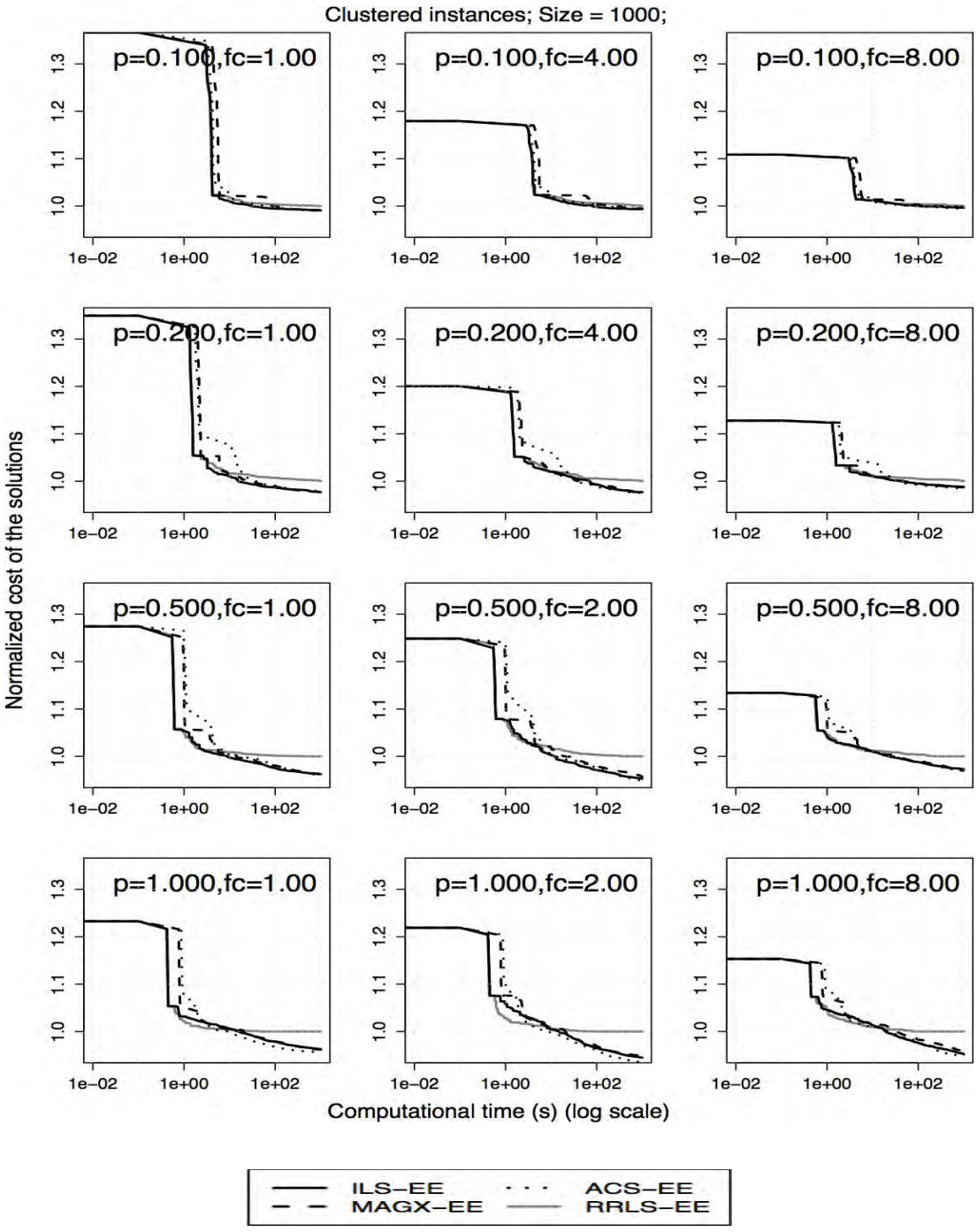
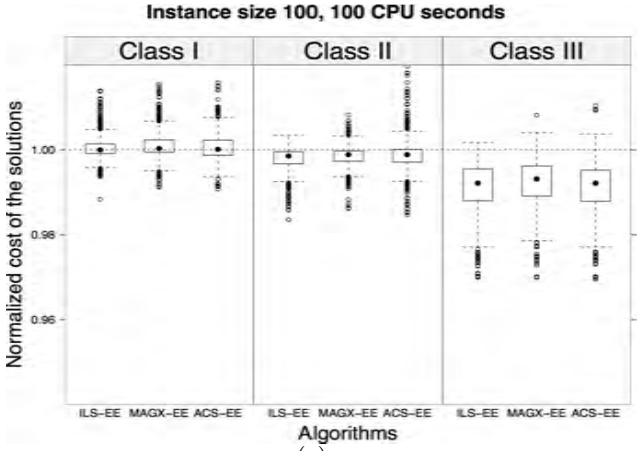
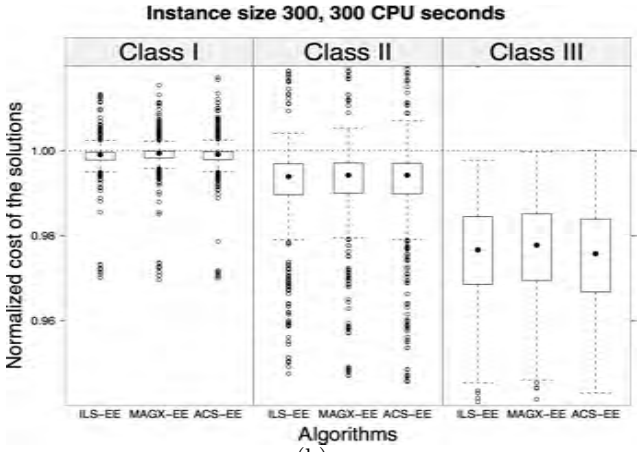


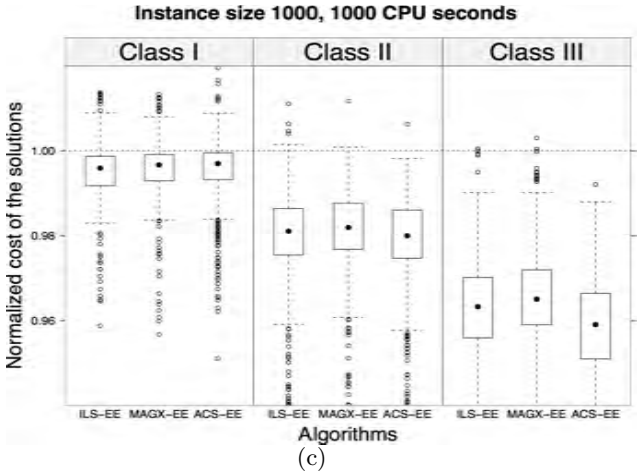
Figure 7.4: Experimental results on clustered VRPSDC instances of size 1000 for 1000 CPU seconds. The plots represent the cost of the solutions obtained by ILS-EE, MAGX-EE, and ACS-EE normalized by the one obtained by RRLS-EE. The normalization is done on an instance-by-instance basis for 10 instances; the normalized solution cost is then aggregated.



(a)



(b)



(c)

Figure 7.5: Experimental results on clustered VRPSDC instances. The box plots represent the cost of the solutions obtained by ILS-EE, MAGX-EE, and ACS-EE. The obtained solution costs of the algorithms are normalized by the final solution cost reached by RRLS-EE. The normalization is done on an instance-by-instance basis for 10 instances; the normalized solution cost is then aggregated. The dotted horizontal line denotes therefore the final cost of RRLS-EE.

Table 7.5: Comparison of the average cost obtained by ILS-EE, MAGX-EE, ACS-EE and RRLS-EE, on clustered instances of size 1000 for 1000 CPU seconds. Typographic conventions are the same as in Table 7.2.

		ILS-EE vs. MAGX-EE		ILS-EE vs. ACS-EE		ILS-EE vs. RRLS-EE		MAGX-EE vs. ACS-EE		MAGX-EE vs. RRLS-EE		ACS-EE vs. RRLS-EE	
	$p(fc)$	d	CI	d	CI	d	CI	d	CI	d	CI	d	CI
Class I	0.050(1.00)	**−0.05**	[−0.08, −0.03]	**−0.11**	[−0.13, −0.08]	**−0.20**	[−0.23, −0.16]	**−0.05**	[−0.08, −0.03]	**−0.14**	[−0.19, −0.10]	**−0.09**	[−0.13, −0.05]
	0.050(2.00)	−0.03	[−0.23, +0.18]	−0.11	[−0.31, +0.09]	+0.01	[−0.21, +0.23]	−0.08	[−0.30, +0.13]	+0.04	[−0.18, +0.25]	+0.12	[−0.10, +0.35]
	0.050(4.00)	**−0.22**	[−0.30, −0.14]	**−0.13**	[−0.21, −0.05]	**−0.20**	[−0.27, −0.12]	*+0.09*	[+0.01, +0.18]	+0.03	[−0.05, +0.10]	**−0.07**	[−0.13, −0.01]
	0.050(8.00)	**−0.04**	[−0.08, −0.01]	−0.01	[−0.05, +0.04]	+0.05	[−0.00, +0.11]	+0.03	[−0.01, +0.08]	*+0.09*	[+0.04, +0.15]	*+0.06*	[+0.02, +0.11]
	0.075(1.00)	**−0.04**	[−0.08, −0.01]	**−0.09**	[−0.14, −0.05]	**−0.47**	[−0.52, −0.43]	**−0.05**	[−0.08, −0.02]	**−0.43**	[−0.47, −0.39]	**−0.38**	[−0.42, −0.34]
	0.075(2.00)	−0.12	[−0.24, +0.00]	−0.12	[−0.27, +0.03]	**−0.99**	[−1.18, −0.80]	−0.00	[−0.15, +0.15]	**−0.87**	[−1.05, −0.70]	**−0.87**	[−1.05, −0.69]
	0.075(4.00)	**−0.13**	[−0.22, −0.04]	−0.11	[−0.26, +0.04]	**−0.44**	[−0.54, −0.33]	+0.02	[−0.11, +0.14]	**−0.31**	[−0.41, −0.21]	**−0.33**	[−0.43, −0.22]
	0.075(8.00)	**−0.08**	[−0.15, −0.00]	−0.03	[−0.10, +0.03]	**−0.29**	[−0.37, −0.21]	+0.04	[−0.03, +0.12]	**−0.21**	[−0.29, −0.13]	**−0.25**	[−0.31, −0.20]
	0.100(1.00)	**−0.10**	[−0.14, −0.05]	**−0.15**	[−0.20, −0.10]	**−1.02**	[−1.07, −0.97]	−0.05	[−0.11, +0.01]	**−0.92**	[−0.99, −0.86]	**−0.87**	[−0.93, −0.81]
	0.100(2.00)	−0.11	[−0.25, +0.02]	−0.07	[−0.23, +0.08]	**−1.15**	[−1.24, −1.06]	+0.04	[−0.14, +0.22]	**−1.04**	[−1.18, −0.90]	**−1.08**	[−1.22, −0.93]
	0.100(4.00)	−0.07	[−0.16, +0.03]	−0.06	[−0.18, +0.06]	**−0.67**	[−0.76, −0.58]	+0.00	[−0.11, +0.12]	**−0.61**	[−0.70, −0.51]	**−0.61**	[−0.72, −0.49]
	0.100(8.00)	−0.05	[−0.11, +0.01]	*+0.11*	[+0.01, +0.21]	**−0.36**	[−0.48, −0.24]	*+0.16*	[+0.05, +0.27]	**−0.31**	[−0.41, −0.20]	**−0.47**	[−0.60, −0.34]
	overall	**−0.09**	[−0.11, −0.06]	**−0.07**	[−0.11, −0.04]	**−0.48**	[−0.51, −0.44]	+0.01	[−0.02, +0.05]	**−0.39**	[−0.43, −0.35]	**−0.40**	[−0.44, −0.36]
Class II	0.150(1.00)	**−0.10**	[−0.16, −0.05]	+0.04	[−0.04, +0.12]	**−1.62**	[−1.70, −1.54]	*+0.14*	[+0.06, +0.22]	**−1.52**	[−1.60, −1.44]	**−1.66**	[−1.75, −1.57]
	0.150(2.00)	**−0.22**	[−0.32, −0.11]	−0.11	[−0.28, +0.06]	**−2.50**	[−2.75, −2.25]	+0.11	[−0.04, +0.25]	**−2.29**	[−2.54, −2.04]	**−2.39**	[−2.61, −2.16]
	0.150(4.00)	−0.04	[−0.17, +0.10]	+0.08	[−0.05, +0.21]	**−1.54**	[−1.74, −1.34]	*+0.11*	[+0.00, +0.23]	**−1.51**	[−1.69, −1.32]	**−1.62**	[−1.79, −1.45]
	0.150(8.00)	**−0.10**	[−0.18, −0.02]	+0.07	[−0.03, +0.18]	**−1.16**	[−1.27, −1.05]	*+0.17*	[+0.07, +0.28]	**−1.06**	[−1.17, −0.95]	**−1.23**	[−1.38, −1.09]
	0.175(1.00)	−0.02	[−0.09, +0.06]	+0.07	[−0.01, +0.14]	**−1.99**	[−2.08, −1.91]	*+0.08*	[+0.03, +0.14]	**−1.98**	[−2.06, −1.89]	**−2.06**	[−2.15, −1.97]
	0.175(2.00)	**−0.18**	[−0.28, −0.08]	+0.04	[−0.10, +0.19]	**−2.62**	[−2.87, −2.36]	*+0.22*	[+0.07, +0.38]	**−2.44**	[−2.69, −2.19]	**−2.66**	[−2.92, −2.40]
	0.175(4.00)	−0.09	[−0.22, +0.04]	+0.12	[−0.02, +0.26]	**−2.02**	[−2.15, −1.90]	*+0.22*	[+0.07, +0.36]	**−1.93**	[−2.04, −1.83]	**−2.14**	[−2.29, (2.00)]
	0.175(8.00)	−0.06	[−0.14, +0.03]	*+0.15*	[+0.05, +0.24]	**−1.22**	[−1.29, −1.14]	*+0.20*	[+0.10, +0.31]	**−1.16**	[−1.24, −1.08]	**−1.36**	[−1.47, −1.25]
	0.200(1.00)	**−0.08**	[−0.16, −0.01]	+0.02	[−0.06, +0.10]	**−2.30**	[−2.38, −2.21]	*+0.10*	[+0.02, +0.18]	**−2.22**	[−2.31, −2.12]	**−2.32**	[−2.40, −2.23]
	0.200(2.00)	**−0.23**	[−0.41, −0.04]	+0.11	[−0.06, +0.28]	**−2.90**	[−3.10, −2.69]	*+0.34*	[+0.20, +0.48]	**−2.67**	[−2.83, −2.52]	**−3.00**	[−3.17, −2.84]
	0.200(4.00)	−0.05	[−0.19, +0.10]	*+0.19*	[+0.04, +0.35]	**−2.50**	[−2.67, −2.34]	*+0.24*	[+0.07, +0.41]	**−2.46**	[−2.62, −2.30]	**−2.69**	[−2.87, −2.51]
	0.200(8.00)	**−0.13**	[−0.23, −0.03]	*+0.18*	[+0.10, +0.27]	**−1.44**	[−1.53, −1.35]	*+0.31*	[+0.22, +0.41]	**−1.31**	[−1.39, −1.23]	**−1.62**	[−1.72, −1.51]
	overall	**−0.11**	[−0.14, −0.08]	*+0.08*	[+0.05, +0.12]	**−1.98**	[−2.04, −1.93]	*+0.19*	[+0.15, +0.22]	**−1.88**	[−1.93, −1.83]	**−2.06**	[−2.12, −2.01]
Class III	0.300(1.00)	−0.08	[−0.22, +0.06]	+0.01	[−0.13, +0.14]	**−3.38**	[−3.53, −3.24]	*+0.09*	[+0.01, +0.16]	**−3.30**	[−3.40, −3.21]	**−3.39**	[−3.47, −3.31]
	0.300(2.00)	−0.12	[−0.32, +0.09]	*+0.20*	[+0.01, +0.40]	**−3.79**	[−3.99, −3.58]	*+0.32*	[+0.16, +0.48]	**−3.67**	[−3.83, −3.52]	**−3.98**	[−4.18, −3.78]
	0.300(4.00)	**−0.28**	[−0.48, −0.08]	+0.04	[−0.16, +0.24]	**−3.43**	[−3.65, −3.21]	*+0.32*	[+0.12, +0.52]	**−3.16**	[−3.38, −2.94]	**−3.47**	[−3.63, −3.31]
	0.300(8.00)	**−0.27**	[−0.39, −0.15]	−0.07	[−0.17, +0.03]	**−2.10**	[−2.20, (2.00)]	*+0.20*	[+0.07, +0.33]	**−1.84**	[−1.94, −1.73]	**−2.03**	[−2.11, −1.95]
	0.500(1.00)	+0.13	[−0.06, +0.31]	*+0.48*	[+0.29, +0.67]	**−3.67**	[−3.84, −3.50]	*+0.35*	[+0.27, +0.42]	**−3.80**	[−3.86, −3.74]	**−4.13**	[−4.19, −4.07]
	0.500(2.00)	−0.11	[−0.37, +0.15]	*+0.77*	[+0.52, +1.02]	**−4.54**	[−4.86, −4.23]	*+0.88*	[+0.69, +1.07]	**−4.44**	[−4.69, −4.19]	**−5.27**	[−5.53, −5.01]
	0.500(4.00)	−0.09	[−0.37, +0.19]	*+0.68*	[+0.39, +0.97]	**−3.98**	[−4.26, −3.69]	*+0.77*	[+0.60, +0.94]	**−3.90**	[−4.11, −3.68]	**−4.63**	[−4.82, −4.44]
	0.500(8.00)	+0.00	[−0.14, +0.14]	*+0.30*	[+0.14, +0.45]	**−2.98**	[−3.09, −2.86]	*+0.29*	[+0.15, +0.44]	**−2.98**	[−3.09, −2.87]	**−3.26**	[−3.39, −3.14]
	0.800(1.00)	**−0.32**	[−0.45, −0.18]	*+0.68*	[+0.54, +0.81]	**−3.60**	[−3.76, −3.45]	*+1.00*	[+0.88, +1.12]	**−3.30**	[−3.45, −3.15]	**−4.25**	[−4.36, −4.15]
	0.800(2.00)	**−0.93**	[−1.22, −0.64]	*+0.66*	[+0.43, +0.89]	**−5.10**	[−5.35, −4.84]	*+1.61*	[+1.35, +1.86]	**−4.21**	[−4.45, −3.96]	**−5.72**	[−5.94, −5.50]
	0.800(4.00)	**−0.64**	[−0.91, −0.37]	*+1.01*	[+0.80, +1.22]	**−4.50**	[−4.77, −4.24]	*+1.66*	[+1.40, +1.92]	**−3.89**	[−4.17, −3.60]	**−5.46**	[−5.70, −5.22]
	0.800(8.00)	**−0.43**	[−0.64, −0.21]	*+0.83*	[+0.63, +1.03]	**−3.79**	[−3.99, −3.59]	*+1.27*	[+1.10, +1.43]	**−3.38**	[−3.54, −3.22]	**−4.59**	[−4.73, −4.44]
	1.000(1.00)	*+0.29*	[+0.16, +0.43]	*+0.85*	[+0.71, +0.98]	**−3.63**	[−3.77, −3.49]	*+0.55*	[+0.46, +0.64]	**−3.91**	[−4.02, −3.81]	**−4.44**	[−4.53, −4.35]
	1.000(2.00)	−0.08	[−0.34, +0.17]	*+1.10*	[+0.84, +1.36]	**−5.28**	[−5.55, −5.00]	*+1.18*	[+0.94, +1.43]	**−5.20**	[−5.42, −4.97]	**−6.30**	[−6.52, −6.09]
	1.000(4.00)	−0.14	[−0.42, +0.15]	*+1.17*	[+0.90, +1.45]	**−4.82**	[−5.16, −4.47]	*+1.31*	[+1.09, +1.53]	**−4.69**	[−5.02, −4.35]	**−5.92**	[−6.22, −5.62]
	1.000(8.00)	−0.07	[−0.35, +0.20]	*+0.89*	[+0.60, +1.18]	**−4.70**	[−4.97, −4.44]	*+0.97*	[+0.74, +1.19]	**−4.63**	[−4.84, −4.43]	**−5.55**	[−5.77, −5.33]
	overall	**−0.20**	[−0.25, −0.14]	*+0.60*	[+0.54, +0.65]	**−3.96**	[−4.02, −3.89]	*+0.79*	[+0.74, +0.84]	**−3.77**	[−3.83, −3.71]	**−4.52**	[−4.60, −4.45]
	overall	**−0.14**	[−0.16, −0.11]	*+0.23*	[+0.21, +0.26]	**−2.32**	[−2.38, −2.26]	*+0.37*	[+0.34, +0.40]	**−2.19**	[−2.24, −2.13]	**−2.55**	[−2.61, −2.49]

Table 7.5 reports the observed relative difference between the solution costs obtained by the algorithms with a 95% confidence bound obtained through a t-test on instances of size 1000. The results confirm that the three sophisticated estimation-based metaheuristics are more effective than RRLS-EE and that they obtain average solution costs, which are significantly less than the cost of the best solution obtained by RRLS-EE on a wide range of instance sizes and probability levels: ILS-EE, MAGX-EE, and ACS-EE obtains average solution cost that are 2.32%, 2.19%, and 2.55% less than that of RRLS-EE, respectively. The differences in the average solution costs between ILS-EE, MAGX-EE, and ACS-EE are rather small and the observed differences are less than 1%. On Class I instances, the average solution cost obtained by ILS-EE is 0.09% and 0.07% less than that of MAGX-EE and ACS-EE, respectively. On Class II instances, the average solution cost obtained by ACS-EE is 0.08% and 0.19% less than that of ILS-EE and MAGX-EE, respectively. On Class III instances, the observed differences are 0.60% and 0.79%. The aggregated results over all instances show that ACS-EE is more effective than ILS-EE and MAGX-EE: the average solution cost obtained by ACS-EE is 0.23% and 0.37% less than that of ILS-EE and MAGX-EE, respectively. The general trends of the results on instances of size 100 and 300, which are reported in Balaprakash et al. (2009a), are consistent with the results presented here except that the observed differences decrease as instance size gets smaller.

7.4 Summary

In this chapter, we extended the estimation-based PTSP algorithms to tackle the VRPSDC. The proposed extension primarily consists in using a VRPSDC-specific cost evaluation procedure and in tuning the parameters of the algorithms for the VRPSDC. All estimation-based metaheuristics use the PTSP iterative improvement algorithm `2.5-opt-EEais` as the local search.

We carried out three sets of experiments. In the first set, we investigated the effectiveness of using `2.5-opt-EEais` to tackle the VRPSDC. We showed that a simple random restart local search that adopts `2.5-opt-EEais` as the local search outperforms the existing tailor-made tabu search algorithm. In the second set, we assessed the VRPSDC cost estimation procedure by comparing it with the currently used VRPSDC analytical computation procedure. We studied the two cost evaluation procedures within a random restart local search algorithm. We demonstrated that the proposed VRPSDC cost estimation procedure is highly effective as it substantially reduces the computation time needed to compare the cost of the solutions. Motivated by the ef-

fectiveness of `2.5-opt-EEais`, we also used a random restart local search algorithm developed for the PTSP to tackle the VRPSDC. The poor performance of this algorithm showed that it is important to take into account the stochastic demands and the resulting recourse actions when deciding between solutions. Finally, in the third set, we studied the proposed estimation-based iterated local search, memetic, and ant colony optimization algorithms. All three algorithms found high quality solutions that are better than that of random restart local search. The difference in the solution cost between iterated local search, memetic, and ant colony optimization algorithms are rather small. Besides the adoption of the highly effective PTSP iterative improvement algorithm `2.5-opt-EEais` as local search, the high performance of iterated local search, memetic, and ant colony optimization algorithms is ascribed to the rigorous parameter tuning, which is performed by grouping the instances into a number of classes and by tuning each algorithm on each instance class. Although the observed differences are small, the estimation-based ant colony optimization algorithm is slightly more effective than the iterated local search and memetic algorithms.

The two main contributions of the chapter are the following. From a methodological perspective, for the first time the VRPSDC is tackled using the empirical estimation approach and it is shown to be more effective than the previously proposed analytical computation approach, particularly for large instances. From an algorithmic point-of-view, we showed that the proposed estimation-based algorithms are more effective than the existing analytical computation tabu search algorithm with respect to computation time and solution quality. This allowed us to obtain new state-of-the-art algorithms for the VRPSDC.

Chapter 8

Conclusions

Stochastic combinatorial optimization problems are optimization problems in which problem parameters are affected by uncertainty; however, probability distributions describing the uncertainty are known or can be estimated. In this thesis, we focused on stochastic routing problems, a prominent class of stochastic combinatorial optimization problems. Stochastic routing problems involve finding an efficient way to distribute or collect goods across a logistic network. In order to tackle these problems, a setting is considered in which the cost of each solution is a random variable, and the customary goal is to find the solution that minimizes the expectation of the latter. It has been shown that, for some problems and for known probability distributions, the expectation can be computed analytically. Unfortunately, this typically involves complex analytical developments and computationally expensive procedures. Moreover, computing the expectation through the analytical computation approach is a highly problem-specific issue and it requires a deep understanding of the underlying probabilistic model. An alternative approach is empirical estimation that estimates the expectation through Monte Carlo simulation. The main advantage of the empirical estimation approach over the analytical computation approach is generality: a sample estimate of the expected cost of a given solution can be obtained by simply averaging sample cost estimates over a number of realizations of the random variable.

To tackle stochastic routing problems, stochastic local search algorithms such as iterative improvement algorithms and metaheuristics are quite promising because they offer effective strategies to tackle the combinatorial nature of these problems. However, a crucial factor that determines the success of these algorithms in stochastic settings is the trade-off between the computation time needed to search high quality solutions and the computation time needed for computing the cost of the solutions obtained during

the search.

In previous research on stochastic routing problems, most commonly the analytical computation approach was used to design iterative improvement algorithms and metaheuristics. This is particularly the case for the prototypical examples of stochastic routing problems, such as the probabilistic traveling salesman problem (PTSP) and the vehicle routing problem with stochastic demands and customers (VRPSDC). It was even conjectured that for the PTSP an estimation-based approach is less effective (Bianchi, 2006). The main contribution of this thesis is to show that the estimation-based approach is actually a viable approach even for the simple, prototypical stochastic routing problems, outperforming by quite a strong margin previously proposed algorithms. In particular, this was shown for the PTSP and the VRPSDC. More in detail, the contributions of this thesis can be summarized as follows.

Estimation-based iterative improvement algorithm for the PTSP

The thesis proposes an effective estimation-based iterative improvement algorithm for the PTSP. The main novelty of the proposed algorithm consists of using the empirical estimation approach and an effective data structure for the PTSP delta evaluation. Inspired by iterative improvement algorithms proposed for the closely related traveling salesman problem, a particular attention is given to the adoption of neighborhood reduction techniques. A systematic experimental analysis is carried out to assess the effectiveness of each algorithmic component adopted in the estimation-based iterative improvement algorithm.

Adaptive sample size and variance reduction techniques in delta evaluation for the PTSP

Adaptive sample size procedures offer a computational benefit by prescribing the most appropriate number of realizations need for cost estimation. Variance reduction techniques are used to reduce the number of realizations needed for obtaining precise cost estimates. The thesis proposes a new way of using two variance reduction techniques, namely, method of common random numbers and importance sampling. The resulting algorithm that combines these two techniques together with the adaptive sample size procedure is our final, new state-of-the-art iterative improvement algorithm called `2.5-opt-EEais`.

Estimation-based metaheuristics for the PTSP

For the PTSP, the estimation-based metaheuristics proposed so far in the literature are proof-of-concept algorithms because they do not use any local search as a subsidiary solution improvement procedure.

The thesis discusses three high performing estimation-based metaheuristics, namely, iterated local search, memetic, and ant colony optimization algorithms, to solve the PTSP. These algorithms use the empirical estimation approach to evaluate the solution cost and exploit the estimation-based iterative improvement algorithm as subsidiary solution improvement procedure. A particularity of the estimation approach is the adoption of the method of common random numbers and the adaptive sample size procedure that uses statistical tests. The parameters of all estimation-based algorithms are rigorously fine-tuned and tested on instances with different characteristics.

Estimation-based metaheuristics for VRPSDC

So far, the VRPSDC has been tackled by an analytical computation algorithm that adopts a computationally expensive closed-form expression. However, for large instances with several hundreds of nodes, the high computational overhead, due to the adoption of the analytical computation approach, affects the performance of the algorithm considerably. For the first time, the estimation-based approach is adopted within metaheuristics to tackle the VRPSDC and it is shown to be more effective than the currently used analytical computation approach.

Advancement of the state-of-the-art for the PTSP and the VRPSDC

The estimation-based algorithms proposed in the thesis are compared to the previously proposed algorithms using an accurately designed and statistically sound experimental methodology. Primary importance is given to assess the performance of the algorithms on large instances. The estimation-based iterative improvement algorithm for the PTSP discussed in this thesis is up to two orders of magnitude faster to reach an average solution quality similar to that of the previously proposed iterative improvement algorithms. The estimation-based metaheuristics for the PTSP proposed in this thesis are more effective than the best available metaheuristic algorithms with respect to solution quality and computation time. Similarly, the estimation-based metaheuristics developed for the VRPSDC define the new state-of-the-art.

Engineering algorithms for stochastic routing problems

The thesis applies a principled approach to develop high performing algorithms for stochastic routing problems. The adopted approach is a bottom-up engineering process that starts from basic algorithms and adds complexity step-by-step. In order to tackle the PTSP and the VRPSDC, the engineering process is strongly focused on the development and refinement of the PTSP iterative improvement algorithm. This process is supported by an efficient implementation of algorithm data structures, comprehensive experimental analysis, and the usage of the automatic tuning of algorithm parameters. Although not always necessary in other stochastic combinatorial optimization problems, the strong focus on the iterative improvement algorithm played a central role in attaining state-of-the-art metaheuristics for the PTSP and the VRPSDC.

Iterative F-Race for parameter tuning

The thesis introduces Iterative F-Race for tuning the parameter values of algorithms. While typically parameters are tuned by hand, recent studies have shown that automatic tuning procedures can effectively handle this task and they often find better parameter values. F-race has been proposed specifically for this purpose and it has proven to be very effective in a number of cases. Iterative F-Race is an improved variant of `F-Race` that, on the one hand, is suitable for tuning tasks with a large number of initial parameter values and, on the other hand, allows a significant reduction of the computation time needed for tuning tasks without any major loss in solution quality. This algorithm is used to fine tune the parameter values of all estimation-based algorithms developed for the PTSP and the VRPSDC.

Open issues

The extensive empirical evidence on the effectiveness of the estimation-based algorithms puts forward a number of open issues and avenues for further research.

Investigations for the VRPSDC

Further work will be devoted to extend the proposed estimation-based algorithms for the VRPSDC with multiple vehicles. Given the observed superior performance of our PTSP-specific iterative improvement algorithm for the VRPSDC and the previously proposed analytical approximation scheme that ignores stochastic demands, it seems that the element of stochastic demands, which makes delta evaluation difficult, is not crucial. Also, for the related VRPSD, Bianchi et al. (2006) showed that the TSP approximation is better than the problem-specific delta evaluation. In this context, the impact of stochastic demands in the VRPSD and the VRPSDC needs further investigation.

Improving estimation-based comparison

In all the proposed estimation-based algorithms, evaluation procedures are based on inferential statistics. There are several other methodologies proposed in the simulation optimization literature such as ordinal optimization that decides which solution is better by ranking the solution costs. Branke et al. (2007) already made an extensive experimental investigation of several comparison procedures and proposed a framework to compare them. Using this framework to identify the best comparison procedure for the proposed estimation-based algorithms will eventually increase their performance.

Variance reduction techniques

Given that the adoption of variance reduction techniques in the proposed estimation-based algorithms is quite effective, further research effort will be devoted to the adoption of other variance reduction techniques. In particular, the possibility of using antithetic variates and control variates within the proposed estimation-based algorithms will be explored.

Extension to other stochastic routing problems

The generality of the empirical estimation approach offers a lot of flexibility to extend the proposed estimation-based algorithms to other stochastic routing problems, in par-

ticular, problems that include stochastic travel times. A possible future work is to develop general-purpose algorithms that can tackle routing problems with stochastic customers, stochastic demands, and stochastic travel times.

Stochastic constraints

In numerous practical settings, combinatorial optimization problems have stochastic constraints. However, the design and development of algorithms for these problems has received relatively little attention. The estimation-based approach puts forward a lot of versatility to handle stochastic constraints. Developing estimation-based iterative improvement algorithms and metaheuristics for these problems is an open area that needs to be pursued.

Multi-objective stochastic optimization

In the stochastic combinatorial optimization problem literature, little work is done on the development of algorithms to tackle problems that have several (often conflicting) cost functions to evaluate a solution. A promising research direction is to investigate the application of estimation-based algorithms to multi-objective stochastic combinatorial optimization problems.

Comparison with other algorithmic approaches and hybridization

Algorithms that are exclusively developed for stochastic combinatorial optimization such as stochastic partition methods will be considered for an empirical comparison with the proposed estimation-based algorithms. Hybridization of stochastic partition methods with the proposed estimation-based algorithms is a promising area of research that needs to be investigated.

Comparison with the re-optimization approach

Another interesting topic to investigate is the extension of the PTSP model to allow full or limited re-optimizations of *a posteriori* solutions; in other words, different recourse actions could be used, for example, by applying local search to the resulting *a posteriori* solutions. This would be particularly attractive for the PTSP instances with low probability values, where the *a posteriori* solutions contain only a small number of nodes. In these settings, depending on the application context, the *a priori* optimization might be less useful given that the *a posteriori* solutions will be very different from the *a priori* ones.

Stochastic local search engineering

Stochastic local search engineering is relatively a new area of research and it has received increased attention in recent years. In fact, the bottom-up engineering process that we followed in the thesis played a crucial role in the development of effective iterative improvement and metaheuristic algorithms. We believe that this bottom-up engineering process is potentially a successful way of deriving high performing algorithms for other complex stochastic routing problems. In this context, there are a number of avenues open for further research. One main focus of current research in stochastic local search engineering is to develop a framework of principled procedures for algorithm design, implementation, analysis, and in-depth experimental studies. In this context, there is an indispensable need for the development of tools that support the algorithm engineering such as efficient data structures, software frameworks, statistical analysis, experimental design, automated tuning, and search space analysis.

Improving Iterative F-Race

Further research is currently being carried out to extend the parameter tuning algorithm Iterative F-Race to include categorical variables. We will also investigate the adoption of distributions like Cauchy and some advanced techniques for updating the distribution. Finally, based on the case studies that were made with Iterative F-Race, we speculate that the difficulty of the tuning task depends on a number of factors such as the sensitivity of the parameters that need to be tuned and problem instances that need to be tackled. In this context, search space analysis on the parameter values of algorithms is an important area to investigate further. The adoption of the response surface methodology and of prediction algorithms within Iterative F-Race is also a future line of research.

Concluding statement

Designing effective algorithms for stochastic routing problems is a difficult task. The difficulty is due to the element of uncertainty in the problem parameters, which increases the difficulty of finding an optimal solution in a large search space. Although iterative improvement and metaheuristic algorithms are promising and practical techniques to tackle these problems, their full potential for tackling stochastic combinatorial optimization problems has not yet been fully explored. This is primarily due to the adoption of the computationally expensive analytical computation approach. The use of the alternative approach, the empirical estimation approach, has received relatively little consideration from the research community.

In this thesis, a principled adoption of the empirical estimation approach in iterative improvement and metaheuristic algorithms is shown to be extremely effective for tackling stochastic routing problems. The proposed estimation-based iterative improvement algorithms and metaheuristics for the two case study problems, the probabilistic traveling salesman problem and the vehicle routing problem with stochastic demands and customers, define the new state-of-the-art.

Appendices

Appendix A

Estimation-based Ant Colony Optimization and Local Search for the PTSP

In Section 6.1 of Chapter 6, we mentioned that we studied several ACO algorithms to solve the PTSP. In this appendix, we present this study. Given that the best analytical computation algorithm for the PTSP is pACS+1-shift (see Chapter 3 for an explanation), we use it as a starting point; in Section A.1, we replace `1-shift` with our estimation-based iterative improvement algorithm, `2.5-opt-EEais` in pACS and we show that pACS+2.5-opt-EEais outperforms pACS+1-shift; in Section A.2, we bring the estimation-based solution evaluation into pACS+2.5-opt-EEais and we show that the cost evaluation performed by the estimation-based approach is comparable to that of the analytical computation approach; in Section A.3, we customize three high performing ACO variants, $\mathcal{MAX}$–$\mathcal{MIN}$ ant system, rank-based ant system, and best-worst ant system. We compare the three variants to ACS and we show that the differences in solution costs among the four ACO variants are minor, once their parameters are fine tuned.

A.1 Effectiveness of `2.5-opt-EEais` in pACS

In this section, we show that the adoption of `2.5-opt-EEais` instead of `1-shift` as a subsidiary solution improvement procedure significantly improves the effectiveness of pACS. For a detailed explanation of `2.5-opt-EEais`, we refer the reader to Chapter 5. We denote pACS+2.5-opt-EEais the algorithm obtained by combining pACS with

Table A.1: Comparison of the average cost obtained by pACS+2.5-opt-EEais and pACS+1-shift over 30 independent runs on instance `rat783` for each probability value p. Typographic conventions are the same as in Table 6.2.

p	Difference [95% CI]
0.050	*+0.24* [+0.01, +0.46]
0.075	**−2.02** [−2.51, −1.53]
0.100	**−3.03** [−3.43, −2.63]
0.150	**−5.21** [−5.78, −4.64]
0.175	**−5.80** [−6.27, −5.33]
0.200	**−6.21** [−6.66, −5.77]
0.300	**−9.40** [−9.92, −8.88]
0.400	**−10.76** [−11.45, −10.07]
0.500	**−12.18** [−12.76, −11.59]

`2.5-opt-EEais`. Concerning parameter values, `2.5-opt-EEais` uses the values given in Table 6.1 on page 105 and pACS adopts the parameter values suggested by Bianchi (2006) and Bianchi and Gambardella (2007).

pACS+1-shift and pACS+2.5-opt-EEais are evaluated on the homogeneous PTSP instances used by Bianchi (2006), which are obtained by assigning a same probability value to each node for TSPLIB instances, `ch150`, `d198`, `lin318`, `att532`, and `rat783`. The algorithms were implemented in `C` and compiled with `gcc`, version 3.3. Experiments were carried out on AMD Opteron^{TM}244 1.75 GHz processors with 1 MB L2-Cache and 2 GB RAM, running under Rocks Cluster GNU/Linux. We used the stopping criterion suggested by Bianchi and Gambardella (2007) and by Bianchi (2006), where each algorithm is allowed to run for a computation time of $n^2/100$ CPU seconds. The computational results obtained on the instance `rat783` are shown in Table A.1 and Figure A.1.

The results show that the adoption of `2.5-opt-EEais` in pACS is indeed very effective. The average cost of the solutions found by pACS+2.5-opt-EEais is between 2.02% to 12.18% less than those of pACS+1-shift and the observed difference is significant according to Student's t-test. An exception is for $p = 0.050$, where pACS+1-shift obtains an average solution cost that is 0.24% less than that of pACS+2.5-opt-EEais. The general trends of the experimental results obtained on the other instance sizes are similar. For the complete set of results and for the absolute values, we refer the reader to the following supplementary page (Balaprakash et al., 2008c):

`http://iridia.ulb.ac.be/supp/IridiaSupp2008-018/`

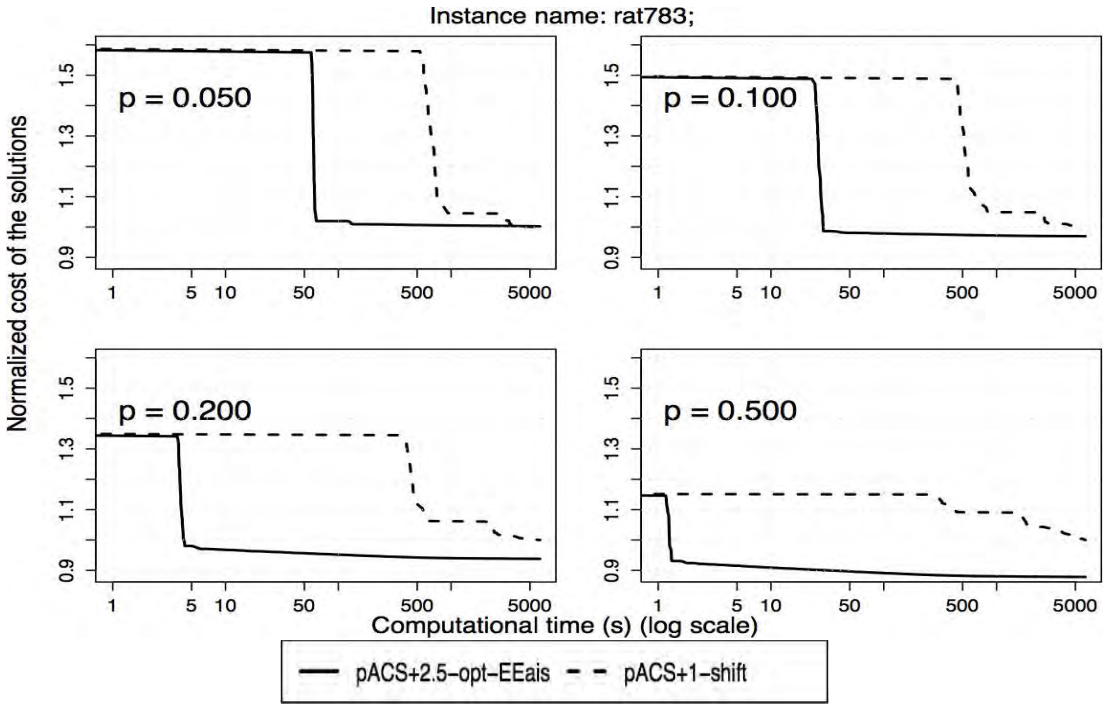


Figure A.1: Experimental results on the instance `rat783`. The plots represent the development of the solution cost over time for pACS+2.5-opt-EEais and pACS+1-shift. The obtained solution costs of the two algorithms are normalized by the final solution cost reached by pACS+1-shift. The normalization is done on a run-by-run basis for 30 runs; the normalized solution cost is then aggregated.

A.2 Estimation-based ant colony system

In order to design a complete estimation-based ACS, that is, to make the solution evaluation approach of ACS coherent with that of the underlying iterative improvement algorithm, we modified pACS+2.5-opt-EEais in such a way that the solution costs are evaluated using Equation 4.2 instead of Equation 4.1. As discussed in Section 6.1 of Chapter 6, for each solution x^i, an *unbiased* estimator $\hat{F}_{M_i}(x^i)$ of $F(x^i)$ is obtained through M_i independent realizations of $\boldsymbol{\omega}$. Moreover, we use the method of common random numbers and the adaptive sample size procedure, ANOVA-Race discussed in Section 6.1 of Chapter 6. We denote the complete estimation-based algorithm ACS-EE, where `EE` stands for empirical estimation.

We evaluate ACS-EE and pACS+2.5-opt-EEais on three groups of homogeneous PTSP instances: (i) instances derived from TSPLIB instances (`ch150`, `d198`, `lin318`, `att532`, and `rat783`); (ii) clustered instances of size 1000; (iii) uniform instances of

size 1000. The second and third groups of instances are generated afresh using the DIMACS instance generators. All these instances use probability values as detailed in Section A.1. For an instance size n, Bianchi (2006) and Bianchi and Gambardella (2007) used $n^2/100$ CPU seconds as a stopping criterion, which allowed pACS+1-shift to perform more than five iterations. Such a high computation time is needed because the computational complexity of `1-shift` is very high. Since `2.5-opt-EEais` is between two and three orders of magnitude faster than `1-shift`—see Chapter 4, we study the algorithms under $n^2/10000$ and $n^2/1000$ CPU seconds. The adoption of $n^2/100000$ CPU seconds is not insightful because it does not allow the algorithms to perform more than five iterations. Note that Equation 4.1 is used for the post-evaluation of the best-so-far solutions found by ACS-EE. We present the results obtained on clustered instances of size 1000. The trend of the results obtained on TSPLIB and uniform instances is similar to that of clustered instances. A detailed presentation of these results is available in Balaprakash et al. (2008c).

The parameters of the adaptive sampling procedure are fixed *a priori*: M_{min} is set to 5 and M is set to 1000. The null hypothesis is rejected at a significance level of 0.05. ACS-EE adopts the same parameter values as pACS+2.5-opt-EEais. ACS-EE uses a same set of realizations for all iterations. However, the realizations are selected randomly from the given set for each iteration. Note that the implementation of ACS-EE and pACS+2.5-opt-EEais is based on ACOTSP (Stützle, 2002) and the two algorithms differ only in the solution evaluation procedure.

The computational results in Table A.2 show that for both stopping criteria the two algorithms provide similar results. With 95% confidence, under the current experimental settings, we can state that should ever the expected cost of solutions found by ACS-EE be higher than the one of those found by pACS+2.5-opt-EEais, the difference between the expected costs would be at most 0.74% and 0.49% under 100 CPU seconds and 1000 CPU seconds, respectively.

We also tested the algorithms on instances with p>0.5, where we found that ACS-EE is significantly better than pACS+2.5-opt-EEais. This can be explained as follows: instances with high probability values have low variance with respect to the mean. In this case, ANOVA-Race needs only few realizations to select the best solution. This allows ACS-EE to perform more iterations when compared to pACS+2.5-opt-EEais. Consequently, ACS-EE obtains higher quality solutions.

Note that the results presented in this section and in Section A.1 suggest different conclusions from those presented in Bianchi (2006) and Bianchi and Gambardella (2007), where it was shown that in pACS+1-shift the adoption of an estimation-based

Table A.2: Comparison of the average cost obtained by ACS-EE and pACS+2.5-opt-EEais over 30 clustered instances of size 1000 for 100 and 1000 CPU seconds. Typographic conventions are the same as in Table 6.2.

100 CPU seconds		1000 CPU seconds	
	ACS-EE vs. pACS+2.5-opt-EEais		ACS-EE vs. pACS+2.5-opt-EEais
p	Difference [95% CI]	p	Difference [95% CI]
0.050	-0.54 $[-1.25, +0.17]$	0.050	$+0.12$ $[-0.25, +0.49]$
0.075	$+0.14$ $[-0.46, +0.74]$	0.075	$+0.02$ $[-0.05, +0.10]$
0.100	$+0.04$ $[-0.27, +0.36]$	0.100	$+0.02$ $[-0.04, +0.08]$
0.125	$+0.03$ $[-0.22, +0.28]$	0.125	$+0.04$ $[-0.05, +0.13]$
0.150	-0.06 $[-0.39, +0.27]$	0.150	$+0.04$ $[-0.06, +0.15]$
0.175	$+0.13$ $[-0.12, +0.37]$	0.175	$+0.11$ $[-0.03, +0.26]$
0.200	-0.08 $[-0.39, +0.23]$	0.200	$+0.04$ $[-0.11, +0.20]$
0.300	-0.19 $[-0.44, +0.05]$	0.300	-0.03 $[-0.14, +0.08]$
0.400	-0.00 $[-0.21, +0.21]$	0.400	-0.07 $[-0.15, +0.02]$
0.500	-0.16 $[-0.41, +0.10]$	0.500	-0.05 $[-0.15, +0.05]$
0.600	$\mathbf{-0.32}$ $[-0.56, -0.09]$	0.600	-0.02 $[-0.12, +0.08]$
0.700	$\mathbf{-0.49}$ $[-0.76, -0.21]$	0.700	-0.09 $[-0.26, +0.08]$
0.800	$\mathbf{-0.62}$ $[-0.81, -0.43]$	0.800	$\mathbf{-0.21}$ $[-0.31, -0.10]$
0.900	$\mathbf{-0.99}$ $[-1.22, -0.77]$	0.900	$\mathbf{-0.15}$ $[-0.29, -0.02]$

approach is less effective than the analytical computation approach. This difference in results can be put down to our more advanced estimation approach and our effective estimation-based local search algorithm.

A.3 Comparison between estimation-based ACO algorithms

So far ACS is widely adopted to tackle the PTSP (Bianchi et al., 2002a,b; Branke and Guntsch, 2004; Bianchi, 2006; Bianchi and Gambardella, 2007). Although ACS is a high performing ACO algorithm, we cannot rule out other existing ACO algorithms as promising ones for the PTSP. This is due to the fact that there is no theoretical justification or empirical evidence in the PTSP literature suggesting that ACS is the best choice. We address this issue by comparing ACS with the following three ACO algorithms: $\mathcal{MAX}$–$\mathcal{MIN}$ ant system (MMAS) (Stützle and Hoos, 2000), rank-based ant system (RAS) (Bullnheimer et al., 1999), and best-worst ant system (BWAS) (Cordón et al., 2002).

In all three algorithms, m ants construct solutions using only the random proportional rule and they differ from ACS with respect to the pheromone update procedure. In MMAS, only the iteration-best or best-so-far ant updates the pheromone trail τ_{ij} associated with edge $\langle i, j\rangle$; the update rule used is: $\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{best}$, where ρ is a parameter and $\Delta\tau_{ij}^{best}$ is set to either $1/C^{best}$—only when the edge $\langle i, j\rangle$ belongs to the chosen best solution—or 0. The value of C^{best} is equal to the cost of the iteration-best or best-so-far solution depending on which of the two is chosen. The pheromone values are limited within a maximum and a minimum value in order to reduce the risk of search stagnation; in case of search stagnation, the search is restarted by re-initializing the pheromone values. In RAS, at each iteration, from m solutions only the $(w-1)$ best ranked solutions and the best-so-far solution are allowed to update the pheromone values using the following equation: $\tau_{ij} = \tau_{ij} + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{best}$, where r is the rank of the solution obtained by sorting m solutions by increasing cost, $\Delta\tau_{ij}^{r} = 1/C^{r}$, and $\Delta\tau_{ij}^{best} = 1/C^{best}$ if edge $\langle i, j\rangle$ belongs to the best-so-far solution; C^{r} and C^{best} are the cost of the solution with rank r and the cost of the best-so-far solution, respectively. In BWAS, only the best-so-far solution is allowed to update the pheromone values; the pheromone values of the edges that belong to the worst ant but not to the best-so-far solution are reduced. To avoid premature convergence, BWAS uses pheromone re-initialization and pheromone mutation.

All the aforementioned algorithms are extended to solve the PTSP by using ANOVA-Race to evaluate the solution costs and by using `2.5-opt-EEais` as the underlying solution improvement procedure. We denote them MMAS-EE, RAS-EE, and BWAS-EE.

Similar to ACS-EE, the implementations of MMAS-EE, RAS-EE, and BWAS-EE were based on ACOTSP (Stützle, 2002). We evaluate all the algorithms on TSPLIB instances (`ch150`, `d198`, `lin318`, `att532`, and `rat783`), uniform and clustered instances of size 1000. We allowed each algorithm to run for $n^2/10000$ and $n^2/1000$ CPU seconds. Concerning the parameter values of each algorithm, we use two sets of values: default parameter values and tuned parameter values. We present the empirical results in the following three sections.

A.3.1 Experiments with default parameter values

The default parameter values for each algorithm are chosen reasonably close to the values proposed in the ACO literature for the TSP (Dorigo and Stützle, 2004; Bullnheimer et al., 1999; Cordón et al., 2002): in all the algorithms m, α, and β are set to 10, 1.0, and 2.0, respectively; in ACS-EE, ρ and q_0 are set to 0.1 and 0.98, respectively; in

MMAS-EE, ρ is set to 0.2; in RAS-EE, ρ and w are set to 0.5 and 6, respectively; in BWAS-EE, ρ is set to 0.2. In all algorithms the initial value τ_0 of the pheromone is set to a value inversely proportional to C^{nn}, where C^{nn} is the TSP cost of the nearest neighbor solution: in ACS-EE τ_0 is set to $1/(n \times C^{nn})$, while in the other three algorithms τ_0 is set to $1/(\rho \times C^{nn})$. We denote the algorithms that adopt the default parameter values as ACS-EE(d) MMAS-EE(d), RAS-EE(d), and BWAS-EE(d).

The results obtained on clustered instances of size 1000 are shown in Table A.3. For most probability levels, ACS-EE(d) is better than other algorithms: the average cost of ACS-EE(d) is between 0.42% and 3.91% and between 0.14% and 3.90% less than that of other algorithms for 100 and 1000 CPU seconds, respectively. Most of the differences that have been observed are statistically significant according to the t-test. The results obtained on TSPLIB and uniform instances of size 1000 exhibit a similar trend. The complete results can be inspected in Balaprakash et al. (2008c).

A.3.2 Comparison between the algorithms with tuned and default values

The parameter values of each algorithm are tuned on clustered instances of size 1000 for 100 and 1000 CPU seconds in the same way as described in Section A.1 using Iterative F-Race. The selected values are shown in Table A.4. Note that we use the same parameter values for each algorithm on TSPLIB and uniform instances. We denote the algorithms that use the fine tuned parameter values as ACS-EE(t), MMAS-EE(t), RAS-EE(t), and BWAS-EE(t).

The results from Table A.5 show that, as expected, the adoption of tuned parameter values allows each algorithm to achieve much better results. MMAS-EE(t), RAS-EE(t), and BWAS-EE(t) profit much more from tuning than ACS-EE(t) does. For 100 CPU seconds, the observed improvements are very large and are up to 8.63%. For 1000 CPU seconds, the improvement is up to 3.53%.

Table A.3: Comparison of the average cost obtained by ACS-EE(d), MMAS-EE(d), RAS-EE(d), and BWAS-EE(d) over 50 clustered instances of size 1000 for 100 and 1000 CPU seconds. Typographic conventions are the same as in Table 6.2.

	100 CPU seconds		
	ACS-EE(d) vs. MMAS-EE(d)	ACS-EE(d) vs. RAS-EE(d)	ACS-EE(d) vs. BWAS-EE(d)
p	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]
0.050	*+2.67* [+0.61, +4.73]	+1.43 [−0.58, +3.45]	+0.75 [−1.39, +2.88]
0.075	**−2.84** [−3.42, −2.26]	**−3.42** [−4.03, −2.80]	**−3.14** [−3.65, −2.64]
0.100	**−3.91** [−4.36, −3.45]	**−1.64** [−2.05, −1.23]	**−1.06** [−1.73, −0.38]
0.150	**−0.70** [−0.88, −0.51]	**−0.61** [−0.84, −0.37]	−0.12 [−0.29, +0.05]
0.175	**−1.03** [−1.24, −0.83]	**−0.99** [−1.18, −0.80]	**−0.42** [−0.62, −0.21]
0.200	**−1.16** [−1.36, −0.97]	**−1.08** [−1.26, −0.90]	**−0.52** [−0.69, −0.36]
0.300	**−2.25** [−2.45, −2.04]	**−2.02** [−2.16, −1.87]	**−1.17** [−1.37, −0.98]
0.400	**−3.11** [−3.32, −2.90]	**−2.88** [−3.07, −2.70]	**−1.59** [−1.77, −1.40]
0.500	**−3.32** [−3.53, −3.11]	**−3.29** [−3.48, −3.11]	**−1.73** [−1.92, −1.53]

	1000 CPU seconds		
	ACS-EE(d) vs. MMAS-EE(d)	ACS-EE(d) vs. RAS-EE(d)	ACS-EE(d) vs. BWAS-EE(d)
p	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]
0.050	**−1.89** [−2.22, −1.56]	**−1.08** [−1.43, −0.73]	**−0.91** [−1.18, −0.63]
0.075	**−1.80** [−2.01, −1.60]	**−1.37** [−1.52, −1.22]	**−0.83** [−1.01, −0.64]
0.100	**−0.75** [−0.85, −0.65]	**−0.70** [−0.79, −0.61]	**−0.38** [−0.46, −0.29]
0.150	**−0.79** [−0.87, −0.71]	**−1.26** [−1.36, −1.16]	**−0.57** [−0.66, −0.48]
0.175	**−0.92** [−1.03, −0.82]	**−1.74** [−1.83, −1.64]	**−0.55** [−0.64, −0.47]
0.200	**−0.96** [−1.06, −0.86]	**−2.12** [−2.24, −2.00]	**−0.45** [−0.54, −0.36]
0.300	**−0.62** [−0.73, −0.51]	**−3.19** [−3.31, −3.07]	**−0.28** [−0.38, −0.17]
0.400	**−0.23** [−0.33, −0.13]	**−3.61** [−3.77, −3.45]	**−0.29** [−0.40, −0.18]
0.500	**−0.14** [−0.25, −0.03]	**−3.90** [−4.03, −3.77]	**−0.41** [−0.50, −0.31]

Table A.4: Fine tuned parameters values

100 CPU seconds

algorithm	parameters	range	selected value		
			Class-I	Class-II	Class-III
ACS-EE	m	[3, 15]	5	4	11
	β	[0.0, 5.0]	3.3	0.16	1.0
	ρ	[0.001, 1.0]	0.75	0.84	1.0
	q_0	[0.0, 1.0]	0.84	1.0	0.99
MMAS-EE	m	[3, 15]	5	4	15
	α	[0.001, 1.5]	1.4	1.3	0.99
	β	[0.0, 5.0]	3.2	0.97	2.1
	ρ	[0.001, 1.0]	1.0	1.0	0.97
RAS-EE	m	[3, 15]	3	3	6
	α	[0.001, 1.5]	0.33	1.1	0.71
	β	[0.0, 5.0]	5.0	2.6	2.1
	ρ	[0.001, 1.0]	1.0	0.94	0.83
	w	[1, 10]	1	1	1
BWAS-EE	m	[3, 15]	3	4	4
	α	[0.001, 1.5]	0.99	1.4	0.89
	β	[0.0, 5.0]	3.1	2.9	2.3
	ρ	[0.001, 1.0]	0.95	0.97	0.66

1000 CPU seconds

algorithm	parameters	range	selected value		
			Class-I	Class-II	Class-III
ACS-EE	m	[3, 15]	4	3	5
	β	[0.0, 5.0]	0.05	0.85	3.7
	ρ	[0.001, 1.0]	0.67	0.079	0.82
	q_0	[0.0, 1.0]	0.99	0.99	0.96
MMAS-EE	m	[3, 15]	8	15	6
	α	[0.001, 1.5]	1.5	1.2	1.1
	β	[0.0, 5.0]	1.6	1.9	0.95
	ρ	[0.001, 1.0]	0.99	0.98	0.62
RAS-EE	m	[3, 15]	10	6	11
	α	[0.001, 1.5]	1.2	1.5	1.4
	β	[0.0, 5.0]	0.85	2.1	2.7
	ρ	[0.001, 1.0]	1.0	0.57	0.37
	w	[1, 10]	1	1	1
BWAS-EE	m	[3, 15]	5	10	6
	α	[0.001, 1.5]	0.6	1.1	0.9
	β	[0.0, 5.0]	2.7	0.09	2.4
	ρ	[0.001, 1.0]	0.99	0.85	0.27

Table A.5: Comparison of the average cost obtained by the algorithms with tuned values and by the algorithms with default values over 50 clustered instances of size 1000 for 100 and 1000 CPU seconds. Typographic conventions are the same as in Table 6.2.

100 CPU seconds

	ACS-EE(t) vs. ACS-EE(d)	MMAS-EE(t) vs. MMAS-EE(d)	RAS-EE(t) vs. RAS-EE(d)	BWAS-EE(t) vs. BWAS-EE(d)
p	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]
0.050	**−8.05** [−9.71, −6.39]	**−5.03** [−6.24, −3.82]	**−8.63** [−10.09, −7.17]	**−7.87** [−9.47, −6.28]
0.075	**−2.18** [−2.73, −1.63]	**−5.84** [−6.37, −5.30]	**−7.00** [−7.58, −6.41]	**−5.71** [−6.13, −5.28]
0.100	−0.21 [−0.49, +0.07]	**−4.81** [−5.25, −4.38]	**−2.80** [−3.14, −2.45]	**−1.74** [−2.48, −1.00]
0.150	**−1.06** [−1.28, −0.84]	**−1.75** [−1.92, −1.58]	**−1.75** [−1.95, −1.56]	**−1.13** [−1.29, −0.98]
0.175	**−0.97** [−1.17, −0.77]	**−2.06** [−2.24, −1.88]	**−2.05** [−2.17, −1.92]	**−1.26** [−1.41, −1.12]
0.200	**−1.14** [−1.29, −0.99]	**−2.27** [−2.47, −2.07]	**−2.23** [−2.42, −2.03]	**−1.48** [−1.64, −1.32]
0.300	**−0.66** [−0.82, −0.50]	**−2.78** [−2.97, −2.59]	**−2.65** [−2.83, −2.48]	**−1.58** [−1.77, −1.39]
0.400	**−0.44** [−0.61, −0.27]	**−3.28** [−3.49, −3.08]	**−3.24** [−3.43, −3.05]	**−1.60** [−1.80, −1.40]
0.500	−0.05 [−0.18, +0.07]	**−3.19** [−3.39, −2.99]	**−3.10** [−3.30, −2.90]	**−1.16** [−1.36, −0.95]

1000 CPU seconds

	ACS-EE(t) vs. ACS-EE(d)	MMAS-EE(t) vs. MMAS-EE(d)	RAS-EE(t) vs. RAS-EE(d)	BWAS-EE(t) vs. BWAS-EE(d)
p	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]
0.050	**−1.14** [−1.42, −0.85]	**−2.64** [−2.94, −2.34]	**−2.82** [−3.06, −2.58]	**−1.65** [−1.90, −1.41]
0.075	**−0.30** [−0.36, −0.23]	**−2.08** [−2.28, −1.87]	**−1.67** [−1.81, −1.53]	**−1.09** [−1.30, −0.89]
0.100	**−0.22** [−0.26, −0.17]	**−0.95** [−1.05, −0.84]	**−0.92** [−1.02, −0.82]	**−0.58** [−0.67, −0.49]
0.150	**−0.16** [−0.22, −0.09]	**−0.94** [−1.04, −0.84]	**−1.44** [−1.53, −1.34]	**−0.68** [−0.78, −0.58]
0.175	−0.04 [−0.11, +0.04]	**−1.06** [−1.16, −0.95]	**−1.83** [−1.92, −1.75]	**−0.61** [−0.69, −0.53]
0.200	**−0.09** [−0.17, −0.00]	**−1.07** [−1.15, −0.98]	**−2.13** [−2.26, −2.01]	**−0.44** [−0.54, −0.34]
0.300	*+0.19* [+0.08, +0.30]	**−0.72** [−0.82, −0.62]	**−3.19** [−3.29, −3.08]	**−0.11** [−0.21, −0.00]
0.400	+0.07 [−0.02, +0.16]	**−0.23** [−0.32, −0.14]	**−3.39** [−3.57, −3.22]	−0.07 [−0.17, +0.04]
0.500	+0.10 [−0.00, +0.21]	−0.09 [−0.18, +0.01]	**−3.53** [−3.66, −3.40]	−0.10 [−0.22, +0.02]

A.3.3 Comparison between the algorithms with tuned parameter values

The computational results of the four algorithms that adopt the tuned parameter values on clustered instances of size 1000 are given in Table A.6. For the absolute values, we refer the reader to Balaprakash et al. (2008c). From the results, we cannot identify a clear winner among the considered algorithms. For 100 CPU seconds, with a confidence level of 95%, under the current experimental setting, we can state that should ever the expected cost of the solutions found by MMAS-EE(t), RAS-EE(t), and BWAS-EE(t) be larger than those found by ACS-EE(t), the difference would be at most 1.46%, 2.76% and 1.26%, respectively. For 1000 CPU seconds, the aforementioned differences would be at most 0.37%, 0.83% and 0.11%, respectively. There are a few exceptions, where the differences are significant but rather small: for 100 CPU seconds, the maximum observed difference is less than 1% (except for $p = 0.050$ and $p = 0.075$, where the average cost of RAS-EE(t) is 2.08% and 1.59% less than that of ACS-EE(t), respectively) and for 1000 CPU seconds it is less than 0.7%.

From the absolute values reported in Balaprakash et al. (2008c), we observed that for 1000 CPU seconds all the algorithms obtain average solution costs that are smaller than that of 100 CPU seconds; the improvements for an order of magnitude increase in the computation time are in the range of 0.4% to 2.1% except for $p = 0.050$, where the improvements are between 2.9% to 4.5%.

In Tables A.7 and A.8, we report some exemplary results obtained on uniform instances of size 1000 and on TSPLIB instance `rat783`. The conclusions of the comparison are similar to the one of clustered instances of size 1000.

In order to further assess the solution costs achieved by the algorithms for a very long computation time, we allowed the algorithms to run for 10000 CPU seconds, as suggested by Bianchi (2006), Bianchi and Gambardella (2007), on clustered and uniform instances of size 1000. The parameter values of each algorithm are the same as that of 1000 CPU seconds. The results are shown in Tables A.6 and A.7. The general trend is similar to that of shorter computation times: there is no clear winner among the considered algorithms.

A.4 Summary

The main contribution of this appendix is the development and the empirical analysis of estimation-based ACO algorithms for the PTSP. We used the best performing analyti-

Table A.6: Comparison of the average cost obtained by ACS-EE(t), MMAS-EE(t), RAS-EE(t), and BWAS-EE(t) over 50 clustered instances of size 1000 for 100, 1000, and 10000 CPU seconds. Typographic conventions are the same as in Table 6.2.

	100 CPU seconds		
	ACS-EE(t) vs. MMAS-EE(t)	ACS-EE(t) vs. RAS-EE(t)	ACS-EE(t) vs. BWAS-EE(t)
p	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]
0.050	−0.60 [−1.50, +0.30]	*+2.08* [+1.39, +2.76]	+0.55 [−0.15, +1.26]
0.075	*+0.94* [+0.43, +1.46]	*+1.59* [+1.18, +2.00]	+0.48 [−0.02, +0.99]
0.100	*+0.74* [+0.52, +0.97]	*+0.98* [+0.81, +1.14]	*+0.49* [+0.30, +0.68]
0.125	+0.03 [−0.10, +0.16]	**−0.15** [−0.28, −0.02]	**−0.17** [−0.30, −0.03]
0.150	+0.00 [−0.16, +0.16]	+0.10 [−0.05, +0.24]	−0.04 [−0.21, +0.13]
0.175	+0.07 [−0.06, +0.20]	+0.10 [−0.02, +0.22]	−0.12 [−0.26, +0.01]
0.200	−0.02 [−0.18, +0.15]	+0.02 [−0.13, +0.18]	**−0.18** [−0.31, −0.05]
0.300	−0.12 [−0.26, +0.02]	−0.01 [−0.17, +0.14]	**−0.26** [−0.42, −0.10]
0.400	**−0.27** [−0.41, −0.12]	−0.07 [−0.23, +0.08]	**−0.43** [−0.57, −0.28]
0.500	**−0.18** [−0.33, −0.04]	**−0.25** [−0.41, −0.10]	**−0.63** [−0.76, −0.50]

	1000 CPU seconds		
	ACS-EE(t) vs. MMAS-EE(t)	ACS-EE(t) vs. RAS-EE(t)	ACS-EE(t) vs. BWAS-EE(t)
p	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]
0.050	**−0.38** [−0.59, −0.16]	*+0.64* [+0.44, +0.83]	**−0.38** [−0.60, −0.17]
0.075	−0.02 [−0.04, +0.00]	+0.01 [−0.05, +0.06]	**−0.03** [−0.06, −0.00]
0.100	−0.01 [−0.04, +0.02]	+0.00 [−0.04, +0.05]	−0.01 [−0.04, +0.02]
0.150	−0.00 [−0.06, +0.05]	+0.02 [−0.04, +0.08]	−0.04 [−0.09, +0.01]
0.175	*+0.10* [+0.02, +0.17]	+0.06 [−0.02, +0.14]	+0.02 [−0.05, +0.09]
0.200	+0.02 [−0.05, +0.10]	−0.07 [−0.16, +0.02]	**−0.10** [−0.17, −0.02]
0.300	*+0.29* [+0.21, +0.37]	*+0.19* [+0.11, +0.27]	+0.02 [−0.08, +0.11]
0.400	+0.07 [−0.02, +0.16]	**−0.15** [−0.25, −0.05]	**−0.15** [−0.23, −0.07]
0.500	+0.05 [−0.05, +0.15]	**−0.28** [−0.38, −0.18]	**−0.20** [−0.29, −0.10]

	10000 CPU seconds		
	ACS-EE(t) vs. MMAS-EE(t)	ACS-EE(t) vs. RAS-EE(t)	ACS-EE(t) vs. BWAS-EE(t)
p	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]
0.050	**−0.16** [−0.26, −0.07]	*+0.15* [+0.10, +0.21]	*+0.09* [+0.05, +0.13]
0.075	**−0.01** [−0.03, −0.00]	+0.02 [−0.00, +0.03]	*+0.02* [+0.01, +0.03]
0.100	**−0.03** [−0.05, −0.01]	−0.04 [−0.11, +0.03]	−0.02 [−0.04, +0.01]
0.150	−0.04 [−0.08, +0.01]	+0.02 [−0.01, +0.05]	−0.05 [−0.11, +0.01]
0.175	−0.07 [−0.15, +0.01]	−0.00 [−0.07, +0.07]	−0.04 [−0.12, +0.04]
0.200	+0.00 [−0.08, +0.09]	−0.05 [−0.14, +0.03]	−0.01 [−0.12, +0.10]
0.300	+0.00 [−0.07, +0.07]	**−0.15** [−0.24, −0.06]	**−0.09** [−0.17, −0.02]
0.400	*+0.09* [+0.01, +0.17]	**−0.33** [−0.47, −0.20]	−0.03 [−0.13, +0.06]
0.500	+0.02 [−0.09, +0.12]	**−0.48** [−0.62, −0.35]	−0.06 [−0.13, +0.02]

Table A.7: Comparison of the average cost obtained by ACS-EE(t), MMAS-EE(t), RAS-EE(t), and BWAS-EE(t) over 50 uniform instances of size 1000 for 100, 1000, and 10000 CPU seconds. Typographic conventions are the same as in Table 6.2.

100 CPU seconds

	ACS-EE(t) vs. MMAS-EE(t)	ACS-EE(t) vs. RAS-EE(t)	ACS-EE(t) vs. BWAS-EE(t)
p	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]
0.050	−0.18 [−0.68, +0.31]	*+1.39* [+0.99, +1.80]	+0.34 [−0.03, +0.70]
0.075	*+0.44* [+0.28, +0.60]	*+0.81* [+0.66, +0.96]	*+0.27* [+0.12, +0.42]
0.100	*+0.81* [+0.66, +0.96]	*+0.93* [+0.74, +1.13]	*+0.66* [+0.50, +0.82]
0.150	+0.09 [−0.06, +0.25]	+0.13 [−0.04, +0.30]	−0.06 [−0.24, +0.13]
0.175	+0.00 [−0.19, +0.19]	−0.03 [−0.20, +0.13]	−0.16 [−0.35, +0.02]
0.200	+0.15 [−0.02, +0.32]	+0.14 [−0.06, +0.34]	**−0.35** [−0.60, −0.11]
0.300	+0.03 [−0.15, +0.20]	+0.04 [−0.11, +0.19]	**−0.41** [−0.64, −0.19]
0.400	−0.08 [−0.26, +0.09]	**−0.23** [−0.41, −0.05]	**−0.64** [−0.81, −0.47]
0.500	+0.04 [−0.13, +0.21]	**−0.53** [−0.70, −0.37]	**−0.75** [−0.92, −0.58]

1000 CPU seconds

	ACS-EE(t) vs. MMAS-EE(t)	ACS-EE(t) vs. RAS-EE(t)	ACS-EE(t) vs. BWAS-EE(t)
p	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]
0.050	−0.12 [−0.29, +0.04]	*+0.28* [+0.11, +0.45]	−0.07 [−0.28, +0.14]
0.075	−0.04 [−0.13, +0.05]	−0.01 [−0.10, +0.08]	*+0.06* [+0.00, +0.12]
0.100	+0.04 [−0.03, +0.11]	−0.03 [−0.15, +0.09]	*+0.12* [+0.03, +0.21]
0.150	+0.09 [−0.01, +0.20]	+0.00 [−0.12, +0.12]	−0.08 [−0.20, +0.04]
0.175	+0.03 [−0.05, +0.11]	−0.06 [−0.19, +0.08]	−0.07 [−0.19, +0.05]
0.200	+0.04 [−0.09, +0.18]	**−0.12** [−0.24, −0.01]	−0.10 [−0.22, +0.02]
0.300	+0.11 [−0.01, +0.23]	−0.08 [−0.22, +0.06]	**−0.23** [−0.34, −0.13]
0.400	+0.01 [−0.11, +0.13]	**−0.45** [−0.58, −0.32]	**−0.35** [−0.46, −0.24]
0.500	+0.05 [−0.06, +0.17]	**−0.60** [−0.74, −0.46]	**−0.33** [−0.46, −0.20]

10000 CPU seconds

	ACS-EE(t) vs. MMAS-EE(t)	ACS-EE(t) vs. RAS-EE(t)	ACS-EE(t) vs. BWAS-EE(t)
p	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]
0.050	**−0.21** [−0.34, −0.08]	+0.09 [−0.01, +0.18]	+0.05 [−0.08, +0.18]
0.075	+0.02 [−0.07, +0.11]	+0.06 [−0.05, +0.17]	+0.06 [−0.03, +0.16]
0.100	+0.04 [−0.10, +0.19]	−0.08 [−0.27, +0.12]	+0.02 [−0.16, +0.19]
0.150	−0.03 [−0.24, +0.19]	−0.10 [−0.23, +0.04]	+0.00 [−0.15, +0.16]
0.175	*+0.20* [+0.06, +0.33]	−0.04 [−0.22, +0.14]	+0.13 [−0.01, +0.28]
0.200	+0.13 [−0.07, +0.32]	−0.04 [−0.25, +0.17]	+0.02 [−0.17, +0.21]
0.300	+0.08 [−0.08, +0.25]	**−0.37** [−0.57, −0.16]	**−0.16** [−0.31, −0.01]
0.400	*+0.23* [+0.10, +0.37]	**−0.39** [−0.59, −0.20]	+0.03 [−0.12, +0.18]
0.500	+0.08 [−0.09, +0.24]	**−0.81** [−1.01, −0.60]	−0.05 [−0.20, +0.09]

Table A.8: Comparison of the average cost obtained by ACS-EE(t), MMAS-EE(t), RAS-EE(t), and BWAS-EE(t) over 30 independent runs on instance `rat783` for $n^2/10000 = 61$ and $n^2/1000 = 613$ CPU seconds. Typographic conventions are the same as in Table 6.2.

	61 CPU seconds		
	ACS-EE(t) vs. MMAS-EE(t)	ACS-EE(t) vs. RAS-EE(t)	ACS-EE(t) vs. BWAS-EE(t)
p	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]
0.050	−0.06 [−0.24, +0.13]	−0.13 [−0.36, +0.11]	−0.20 [−0.44, +0.03]
0.075	+0.05 [−0.10, +0.20]	+0.03 [−0.14, +0.21]	−0.07 [−0.22, +0.08]
0.100	+0.00 [−0.16, +0.16]	−0.12 [−0.36, +0.11]	−0.11 [−0.27, +0.05]
0.150	−0.09 [−0.28, +0.10]	−0.02 [−0.18, +0.14]	−0.08 [−0.25, +0.09]
0.175	−0.03 [−0.17, +0.11]	+0.09 [−0.04, +0.22]	−0.05 [−0.21, +0.11]
0.200	−0.05 [−0.21, +0.12]	−0.07 [−0.22, +0.09]	−0.18 [−0.36, +0.01]
0.300	+0.17 [−0.03, +0.36]	*+0.18* [+0.05, +0.30]	−0.03 [−0.18, +0.12]
0.400	−0.10 [−0.26, +0.06]	**−0.31** [−0.47, −0.15]	**−0.31** [−0.45, −0.16]
0.500	+0.03 [−0.13, +0.19]	**−0.40** [−0.59, −0.21]	**−0.25** [−0.40, −0.09]

	613 CPU seconds		
	ACS-EE(t) vs. MMAS-EE(t)	ACS-EE(t) vs. RAS-EE(t)	ACS-EE(t) vs. BWAS-EE(t)
p	Difference [95% CI]	Difference [95% CI]	Difference [95% CI]
0.050	+0.21 [−0.18, +0.59]	−0.07 [−0.52, +0.38]	**−0.57** [−1.09, −0.04]
0.075	−0.10 [−0.30, +0.10]	**−0.49** [−0.73, −0.24]	**−0.34** [−0.62, −0.07]
0.100	*+0.29* [+0.03, +0.56]	−0.24 [−0.57, +0.08]	−0.09 [−0.43, +0.26]
0.150	*+1.09* [+0.70, +1.47]	*+0.88* [+0.54, +1.22]	*+0.88* [+0.52, +1.23]
0.175	*+1.57* [+1.21, +1.94]	*+1.32* [+0.88, +1.76]	*+1.24* [+0.88, +1.60]
0.200	*+1.65* [+1.33, +1.96]	*+1.09* [+0.69, +1.49]	*+1.33* [+1.01, +1.64]
0.300	*+0.96* [+0.73, +1.20]	*+0.62* [+0.37, +0.87]	*+0.31* [+0.01, +0.62]
0.400	+0.06 [−0.27, +0.39]	**−0.40** [−0.68, −0.11]	**−0.69** [−1.00, −0.38]
0.500	**−0.86** [−1.19, −0.53]	**−1.41** [−1.64, −1.18]	**−1.66** [−1.93, −1.40]

cal computation ACO algorithm pACS+1-shift as a starting point. We showed that the adoption of the iterative improvement algorithm `2.5-opt-EEais` allows pACS to obtain a significant improvement in the solution cost. To develop a complete estimation-based ACS, we adopted an estimation-based approach to evaluate the solution costs. Finally, we customized $\mathcal{MAX}$–$\mathcal{MIN}$ ant system, rank-based ant system, and best-worst ant system to solve the PTSP. We showed that all of them can be used to effectively tackle the PTSP provided that their parameter values are fine tuned. The major advantage of the estimation-based approach is that algorithm designers do not require *a priori* knowledge on how to compute the expected cost analytically. This is particularly useful when applying ACO algorithms to complex stochastic combinatorial optimization problems, where it might be very difficult, or even impossible, to derive closed-form expressions.

Appendix B

The Iterative F-Race algorithm

Finding appropriate values for the parameters of an algorithm is a challenging, important, and time consuming task. While typically parameters are tuned by hand, recent studies have shown that automatic tuning procedures can effectively handle this task and often find better parameter settings. `F-Race` has been proposed specifically for this purpose and it has proven to be very effective in a number of cases. `F-Race` is a racing algorithm that starts by considering a number of candidate parameter settings and eliminates inferior ones as soon as enough statistical evidence arises against them. In this appendix, we propose two modifications to the usual way of applying `F-Race` that, on the one hand, make it suitable for tuning tasks with a very large number of initial candidate parameter settings and, on the other hand, allow a significant reduction in the number of function evaluations without any major loss in solution quality. We evaluate the proposed modifications on a number of stochastic local search algorithms and we show their effectiveness.

B.1 Introduction

The full potential of a parameterized algorithm cannot be achieved unless its parameters are fine tuned. Often, practitioners tune the parameters using their personal experience guided by some rules of thumb. Usually, such a procedure is tedious and time consuming and, hence, it is not surprising that some authors say that 90% of the total time needed for developing an algorithm is dedicated to find the right parameter values (Adenso-Diaz and Laguna, 2006). Therefore, an effective automatic tuning procedure is an absolute must by which the computational time and the human intervention required for tuning can be significantly reduced. In fact, the selection of parameter values

that drive heuristics is itself a scientific endeavor and deserves more attention than it has received in the operations research literature (Barr et al., 1995). In this context, few procedures have been proposed in the literature. `F-Race` (Birattari et al., 2002; Birattari, 2004) is one among them and has proven to be successful and useful in a number of tuning tasks (Birattari, 2004; Becker et al., 2005; Chiarandini et al., 2006; Pellegrini and Birattari, 2006).

Inspired by a class of racing algorithms proposed in the machine learning literature, `F-Race` evaluates a given set of parameter configurations sequentially on a number of problem instances. As soon as statistical evidence is obtained that a candidate configuration is worse than at least another one, the inferior candidate is discarded and not considered for further evaluation. In all previously published works using `F-Race`, the initial candidate configurations were obtained through a full factorial design. This design is primarily used to select the best parameter configuration from a relatively small set of promising configurations that the practitioner has already established. Nevertheless, the main difficulty of this design is that, if the practitioner is confronted with a large number of parameters and a wide range of possible values for each parameter, the number of initial configurations becomes quite large. In such cases, the adoption of the full factorial design within `F-Race` can become impractical and computationally prohibitive. In order to tackle this problem, we propose two supplementary procedures to the original `F-Race` approach. The first procedure consists in generating configurations by random sampling. Notwithstanding the simplicity, the empirical results show that this approach can be more effective—in the context of tuning tasks—than the adoption of a full factorial design. However, if the number of parameters is large, this methodology might need a large number of configurations to achieve good results. We alleviate this problem taking inspiration from model-based search techniques (Zlochin et al., 2004) in the second procedure. This procedure uses a probabilistic model defined on the set of all possible parameter configurations and at each iteration, a small set of parameter configurations is generated according to the model. Elite configurations selected by `F-Race` are then used to update the model in order to bias the search around the high quality parameter configurations.

The appendix is organized as follows. In Section B.2, we introduce the proposed supplementary procedures. In Section B.3, we present some empirical results. We discuss some related work in Section B.4, and summarize the appendix in Section B.5.

B.2 Sampling F-Race and Iterative F-Race

For a formal definition of the problem of tuning SLS algorithms, we follow Birattari et al. (2002). The problem is defined as a 6 tuple $\langle \Theta, I, t, P_I, P_c, \mathcal{C} \rangle$, where Θ is the finite set of candidate configurations, I is the possibly infinite set of problem instances, t is a function associating to every instance the computation time that is allocated to it, P_I is a probability measure over the set I, P_C is a probability measure over the set of all possible values for the cost of the best solution found in a run of a configuration $\theta \in \Theta$ on an instance i, $\mathcal{C}(\theta)$ is the criterion that needs to be optimized with respect to θ: the solution of the tuning problem consists in finding a configuration θ^* such that

$$\theta^* = \arg\min_{\theta} \mathcal{C}(\theta). \tag{B.1}$$

Typically, $\mathcal{C}(\theta)$ is an expected value where the expectation is considered with respect to both P_I and P_C. The main advantage of using expectation is that it can be effectively and reliably estimated using Monte Carlo procedures. In this appendix, we focus on the minimization of the expected value of the solution cost and the criterion is given by:

$$\mathcal{C}(\theta) = \mathbb{E}_{I,C}\Big[c(\theta, i)\Big] = \int_I \int_C c_t(\theta, i)\ \mathrm{d}P_C(c_t|\theta, i)\ \mathrm{d}P_I(i), \tag{B.2}$$

where, $c_t(\theta, i)$ is a random variable that represents the cost of the best solution found by running configuration θ on instance i for t seconds. The integration is taken in the Lebesgue sense and the integrals are estimated in a Monte Carlo fashion on the basis of a so-called *tuning set* of instances. It is straightforward to use criteria other than the expected value such as inter-quartile range of the solution cost. In the case of decision problems, the practitioner might be interested in minimizing the run-time of an algorithm, a task that can be handled in a straightforward way by F-Race.

F-Race is inspired by a class of racing algorithms proposed in the machine learning literature for tackling the model selection problem (Maron and Moore, 1994; Moore and Lee, 1994). In F-Race, as in other racing algorithms, a set of given candidate configurations are sequentially evaluated on a number of tuning instances. As soon as sufficient evidence is gathered that a candidate configuration is worse than at least another one, the former is discarded from the race and is not further evaluated. The race terminates when either one single candidate configuration remains, or the available budget of computation time is used. The peculiarity of F-Race compared to other racing algorithms is the adoption of the *Friedman two-way analysis of variance by ranks* (Conover, 1999), a nonparametric statistical test that appears particularly suitable in the context

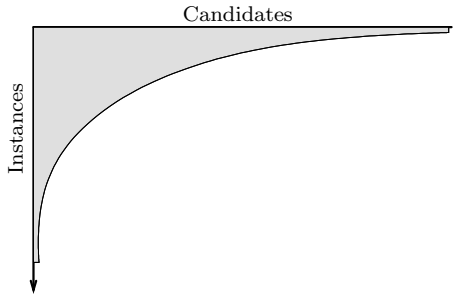


Figure B.1: Visual representation of a typical trace of `F-Race` giving the number of surviving configurations in dependence of the number of instances seen. The *x-axis* represents the number of candidate configurations that are still in the race and the *y-axis* represents the number of instances that has been used for evaluating the configurations. As the evaluation proceeds, `F-Race` focuses more and more on the promising configurations.

of racing algorithms for the tuning task. The progress of the `F-Race` procedure can be graphically illustrated as shown in Figure 1.

The main focus of this appendix is the method by which the initial set of configurations is obtained in `F-Race`: while `F-Race` does not specify how Θ is defined, in most of the studies on `F-Race`, the configurations are defined using a full factorial design (`FFD`). In the simplest case, this is done as follows. Let $\mathrm{M} = \{\mathrm{M}_1, \ldots, \mathrm{M}_d\}$ be the set of parameters that need to be tuned whose ranges are given by (min_k, max_k), for $k = 1, \ldots, d$, where min_k and max_k are the minimum and maximum values of the parameter M_k, respectively. For each element in M, the practitioner has to choose a certain number of values; each possible combination of these parameter values leads to one unique configuration and the set of all possible combinations forms the initial set of configurations. If l_k values are chosen for M_k, then the number of initial configurations is $\prod_{k=1}^{d} l_k$. When each parameter takes l values, then $\prod_{k=1}^{d} l = l^d$; that is, the number of configurations grows exponentially with respect to the number of parameters. As a consequence, even a reasonable number of possible values for each parameter makes the adoption of a full factorial design impractical and computationally prohibitive.

B.2.1 Sampling `F-Race`

A simple way to overcome the shortcomings of `FFD` is sampling. This means that the elements of Θ are sampled according to a given probability measure P_X defined on the space X of parameter values. If *a priori* knowledge is available on the effect of the parameters and on their interactions, this knowledge can be used to shape the probability measure P_X and therefore to suitably bias the sampling of the initial configurations.

On the other hand, if no *a priori* knowledge on the parameter values is available, except the boundary constraints, then each possible value in the available range for each parameter should be given equal probability of being selected in sampling. In this case, P_X is a d-variate uniform distribution, which is factorized by a product of d univariate independent uniform distributions. A sample from the d-variate uniform distribution is a vector corresponding to a configuration θ such that a value x_k in the vector is sampled from the univariate independent uniform distribution parameterized by (min_k, max_k). We call this strategy *random sampling design* (RSD). The F-Race procedure is then applied to the set of sampled configurations. We denote this procedure as RSD/F-Race. It should be noted that the performance of the winning configuration is greatly determined by the number of sampled configurations, N_{max}.

B.2.2 Iterative F-Race

RSD/F-Race can identify promising configurations in the search space. However, finding the best configuration from the promising regions is often a difficult task. In order to address this issue, we propose iterative F-Race (I/F-Race), a supplementary mechanism to the original F-Race approach. It is an iterative procedure in which each iteration consists in first defining a probability measure over the parameter space using promising configurations obtained from the previous iteration, then generating configurations that are distributed according to the newly defined probability measure, and finally applying F-Race on the generated configurations. This approach falls under the general framework of model-based search (Zlochin et al., 2004).

The way in which the probability measure is defined at each iteration plays a crucial role in biasing the search towards regions containing high quality configurations. The main issues in the search bias are the choice of the distribution and search intensification. For what concerns the distribution, there exist a number of choices. Here, we adopt a d-variate normal distribution parameterized by mean vector and covariance matrix. In order to intensify the search around the promising configurations, a d-variate normal distribution is defined on each surviving configuration from the previous iteration such that the distribution is centered at the values of the corresponding configuration. Moreover, the spread of the normal densities given by the covariance matrix is gradually reduced at each iteration.

This appendix focuses on a scenario in which the practitioner does not have any *a priori* knowledge on the parameter values. Hence, we assume that the values taken by the parameters are independent, that is, knowing a value for a particular parameter does

not give any information on the values taken by the other parameters. Consequently, the d-variate normal distribution is factorized by a product of d univariate independent normal densities parameterized by $\mu = (\mu_1, \ldots, \mu_d)$ and $\sigma = (\sigma_1, \ldots, \sigma_d)$. At each iteration, the standard deviation vector σ of the normal densities is reduced heuristically using the idea of volume reduction. Suppose that N_s configurations survive after a given iteration; we denote the surviving configurations as $\theta_s = (x_1^s, \ldots, x_d^s)$, for $s = 1, \ldots, N_s$. At a given iteration r, let V_r be the total volume of the d-dimensional sampling region bounded by $(\mu_k^{s_r} \pm \sigma_k^{s_r})$, for $k = 1, \ldots, d$; for iteration $r + 1$, in order to intensify the search, we reduce the volume of the sampling region by a factor equal to the number of sample configurations allowed for each iteration, N_{max}; therefore $\mathrm{V}_{r+1} = \mathrm{V}_r / N_{max}$, from which after some basic mathematical transformation, we have:

$$\sigma_k^s = R_k^{s_{prev}} \cdot \left(\frac{1}{N_{max}}\right)^{1/d} \qquad \text{for } k = 1, \ldots, d, \tag{B.3}$$

where $R_k^{s_{prev}}$ is set to standard deviation of the normal distribution component from which x_k^s has been sampled from the previous iteration. In simple terms, the adoption of Equation B.3 allows `I/F-Race` to reduce the range of each parameter that falls around one standard deviation from the mean at a constant rate of $(1/N_{max})^{1/d}$ for each iteration—the larger the value of N_{max}, the higher the rate of volume reduction. Though one could use more advanced techniques to update the distribution as suggested by the model-based search framework (Zlochin et al., 2004), we have adopted the above described heuristic way of intensifying search due to its simplicity.

Note that in the first iteration, a d-variate uniform distribution is used as the probability measure, thus for the following iteration, $R_k^{s_{prev}}$ is set to the half of range, that is, $(max_k - min_k)/2$, where max_k and min_k are parameters of the uniform distribution component from which x_k^s has been sampled, respectively.

The proposed approach adopts a strategy in which the number of configurations drawn from a d-variate normal distribution defined on a surviving configuration is inversely proportional to the configurations' expected solution cost. Recall that we are faced with the minimization of the expected solution cost. To do so, a *selection probability* is defined: the surviving configurations are ranked according to their expected solution costs and the probability of selecting a d-variate normal distribution defined on a configuration with rank z is given by

$$p_z = \frac{N_s - z + 1}{N_s \cdot (N_s + 1)/2}. \tag{B.4}$$

A configuration is obtained by first choosing a d-variate normal distribution according to Equation B.4, and then sampling from the chosen distribution. This is repeated until N_{max} configurations are sampled.

B.2.2.1 Implementation specific details

In order to guarantee that I/F-Race does a specific minimum number of iterations and that it has a minimum number of survivors, we have modified F-Race slightly to stop it prematurely. At each iteration, the race is stopped if one of the following conditions is true:

- when N_{min} configurations remain;
- when a certain amount of computational budget, CB_{min}, is used;
- when the configurations in the race are evaluated on at least I_{max} instances.

Though these modifications introduce 3 parameters, they are set in a reasonable and straightforward way with respect to the total computational budget CB when the algorithm starts: (i) CB_{min} is set to $CB/5$: this setting allows I/F-Race to perform at least five iterations; (ii) N_{min} is set to d: this setting enables I/F-Race to search in a number of promising regions rather than just concentrating on a single region; (iii) I_{max} is set to $2 \cdot (CB_{min}/N_{max})$: if none of the configurations is eliminated from the race then each configuration has been evaluated on CB_{min}/N_{max} instances; hence, twice this value seems to be a reasonable upper bound.

The maximum number N_{max} of configurations allowed for each race is kept constant throughout the procedure. Moreover, the N_s configurations that have survived the race are allowed to compete with the newly sampled configurations. Therefore, $N_{max} - N_s$ configurations are sampled anew at each iteration.

The order in which the instances are given to the race is randomly shuffled for each iteration. Since the surviving configurations of each race are allowed to enter into the next race, their results could be reused if the configuration has already been evaluated on a particular instance. However, since we do not want to bias I/F-Race in the empirical study, we did not use this possibility here.

The boundary constraints are handled in an explicit way. We adopt a method that consists in assigning the boundary value if the sampled value is outside the boundary. The rationale behind this adoption is to allow the exploration of values that lay at the boundary. In the case of parameters that take integer values, the value assigned to each integer parameter in the entire procedure is rounded off to the nearest integer.

B.3 Experiments

In this section, we study the proposed `RSD/F-Race` and `I/F-Race` using three examples. Though any parameterized algorithm may be tuned, all three examples concern the tuning of stochastic local search algorithms (Hoos and Stützle, 2005): (i) tuning $\mathcal{MAX} - \mathcal{MIN}$ ant system ($\mathcal{MM}$AS) (Stützle and Hoos, 2000), a particular ant colony optimization algorithm, for a class of instances of the TSP, (ii) tuning a variant of the estimation-based iterative improvement algorithm `2.5-opt-EEais` for a class of instances of the PTSP, and (iii) tuning a simulated annealing algorithm for a class of instances of the VRPSD. The primary goal of these examples is to show that `RSD/F-Race` and `I/F-Race` can significantly reduce the computational budget required for tuning.

We compare `RSD/F-Race` and `I/F-Race` with an implementation of `F-Race` that uses a full factorial design (`FFD`). For `RSD/F-Race` and `I/F-Race` we make the assumption that the *a priori* knowledge on the parameter values is not available. In the case of `FFD`, we consider two variants:

1. `FFD` that uses *a priori* knowledge; a parameter M_k is allowed to take l_k values, for $k = 1, \ldots, d$, where l_k values are chosen according to the *a priori* knowledge available on the parameter values; we denote this variant by $\texttt{FFD}_{\mathrm{A}}$`/F-Race`.

2. `FFD` that uses random values: a parameter M_k is allowed to take l_k values, for $k = 1, \ldots, d$, where l_k values are chosen randomly; we denote this variant by $\texttt{FFD}_{\mathrm{R}}$`/F-Race`. Note that the number of configurations in this variant is the same as that of $\texttt{FFD}_{\mathrm{A}}$`/F-Race`. This serves as a yardstick to analyze the usefulness of the *a priori* knowledge. The rationale behind the adoption of this yardstick is that if one just takes random values for `FFD` and achieves better results then $\texttt{FFD}_{\mathrm{A}}$`/F-Race`, then we can conjecture that the available *a priori* knowledge is either not accurate or simply not useful, at least in the examples that we consider here.

The minimum number of steps allowed in `F-Race` for all algorithms before applying the *Friedman* test is set to 5 as proposed in Birattari (2004).

The maximum computational budget of $\texttt{FFD}_{\mathrm{A}}$`/F-Race` and $\texttt{FFD}_{\mathrm{R}}$`/F-Race` are set to 10 times the number of initial configurations. The rationale behind this choice is that, if none of the configurations is eliminated, $\texttt{FFD}_{\mathrm{A}}$`/F-Race` and $\texttt{FFD}_{\mathrm{R}}$`/F-Race` evaluate all the configurations on at least 10 instances. This budget is also given for `RSD/F-Race` and `I/F-Race`. In order to force `RSD/F-Race` to use the entire computational budget, the number of configurations is set to one-sixth of the computational budget. Since

I/F-Race needs to perform at least five F-races with the same budget as that of RSD/F-Race, the number of initial configurations in each F-Race run by I/F-Race is set to one-fifth of the number of configurations given to RSD/F-Race. Moreover, in order to study the effectiveness of RSD/F-Race and I/F-Race under strong budget constraints, the computational budget is reduced by a factor of two, four, and eight. Note that, in these cases, the number of configurations in RSD/F-Race and I/F-Race is set according to the allowed budget using the same rule as described before.

Each tuning algorithm is allowed to perform 10 trials and the order in which the instances are given to an algorithm is randomly shuffled for each trial.

All tuning algorithms were implemented and run under R version 2.4[1] and we used a public domain implementation of F-Race in R which is freely available for download (Birattari, 2003). $\mathcal{MM}$AS[2] and the estimation-based iterative improvement algorithm were implemented in C and compiled with gcc, version 3.4. Simulated annealing for the VRPSD is implemented in C++. Experiments were carried out on AMD Opteron^{TM}244 1.75 GHz processors with 1 MB L2-Cache and 2 GB RAM, running under the Rocks Cluster Distribution 4.2 GNU/Linux.

In order to quantify the effectiveness of each algorithm, we study the expected solution cost of the winning configuration $\mathcal{C}(\theta^*)$, where the expectation is taken with respect to the set of all trials and the set of all test instances. We report the expected solution cost of each algorithm, measured as the percentage deviation from a *reference cost*, which is given by the average over $\mathcal{C}(\theta^*)$ obtained by each algorithm. The adoption of *reference cost* allows us to compare the expected solution cost of different algorithms more directly.

In order to test whether the observed differences between the expected solution costs of different tuning algorithms are significant in a statistical sense, a random permutation test is adopted. The level of significance at which we reject the null hypothesis is 0.05; two sided p-value is computed for each comparison.

B.3.1 Tuning $\mathcal{MM}$AS for TSP

In this study, we tune 6 parameters of $\mathcal{MM}$AS:

1. relative influence of pheromone trails, α;

[1] R is a language and environment for statistical computing that is freely available under the GNU GPL license at http://www.r-project.org/

[2] We used the ACOTSP package, which is a public domain software that provides an implementation of various ant colony optimization algorithms applied to the symmetric TSP. The package available at: http://www.aco-metaheuristic.org/aco-code/

Table B.1: Ranges of the parameter values considered for tuning $\mathcal{MM}$AS for TSP with `RSD/F-Race` and `I/F-Race`

parameter	range
α	$[0.0, 1.5]$
β	$[0.0, 5.0]$
ρ	$[0.0, 1.0]$
γ	$[0.01, 5.00]$
m	$[1, 1200]$
nn	$[5, 50]$

2. relative influence of heuristic information, β;
3. pheromone evaporation rate, ρ;
4. parameter used in computing the minimum pheromone trail value τ_{min}, γ, which is given by $\tau_{max}/(\gamma * instance\_size)$;
5. number of ants, m;
6. number of neighbors used in the solution construction phase, nn.

In $\texttt{FFD}_\text{A}$`/F-Race` and $\texttt{FFD}_\text{R}$`/F-Race`, each parameter is allowed to take 3 values. The parameter values in $\texttt{FFD}_\text{A}$`/F-Race` are set as follows: $\alpha \in \{0.75, 1.00, 1.50\}$, $\beta \in \{1.00, 3.00, 5.00\}$, $\rho \in \{0.01, 0.02, 0.03\}$, $\gamma \in \{1.00, 2.00, 3.00\}$, $m \in \{500, 750, 1000\}$, and $nn \in \{20, 30, 40\}$. These values are chosen reasonably close to the values proposed in Dorigo and Stützle (2004). Note that the values are chosen from the version without the local search. Table B.1 shows the ranges of the parameters considered for `RSD/F-Race` and `I/F-Race`. The computational time allowed for evaluating a configuration on an instance is set to 20 seconds. Instances are generated with the DIMACS instance generator (Johnson et al., 2001). We used uniformly distributed Euclidean instances of size 750; 1000 instances were generated for tuning; 300 other instances were generated for evaluating the winning configuration. Table B.2 shows the percentage deviation of each algorithms' expected solution cost from the *reference cost*, the maximum budget allowed for each algorithm and the average number of evaluations used by each algorithm.

From the results, we can see that `I/F-Race` is very competitive: under equal computational budget, the expected solution cost of `I/F-Race` is approximately 17% and 15% less than that of $\texttt{FFD}_\text{R}$`/F-Race` and $\texttt{FFD}_\text{A}$`/F-Race`, respectively (the observed differences are significant according to the random permutation test). On the other hand,

Table B.2: Computational results for tuning $\mathcal{MM}$AS for TSP. The column entries with the label `per.dev` shows the percentage deviation of each algorithms' expected solution cost from the *reference cost*: $+x$ means that the expected solution cost of the algorithm is $x\%$ more than the *reference cost* and $-x$ means that the expected solution cost of the algorithm is $x\%$ less than the *reference cost*. The column entries with the label with `max.bud` shows the maximum number of evaluations given to each algorithm and the column with the label `usd.bud` shows the average number of evaluations used by each algorithm.

algo	per.dev	max.bud	usd.bud
$\texttt{FFD}_\text{R}$/F-Race	+13.45	7290	5954
$\texttt{FFD}_\text{A}$/F-Race	+11.13	7290	5233
RSD/F-Race	−2.69	7290	7232
I/F-Race	−3.92	7290	7181
RSD/F-Race	−2.55	3645	3275
I/F-Race	−3.84	3645	3564
RSD/F-Race	−2.51	1822	1699
I/F-Race	−3.66	1822	1793
RSD/F-Race	−2.17	911	823
I/F-Race	−3.23	911	894

the expected solution cost of RSD/F-Race is also very low. However, I/F-Race reaches an expected cost that is about 1% less than that of RSD/F-Race. Indeed, the observed difference is significant in a statistical sense. Regarding the budget, $\texttt{FFD}_\text{R}$/F-Race and $\texttt{FFD}_\text{A}$/F-Race use only 80% and 70% of the maximum budget. This early termination of the F-Race is attributed to the adoption of FFD: since, there are rather few possible values for each parameter, the inferior configurations are identified and discarded within few steps. However, the poor performance of $\texttt{FFD}_\text{R}$/F-Race and $\texttt{FFD}_\text{A}$/F-Race is not only attributable to the fact that they do not use the budget effectively. Given only half of the computational budget (a maximum budget of 3645), RSD/F-Race and I/F-Race achieve expected solution costs that are still 17% and 15% lower than $\texttt{FFD}_\text{R}$/F-Race and $\texttt{FFD}_\text{A}$/F-Race, respectively (the observed differences are significant according to the random permutation test). Another important observation is that, in the case of I/F-Race and RSD/F-Race, reducing the budget does not degrade the effectiveness to a large extent. Furthermore, in all these reduced budget cases, I/F-Race achieves an expected solution cost which is approximately 1% less than that of RSD/F-Race (the observed differences are significant according to the random permutation test).

Table B.3: Ranges of the parameter values considered for tuning the estimation-based iterative improvement for the PTSP with `RSD/F-Race` and `I/F-Race`

parameter	range
p_1	$[0.0, 1.0]$
w	$[0, 100]$
p_2	$[0.0, 1.0]$

B.3.2 Tuning estimation-based iterative improvement algorithm for the PTSP

In this section, we consider a simple variant of `2.5-opt-EEais`. The adopted variant differs from `2.5-opt-EEais` presented in Chapter 5 only with respect to way in which the shorter segment parameter min_{is} is used. In the simple variant, min_{is} is the the number of nodes of the shorter segment and all the nodes in the shorter segment are biased.

1. importance sampling probability for 2-exchange moves, p';
2. number of nodes to determine the shorter segment in 2-exchange moves, min_{is};
3. importance sampling probability for node-insertion moves p''.

In $\texttt{FFD}_\text{A}\texttt{/F-Race}$, the values are assigned by discretization: for each parameter, the range is discretized as follows: $p' = p'' \in \{0.16, 0.33, 0.50, 0.66, 0.83\}$, and $min_{is} = \{8, 17, 25, 33, 42\}$. Table B.3 shows the ranges of the parameters considered for `RSD/F-Race` and `I/F-Race`. The algorithm is allowed to run until it reaches a local optimum. We used clustered Euclidean instances of size 1000; 800 instances were generated for tuning; 800 more instances were generated for evaluating the winning configuration.

The computational results show that the difference between the expected cost of the solutions obtained by different algorithms exhibits a trend similar to the one observed in the TSP experiments. However, the percentage deviations from the *reference cost* are relatively small: under equal computational budget, the expected solution cost of `I/F-Race` and `RSD/F-Race` are approximately 2% less than that of $\texttt{FFD}_\text{R}\texttt{/F-Race}$ and $\texttt{FFD}_\text{A}\texttt{/F-Race}$, respectively. Note that this difference is significant according to a random permutation test. Though `RSD/F-Race` obtains an expected solution cost which is 0.01% less than that of `I/F-Race`, the random permutation test does not reject the null hypothesis. The overall low percentage deviation between algorithms is attributed to the fact that the estimation-based iterative improvement algorithm is not extremely

Table B.4: Computational results for tuning the estimation-based iterative improvement algorithm for PTSP. The column entries with the label `per.dev` shows the percentage deviation of each algorithms' expected solution cost from the *reference cost*: $+x$ means that the expected solution cost of the algorithm is $x\%$ more than the *reference cost* and $-x$ means that the expected solution cost of the algorithm is $x\%$ less than the *reference cost*. The column entries with the label with `max.bud` shows the maximum number of evaluations given to each algorithm and the column with the label `usd.bud` shows the average number of evaluations used by each algorithm.

algo	per.dev	max.bud	usd.bud
FFD_R/F-Race	+1.45	1250	1196
FFD_A/F-Race	+1.52	1250	1247
RSD/F-Race	−0.62	1250	1140
I/F-Race	−0.53	1250	1232
RSD/F-Race	−0.17	625	615
I/F-Race	−0.52	625	618
RSD/F-Race	−0.06	312	307
I/F-Race	−0.58	312	278
RSD/F-Race	−0.37	156	154
I/F-Race	−0.11	156	150

sensitive to the parameter values: there are only 3 parameters and interactions among them are quite low. As a consequence, the tuning task becomes relatively easy (as in the case of the previous task of tuning of $\mathcal{MM}$AS). This can be easily seen with the used budget of FFD_R/F-Race: if the task of finding good configurations were difficult, the race would have terminated early. Yet, this is not the case and almost the entire computational budget has been used.

The numerical results on the budget constraints show that both RSD/F-Race and I/F-Race are indeed effective. Given only one-eighth of the computational budget (a maximum budget of 156 evaluations), RSD/F-Race and I/F-Race achieve expected solution costs which are approximately 1.4% less than that of FFD_R/F-Race and FFD_A/F-Race. This observed difference is significant according to the random permutation test. However, in this case, the random permutation test cannot reject the null hypothesis that RSD/F-Race and I/F-Race achieve expected solution costs that are equivalent. On the other hand, given one-half and one-fourth of the computational budget, I/F-Race achieves an expected solution cost that is approximately 0.4% less that of RSD/F-Race (observed differences are significant according to the random permutation test).

B.3.3 Tuning a simulated annealing algorithm for the VRPSD

In this study, 4 parameters of a simulated annealing algorithm have been tuned:

Table B.5: Ranges of the parameter values considered for tuning a simulated annealing algorithm for VRPSD with `RSD/F-Race` and `I/F-Race`

parameter	range
α	[0.0, 1.0]
q	[1, 100]
r	[1, 100]
f	[0.01, 1.00]

1. cooling rate, α;
2. a parameter used to compute the number of iterations after which the process of reheating can be applied, q;
3. another parameter used to compute the number of iterations after which the process of reheating can be applied, r;
4. parameter used in computing the starting temperature value, f;

In $\texttt{FFD}_\text{A}$`/F-Race` and $\texttt{FFD}_\text{R}$`/F-Race`, each parameter is allowed to take 3 values and in the former, the values are chosen close to the values adopted in Pellegrini and Birattari (2006): $\alpha \in \{0.25, 0.50, 0.75\}$, $q \in \{1, 5, 10\}$, $r \in \{20, 30, 40\}$, $f \in \{0.01, 0.03, 0.05\}$. Table B.5 shows the ranges of the parameters considered for `RSD/F-Race` and `I/F-Race`. In all algorithms, the computational time allowed for evaluating a configuration on an instance is set to 10 seconds. Instances are generated as described in Pellegrini and Birattari (2006); 400 instances were generated for tuning; 200 more instances were generated for evaluating the winning configuration.

The computational results show that, similar to the previous example, the tuning task is rather easy. Concerning the expected solution cost, the randomized permutation test cannot reject the null hypothesis that the different algorithms produce equivalent results. However, it should be noted that the main advantage of `RSD/F-Race` and `I/F-Race` is their effectiveness under strong budget constraints: `RSD/F-Race` and `I/F-Race`, given only one-eighth of the computational budget, achieve expected solution costs that are not significantly different from $\texttt{FFD}_\text{R}$`/F-Race` and $\texttt{FFD}_\text{A}$`/F-Race`.

B.4 Related work

The problem of tuning SLS algorithms is essentially a mixed variable stochastic optimization problem. Even though a number of algorithms exist for mixed variable

Table B.6: Computational results for tuning a simulated annealing algorithm for the VRPSD. The column entries with the label `per.dev` shows the percentage deviation of each algorithms' expected solution cost from the *reference cost*: $+x$ means that the expected solution cost of the algorithm is $x\%$ more than the *reference cost* and $-x$ means that the expected solution cost of the algorithm is $x\%$ less than the *reference cost*. The column entries with the label with `max.bud` shows the maximum number of evaluations given to each algorithm and the column with the label `usd.bud` shows the average number of evaluations used by each algorithm.

algo	per.dev	max.bud	usd.bud
FFD_R/F-Race	+0.02	810	775
FFD_A/F-Race	+0.11	810	807
RSD/F-Race	−0.05	810	804
I/F-Race	−0.03	810	797
RSD/F-Race	−0.03	405	399
I/F-Race	−0.05	405	399
RSD/F-Race	+0.02	202	200
I/F-Race	−0.01	202	200
RSD/F-Race	+0.02	101	101
I/F-Race	+0.02	101	100

stochastic optimization, it is quite difficult to adopt them for tuning. The primary obstacle is that, since these algorithms have parameters, tuning them is indeed paradoxical. Few procedures have been developed specifically for tuning algorithms: Kohavi and John (1995) proposed an algorithm that makes use of best-first search and cross-validation for automatic parameter selection. Boyan and Moore (1997) introduced a tuning algorithm based on machine learning techniques. The main emphasis of these two works is given only to the parameter value selection; there is no empirical analysis of these algorithms when applied to large number of parameters that have wide range of possible values. Audet and Orban (2006) proposed a pattern search technique called mesh adaptive direct search that uses surrogate models for algorithmic tuning. In this approach, a conceptual mesh is constructed around a solution and the search for better solutions is done around this mesh. The surrogates are used to reduce the computation time by providing an approximation to the original response surface. Nevertheless, this approach has certain number of parameters and it has never been used for tuning SLS algorithms. Adenso-Diaz and Laguna (2006) designed an algorithm called CALIBRA specifically for fine tuning SLS algorithms. It uses Taguchi's fractional factorial experimental designs coupled with local search. In this work, the authors explicitly mention that tuning a wide range of possible values for parameters is feasible with their algorithm. However, a major limitation of this algorithm is that

one cannot use it for tuning SLS algorithms with more than five parameters. Beielstein et al. (2002) proposed an approach to reduce the difficulty of the tuning task. This approach consists in first identifying the parameters that have a significant impact on the algorithms' performance through sensitivity analysis and then tuning them. Recently, Hutter et al. (2007) proposed an iterated local search algorithm for parameter tuning called paramILS. This algorithm is shown to be very effective and most importantly, it can be used to tune algorithms with a large number of parameters.

B.5 Summary

We proposed two supplementary procedures for `F-Race` that are based on random sampling, `RSD/F-Race`, and model-based search techniques, `I/F-Race`. While the adoption of full factorial design in the `F-Race` framework is impractical and computationally prohibitive when used to identify the best from a large number of parameter configurations, `RSD/F-Race` and `I/F-Race` are useful in such cases. We also showed that `RSD/F-Race` and `I/F-Race` can be deployed to reduce the computational time required for tuning.

References

E. Aarts and J. Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, UK, 1997.

B. Adenso-Diaz and M. Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114, 2006.

T. Alkhamis and M. Ahmed. Simulation-based optimization using simulated annealing with confidence intervals. In R. Ingalls, M. Rossetti, J. Smith, and B. Peters, editors, *Proceedings of the 2004 winter simulation conference (WSC04)*, pages 514–518, Piscataway, NJ, 2004. IEEE Press.

T. M. Alkhamis, M. A. Ahmed, and V. K. Tuan. Simulated annealing for discrete optimization with estimation. *European Journal of Operational Research*, 116(3): 530–544, 1999.

M. Alrefaei and S. Andradóttir. A simulated annealing algorithm with constant temperature for discrete stochastic optimization. *Management Science*, 45(5):748–764, 1999.

M. Alrefaei and S. Andradóttir. A modification of the stochastic ruler method for discrete stochastic optimization. *European Journal of Operational Research*, 133(1): 160–182, 2001.

S. Andradóttir. A method for discrete stochastic optimization. *Management Science*, 41(12):1946–1961, 1995.

S. Andradóttir. A global search method for discrete stochastic optimization. *SIAM Journal on Optimization*, 6(2):513–530, 1996.

D. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. Concorde—a code for solving traveling salesman problems, 2001. URL `http://www.math.princeton.edu/tsp/concorde.html`.

REFERENCES

R. Aringhieri. Solving chance-constrained programs combining tabu search and simulation. In C. Ribeiro and S. Martins, editors, *Proceedings of the 3rd international workshop on experimental and efficient algorithms (WEA04)*, volume 3059 of *LNCS*, pages 30–41, Berlin, Germany, 2004. Springer-Verlag.

C. Audet and D. Orban. Finding optimal algorithmic parameters using the mesh adaptive direct search algorithm. *SIAM Journal on Optimization*, 17(3):642–664, 2006.

P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Adaptive sample size and importance sampling in estimation-based local search for the probabilistic traveling salesman problem: A complete analysis. Technical Report TR/IRIDIA/2007-015, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2007. URL `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2007-015r002.pdf`.

P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Estimation-based metaheuristics for the probabilistic traveling salesman problem. IRIDIA Supplementary page, 2008a. URL `http://iridia.ulb.ac.be/supp/IridiaSupp2008-019/`.

P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Extended empirical analysis of adaptive sample size and importance sampling in estimation-based local search for the probabilistic traveling salesman problem. IRIDIA Supplementary page, 2008b. URL `http://iridia.ulb.ac.be/supp/IridiaSupp2008-010/`.

P. Balaprakash, M. Birattari, T. Stützle, Z. Yuan, and M. Dorigo. Estimation-based ant colony optimization and local search for the probabilistic traveling salesman problem. IRIDIA Supplementary page, 2008c. URL `http://iridia.ulb.ac.be/supp/IridiaSupp2008-018/`.

P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Estimation-based metaheuristics for the the vehicle routing problem with stochastic demands and customers. IRIDIA Supplementary page, 2009a. URL `http://iridia.ulb.ac.be/supp/IridiaSupp2009-008/`.

P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Estimation-based metaheuristics for the probabilistic traveling salesman problem: A comparison to progressive approximation, the aggregation approach, and simulated annealing. Technical Report TR/IRIDIA/2009-025, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2009b. URL `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2009-025r001.pdf`.

R. Barr, B. Golden, J. Kelly, M. Rescende, and W. Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1): 9–32, 1995.

S. Becker, J. Gottlieb, and T. Stützle. Applications of racing algorithms: An industrial perspective. In E.-G. Talbi, P. Liardet, P. Collet, E. Lutton, and M. Schoenauer, editors, *Artificial Evolution: 7th International Conference, Evolution Artificielle, EA 2005*, volume 3871 of *LNCS*, pages 271–283, Berlin, Germany, 2005. Springer-Verlag.

T. Beielstein, K. Parsopoulos, and M. V. and. Tuning PSO parameters through sensitivity analysis. Technical report, Universität Dortmund, Dortmund, Germany, 2002. URL `https://eldorado.tu-dortmund.de/handle/2003/5420`.

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

W. Benton and M. Rossetti. The vehicle scheduling problem with intermittent customer demands. *Computers and Operations Research*, 19(6):521–531, 1992.

P. Beraldi and A. Ruszczynski. Beam search heuristic to solve stochastic integer problems under probabilistic constraints. *European Journal of Operational Research*, 167 (1):35–47, 2005.

O. Berman and D. Simchi-Levi. Finding the optimal a priori tour and location of a traveling salesman finding the optimal a priori tour and location of a traveling salesman with nonhomogeneous customers. *Transportation Science*, 22(2):148–154, 1988.

D. P. Bertsekas and D. A. Castanon. Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, 5(1):89–108, 04 1999.

D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3(3):245–262, 1997.

D. Bertsimas. *Probabilistic Combinatorial Optimization Problems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1988.

D. Bertsimas and L. Howell. Further results on the probabilistic traveling salesman problem. *European Journal of Operational Research*, 65(1):68–95, 1993.

D. Bertsimas, P. Jaillet, and A. Odoni. A priori optimization. *Operations Research*, 38 (6):1019–1033, 1990.

REFERENCES

D. J. Bertsimas. A vehicle routing problem with stochastic demand. *Operations Research*, 40(3), 1992.

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

L. Bianchi. *Ant Colony Optimization and Local Search for the Probabilistic Traveling Salesman Problem: A Case Study in Stochastic Combinatorial Optimization.* PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2006.

L. Bianchi and A. Campbell. Extension of the 2-p-opt and 1-shift algorithms to the heterogeneous probabilistic traveling salesman problem. *European Journal of Operational Research*, 176(1):131–144, 2007.

L. Bianchi and L. M. Gambardella. Ant colony optimization and local search based on exact and estimated objective values for the probabilistic traveling salesman problem. Technical Report IDSIA-06-07, IDSIA, USI-SUPSI, Manno, Switzerland, June 2007.

L. Bianchi, L. Gambardella, and M. Dorigo. Solving the homogeneous probabilistic travelling salesman problem by the ACO metaheuristic. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms, Third International Workshop, ANTS 2002*, volume 2463 of *LNCS*, pages 176–187, Berlin, Germany, 2002a. Springer-Verlag.

L. Bianchi, L. M. Gambardella, and M. Dorigo. An ant colony optimization approach to the probabilistic traveling salesman problem. In J. J. Guervós, P. Adamidis, H. Beyer, J. L. Martín, and H. Schwefel, editors, *7th International Conference on Parallel Problem Solving From Nature, PPSN VII*, volume 2439 of *LNCS*, pages 883–892, Berlin, Germany, 2002b. Springer-Verlag.

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*, 162:206–219, 2005.

L. Bianchi, M. Birattari, M. Chiarandini, 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*, 5(1):91–110, 2006.

L. Bianchi, M. Dorigo, L. Gambardella, and W. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.

M. Birattari. The race package for R. Racing methods for the selection of the best. Technical Report TR/IRIDIA/2003-37, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2003. URL http://cran.r-project.org/src/contrib/Descriptions/race.html.

M. Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2004.

M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Springer-Verlag, Berlin, Germany, 2009.

M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In W. B. Langdon, editor, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18, San Francisco, CA, 2002. Morgan Kaufmann.

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, Vienna, Austria, 2005.

M. Birattari, P. Balaprakash, and M. Dorigo. The ACO/F-RACE algorithm for combinatorial optimization under uncertainty. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. J. Gutjahr, R. F. Hartl, and M. Reimann, editors, *Metaheuristics - Progress in Complex Systems Optimization*, Operations Research/Computer Science Interfaces Series, pages 189–203, Berlin, Germany, 2006a. Springer-Verlag.

M. Birattari, M. Zlochin, and M. Dorigo. Towards a theory of practice in metaheuristics design: A machine learning perspective. *Theoretical Informatics and Applications*, 40(2):353–369, 2006b.

M. Birattari, P. Balaprakash, T. Stützle, and M. Dorigo. Extended empirical analysis of estimation-based local search for stochastic combinatorial optimization. IRIDIA Supplementary page, 2007. URL http://iridia.ulb.ac.be/supp/IridiaSupp2007-001/.

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

REFERENCES

N. E. Bowler, T. M. A. Fink, and R. C. Ball. Characterization of the probabilistic traveling salesman problem. *Physical Review E*, 68(3):036703–036710, 2003.

J. Boyan and A. Moore. Using prediction to improve combinatorial optimization search. In P. Smyth and D. Madigan, editors, *Sixth International Workshop on Artificial Intelligence and Statistics*, Fort Lauderdale, Florida, 1997.

J. Branke and M. Guntsch. New ideas for applying ant colony optimization to the probabilistic TSP. In G. Raidl, S. Cagnoni, J. Cardalda, D. Corne, J. Gottlieb, A. Guillot, E. Hart, C. Johnson, E. Marchiori, J.-A. Meyer, and M. Middendorf, editors, *Applications of Evolutionary Computing, 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2003*, volume 2611 of *LNCS*, pages 165–175, Berlin, Germany, 2003. Springer-Verlag.

J. Branke and M. Guntsch. Solving the probabilistic TSP with ant colony optimization. *Journal of Mathematical Modelling and Algorithms*, 3(4):403–425, 2004.

J. Branke, S. E. Chick, and C. Schmidt. Selecting a selection procedure. *Management Science*, 53(12):1916–1932, 2007.

A. Bulgak and J. 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, Piscataway, NJ, 1998. IEEE Press.

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

A. Campbell and B. Thomas. Probabilistic traveling salesman problem with deadlines. *Transportation Science*, 42(1):1–21, 2008a.

A. Campbell and B. Thomas. Runtime reduction techniques for the probabilistic traveling salesman problem with deadlines. *Computers and Operations Research*, 36(4): 1231–1248, 2009.

A. M. Campbell. Aggregation for the probabilistic traveling salesman problem. *Computers and Operations Research*, 33(9):2703–2724, 2006.

A. M. Campbell and B. W. Thomas. Challenges and advances in a priori routing. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces*, pages 123–142. Springer-Verlag, Berlin, Germany, 2008b.

R. Carraway, T. Morin, and H. Moskowitz. Generalized dynamic programming for stochastic combinatorial optimization. *Operations Research*, 37(5):819–829, 1989.

K. Chepuri and T. Homem-de-Mello. Solving the vehicle routing problem with stochastic demands using the cross-entropy method. *Annals of Operations Research*, 134(1): 153–181, 2005.

P. Chervi. A computational approach to probabilistic vehicle routing problems. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1988.

R. Cheung, D. Xu, and Y. Guan. A solution method for a two-dispatch delivery problem with stochastic customers. *Journal of Mathematical Modelling and Algorithms*, 6(1): 87–107, 2007.

M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9(5):403–432, 2006.

W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, third edition, 1999.

J. Cordeau, M. Gendreau, G. Laporte, J. Potvin, and F. Semet. A guide to vehicle routing heuristics. *Journal of the Operational Research Society*, 53(5):512–522, 2002.

O. Cordón, I. F. de Viana, and F. Herrera. Analysis of the best-worst ant system and its variants on the TSP. *Mathware and Soft Computing*, 9(2–3):177–192, 2002.

D. Costa and E. Silver. Tabu search when noise is present: An illustration inthe context of cause and effect analysis. *Journal of Heuristics*, 4(1):5–23, 1998.

J. 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.

B. Dengiz and C. Alabas. Simulation optimization using tabu search. In J. Joines, R. B. RR, and K. K. P. Fishwick, editors, *Proceedings of the 2000 Winter Simulation Conference (WSC00)*, pages 805–810, Piscataway, NJ, 2000. IEEE Press.

REFERENCES

K. F. Doerner, W. Gutjahr, G. Kotsis, M. Polaschek, and C. Strauss. Enriched workflow modelling and stochastic branch-and-bound. *European Journal of Operational Research*, 175(3):1798–1817, 2006.

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

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

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

M. Dror. Modeling vehicle routing with uncertain demands as a stochastic program: properties of the corresponding solution. *European Journal of Operational Research*, 64(3):432–41, 1993.

M. Dror and P. Trudeau. Stochastic vehicle routing with modified savings algorithm. *European Journal of Operational Research*, 23(2):228–235, 1986.

M. Dror, G. Laporte, and P. Trudeau. Vehicle routing with stochastic demands: Properties and solution frameworks. *Transportation Science*, 23(3):166–176, 1989.

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.

E. Erel, I. Sabuncuoglu, and H. Sekerci. Stochastic assembly line balancing using beam search. *International Journal of Production Research*, 43(7):1411–1426, 2005.

T. Feo and M. Resende. A probabilisitc heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Jouranal of Global Optimization*, 6:109–133, 1995.

D. Finke, D. Medeiros, and M. Traband. Shop scheduling using tabu search and simulation. In E. Yücesan, C. Chen, J. Snowdon, and J. Charnes, editors, *Proceedings of the 2002 Winter Simulation Conference (WSC02)*, pages 1013–1017, Piscataway, NJ, 2002. IEEE Press.

R. A. Fisher. *Statistical Methods for Research Workers*. Oliver and Boyd, London, UK, 1925.

L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence Through Simulated Evolution*. New York, John Wiley & Sons, 1966.

L. Fortnow. The status of the p versus np problem. *Communications of the ACM*, 52 (9):78–86, 2009.

L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software*, 33(2):1–15, 2007. URL `http://www.mpfr.org/`.

B. Fox and G. Heine. Probabilistic search with overrides. *Annals of Applied Probability*, 5(4):1087–1094, 1995.

M. C. Fu. Optimization via simulation: A review. *Annals of Operations Research*, 53: 199–248, 1994.

M. C. Fu. Optimization for simulation: theory vs. practice. *INFORMS Journal on Computing*, 14:192–215, 2002.

F. FuCe, W. Hui, and Z. Ying. Solving the vehicle routing problem with stochastic demands and customers. In *PDCAT '05: Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies*, pages 736–739, Washington, DC, 2005. IEEE Computer Society.

M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of* $\mathcal{NP}$*-Completeness*. Freeman, San Francisco, CA, 1979.

S. Gelfand and S. 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, Piscataway, NJ, 1985. IEEE Press.

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

M. Gendreau, G. Laporte, and R. Séguin. Stochastic vehicle routing. *European Journal of Operational Research*, 88:3–12, 1996a.

M. Gendreau, G. Laporte, and R. Séguin. A tabu search algorithm for the vehicle routing problem with stochastic demands and customers. *Operations Research*, 44 (3):469–477, 1996b.

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

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

F. Glover. Tabu search - part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

F. Glover and G. Kochenberger. *Handbook of Metaheuristics*, volume 57 of *International Series in Operation Research and Management Science*. Kluwer Academic Publishers, Norwell, MA, 2002.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, 1997.

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

J. J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. V. Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 160–168, Hillsdale, NJ, 1985. L. Erlbaum Associates Inc.

A. Griffith. *GCC: The Complete Reference*. McGraw Hill/Osborne Media, 2002.

W. Gutjahr, A. Hellmayr, and G. Pflug. Optimal stochastic single-machine-tardiness scheduling by stochastic branch-and-bound. *European Journal of Operational Research*, 117(2):396–413, 1999.

W. J. Gutjahr. A converging ACO algorithm for stochastic combinatorial optimization. In A. Albrecht and K. Steinhofl, editors, *Stochastic Algorithms: Foundations and Applications*, volume 2827 of *LNCS*, pages 10–25, Berlin, Germany, 2003. Springer-Verlag.

W. J. Gutjahr. S-ACO: An ant based approach to combinatorial optimization under uncertainty. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2004*, volume 3172 of *LNCS*, pages 238–249, Berlin, Germany, 2004. Springer-Verlag.

W. J. Gutjahr and G. C. Pflug. Simulated annealing for noisy cost functions. *Journal of Global Optimization*, 8(1):1–13, 1996.

W. J. Gutjahr, C. Strauss, and M. Toth. Crashing of stochastic activities by sampling and optimization. *Business Process Management Journal*, 12:125–135, 2000a.

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, 2000b.

W. J. Gutjahr, S. Katzensteiner, and P. Reiter. A vns algorithm for noisy problems and its application to project portfolio analysis. In J. Hromkovič, R. Královič, M. Nunkesser, and P. Widmayer, editors, *Stochastic Algorithms: Foundations and Applications*, volume 4665 of *LNCS*, pages 93–104, Berlin, Germany, 2007. Springer-Verlag.

J. Haddock and J. Mittenthal. Simulation optimization using simulated annealing. *Computers & Industrial Engineering*, 22(4):387–395, 1992.

E. Hadjiconstantinou and D. Roberts. Routing under uncertainty: An application in the scheduling of field service engineers. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 13, pages 331—352. SIAM Monographs on Discrete Mathematics and Applications, SIAM Publishing, Philadelphia, PA, 2002.

M. Haimovich and A. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10(4):527–542, 1985.

K. Haugen, A. Løkketangen, and D. Woodruff. Progressive hedging as a meta-heuristic applied to stochastic lot-sizing. *European Journal of Operational Research*, 132(1): 116–122, 2001.

D. Haugland, S. C. Ho, and G. Laporte. Designing delivery districts for the vehicle routing problem with stochastic demands. *European Journal of Operational Research*, 180(3):997 – 1010, 2007.

C. Hjorring and J. Holt. New optimality cuts for a single-vehicle stochastic routing problem. *Annals of Operations Research*, 86(0), 1999.

Y. Ho, R. Sreeniva, and P. Vakili. Ordinal optimization of deds. *Discrete Event Dynamic Systems*, 2(1):61–88, 1992.

REFERENCES

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

T. Homem-de-Mello. Variable-sample methods and simulated annealing for discrete stochastic optimization. *Stochastic Programming E-Print Series*, 2000. URL `http://hera.rz.hu-berlin.de/speps/`.

T. Homem-de-Mello. Variable-sample methods for stochastic optimization. *ACM Transactions on Modeling and Computer Simulation*, 13(2):108–133, 2003.

H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco, CA, 2005.

F. Hutter, H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 1152–1157, Menlo Park, CA, 2007. AAAI Press.

L. Hvattum and A. Løkketangen. Using scenario trees and progressive hedging for stochastic inventory routing problems. *Journal of Heuristics*, 15(6):527–557, 2008.

IEEE 754. IEEE Standard for Binary Floating-Point Arithmetic., 1985.

P. Jaillet. *Probabilistic Traveling Salesman Problems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1985.

P. Jaillet. Stochastic routing problems. In G. Andreatta, F. Mason, and P. Serafini, editors, *Advanced School on Stochastics in Combinatorial Optimization*, pages 192–213, Singapore, 1987. World Scientific.

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.

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*, pages 1835–1839, Piscataway, NJ, 2001. IEEE Press.

A. Jézéquel. Probabilistic vehicle routing problems. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1985.

Y. Jin and J. Branke. Evolutionary optimization in uncertain environments–a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

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, Chichester, UK, 1997.

D. S. Johnson, L. A. McGeoch, C. Rego, and F. Glover. 8th DIMACS implementation challenge, 2001. URL `http://www.research.att.com/~dsj/chtsp/`.

H. Jönsson and E. Silver. Some insights regarding selecting sets of scenarios in combinatorial stochastic problems. *International Journal of Production Economics*, 45 (1–3):463–472, 1996.

E. Kao. A preference order dynamic program for a stochastic traveling salesman problem. *Operations Research*, 26(6):1033–1045, 1978.

A. Kenyon and D. Morton. Stochastic vehicle routing with random travel times. *Transportation Science*, 37(1):69–82, 2003.

S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671—680, 1983.

A. J. Kleywegt, A. Shapiro, and T. Homem-de-Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12 (2):479–502, 2002.

R. Kohavi and G. John. Automatic parameter selection by minimizing estimated error. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 304–312, San Francisco, CA, 1995. Morgan Kaufmann.

V. Lambert, G. Laporte, and F. Louveaux. Designing collection routes through bank branches. *Computers and Operations Research*, 20(7):783–791, 1993.

G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

G. Laporte and F. Louveaux. Formulations and bounds for stochastic vehicle routing problem with uncertain supplies. In J. J. Gabzewicz, J. Richard, and L. A. Wolsey, editors, *Economic Decision-Making: Games, Econometrics and Optimization*, pages 443–455. Elsevier, Amsterdam, North-Holland, 1990.

G. Laporte, F. Louveaux, and H. Mercure. Models and exact solutions for a class of stochastic location-routing problems. *European Journal of Operational Research*, 39 (1):71–78, 1989.

G. Laporte, F. Louveaux, and H. Mercure. The vehicle routing problem with stochastic travel times. *Transportation Science*, 26(3), 1992.

G. Laporte, F. Louveaux, and H. Mercure. A priori optimization of the probabilistic traveling salesman problem. *Operations Research*, 42:543–549, 1994.

G. Laporte, M. Gendreau, J. Potvin, and F. Semet. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, 7(4–5):285–300, 2000.

G. Laporte, F. Louveaux, and L. Van Hamme. An integer l-shaped algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research*, 50(3):415–423, 2002.

Y. Liu. A hybrid scatter search for the probabilistic traveling salesman problem. *Computers and Operations Research*, 34(10):2949–2963, 2007.

Y. Liu. Diversified local search strategy under scatter search framework for the probabilistic traveling salesman problem. *European Journal of Operational Research*, 191 (2):332–346, 2008.

A. Løkketangen and D. Woodruff. Progressive hedging and tabu search applied to mixed integer (0,1) multistage stochastic programming. *Journal of Heuristics*, 2(2): 111–128, 1996.

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

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(2–3):301–316, 1998.

K. L. Mak and Z. G. Guo. A genetic algorithm for vehicle routing problems with stochastic demand and soft time windows. In M. Jones, S. Patek, and B. Tawney,

editors, *Proceedings of the 2004 IEEE systems and information engineering design symposium (SIEDS04)*, pages 183–190, Piscataway, NJ, 2004. IEEE Press.

Y. Marinakis and M. Marinaki. A hybrid multi-swarm particle swarm optimization algorithm for the probabilistic traveling salesman problem. *Computers and Operations Research*, 37(3):432—442, 2010.

Y. Marinakis, A. Migdalas, and P. Pardalos. Expanding neighborhood search–GRASP for the probabilistic traveling salesman problem. *Optimization Letters*, 2(3):1862–4472, 2008.

O. Maron and A. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 59–66, San Francisco, CA, 1994. Morgan Kaufmann.

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

P. Merz and B. Freisleben. Memetic algorithms for the traveling salesman problem. *Complex Systems*, 13(4):297–345, 2001.

Metaheuristics Network. `http://www.metaheuristics.net`, 2003. Version visited last on 15 May 2009.

N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 21: 1087—1092, 1953.

A. Moore and M. Lee. Efficient algorithms for minimizing cross validation error. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 190–198, San Francisco, CA, 1994. Morgan Kaufmann.

P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech Concurrent Computation Program Report 826, Caltech, Pasadena, California, 1989.

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

REFERENCES

V. Norkin, Y. Ermoliev, and A. Ruszczyński. On optimal allocation of indivisibles under uncertainty. *Operations Research*, 46(3):381–395, 1998a.

V. Norkin, G. Pflug, and A. Ruszczyński. A branch and bound method for stochastic global optimization. *Mathematical Programming*, 83(3):425–450, 1998b.

I. Or. *Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking*. PhD thesis, Northwestern Univesity, Evanston, IL, 1976.

C. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.

C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

P. Pellegrini and M. Birattari. The relevance of tuning the parameters of metaheuristics. A case study: The vehicle routing problem with stochastic demand. Technical Report TR/IRIDIA/2006-008, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2006.

J. F. Penky and D. L. Miller. A staged primal-dual algorithm for finding a minimum cost perfect two-matching in an undirected graph. *ORSA Journal on Computing*, 6 (1):68–81, 1994.

J. Pichitlamken and B. L. Nelson. A combined procedure for optimization via simulation. *ACM Transactions on Modeling and Computer Simulation*, 13(2):155–179, 2003.

H. Psaraftis. Dynamic vehicle routing: Status and prospects. *Annals of Operations Research*, 61(1):143–164, 1995.

I. Rechenberg. *Evolutionsstrategie — Optimierung technischer Systeme nach Prinzipien der biologischen Information*. Fromman Verlag, Freiburg, Germany, 1973.

R. Rockafellar and R.-B. Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147, 1991.

S. Rosenow. A heuristic for the probabilistic TSP. In H. Schwarze, editor, *Operations Research Proceedings*, Berlin, Germany, 1997. Springer-Verlag.

S. Rosenow. Comparison of an exact branch-and-bound and an approximate evolutionary algorithm for the probabilistic traveling salesman problem. working paper, 1998. URL `http://www2.hsu-hh.de/uebe/paper-engl-SOR98.pdf`.

F. Rossi and I. Gavioli. Aspects of heuristic methods in the probabilisitc traveling salesman problem. In *Advanced School on Stochastics in Combinatorial Optimization*, pages 214–227, Singapore, 1987. World Scientific.

R. Y. Rubinstein. *Simulation and the Monte Carlo Method*. John Wiley & Sons, Inc., New York, 1981.

M. Savelsbergh and M. Goetschalckx. A comparison of the efficiency of fixed versus variable vehicle routes. *Journal of Business Logistics*, 16(1):163–87, 1995.

N. Secomandi. Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. *Computers and Operations Research*, 27 (11):1201–1225, 2000.

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

R. Séguin. *Problèmes stochastiques de tournées de véhicules*. PhD thesis, Université de Montréal, Montréal, Canada, 1994.

L. Shi and S. Ólafsson. Nested partitions method for global optimization. *Operations Research*, 48(3):390–407, 2000.

M. Sniedovich. Analysis of a preference order traveling salesman problem. *Operations Research*, 29(6):1234–1237, 1981.

W. Stewart Jr. and B. Golden. Stochastic vehicle routing: A comprehensive approach. *European Journal of Operational Research*, 14(4):371–385, 1983.

T. Stützle. *Local Search Algorithms for Combinatorial Problems – Analysis, Improvements, and New Applications*. PhD thesis, FB Informatik, Technische Universität Darmstadt, Darmstadt, Germany, 1998.

T. Stützle. ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem, 2002. URL `http://www.aco-metaheuristic.org/aco-code/`.

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

REFERENCES

H. Tang and E. Miller-Hooks. Approximate procedures for probabilistic traveling salesperson problem. *Transportation Research Record: Journal of the Transportation Research Board*, 1882:27–36, 2004.

S. Teng, H. Ong, and H. Huang. An integer L-shaped algorithm for time-constrained traveling salesman problem with stochastic travel and service times. *Asia Pacific Journal of Operational Research*, 21(2):241–258, 2004.

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(4):261–273, 1992.

F. Tillman. The multiple terminal delivery problem with probabilistic demands. *Transportation Science*, 3(3):192—204, 1969.

P. Toth and D. Vigo, editors. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2001.

J. W. Tukey. Comparing individual means in the analysis of variance. *Biometrics*, 5 (2):99–114, 1949.

M. H. van der Vlerk. Stochastic integer programming bibliography. World Wide Web, `http://mally.eco.rug.nl/biblio/sip.html`, 1996-2007.

Černý. Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal Of Optimization Theory And Applications*, 45(1):41–51, 1985.

B. Verweij, S. Ahmed, A. Kleywegt, G. Nemhauser, and A. Shapiro. The sample average approximation method applied to stochastic routing problems: A computational study. *Computational Optimization and Applications*, 24(2-3):289–333, 2003.

L. Wang and C. Singh. Stochastic economic emission load dispatch through a modified particle swarm optimization algorithm. *Electric Power Systems Research*, 78(8): 1466–1476, 2008.

C. Waters. Vehicle-scheduling problems with uncertainty and omitted customers. *The Journal of the Operational Research Society*, 40(12):1099–1108, 1989.

J. Watson, S. Rana, L. Whitley, and A. Howe. The impact of approximate evaluation on the performance of search algorithms for warehouse scheduling. *Journal of Scheduling*, 2(2):79–98, 1999.

D. Yan and H. Mukai. Stochastic discrete optimization. *SIAM Journal on Control and Optimization*, 30(3):594–612, 1992.

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

Y. Yoshitomi. A genetic algorithm approach to solving stochastic job-shop scheduling problems. *International Transactions in Operational Research*, 9(4):479–495, 2002.

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.

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