

Université Libre de Bruxelles

*Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle*

**Estimation-based Metaheuristics for
the Probabilistic Traveling Salesman
Problem**

Prasanna BALAPRAKASH, Mauro BIRATTARI,
Thomas STÜTZLE, and Marco DORIGO

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2009-017

June 2009

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle*
UNIVERSITÉ LIBRE DE BRUXELLES
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2009-017

The information provided is the sole responsibility of the authors and does not necessarily reflect the opinion of the members of IRIDIA. The authors take full responsibility for any copyright breaches that may result from publication of this paper in the IRIDIA – Technical Report Series. IRIDIA is not responsible for any use that might be made of data appearing in this publication.

Estimation-based Metaheuristics for the Probabilistic Traveling Salesman Problem

Prasanna BALAPRAKASH

pbalapra@ulb.ac.be

Mauro BIRATTARI

mbiro@ulb.ac.be

Thomas STÜTZLE

stuetzle@ulb.ac.be

Marco DORIGO

mdorigo@ulb.ac.be

IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium

June 16, 2009

Abstract

The PROBABILISTIC TRAVELING SALESMAN PROBLEM (PTSP) is a central problem in stochastic routing. Recently, we have shown that empirical estimation is a promising approach to devise highly effective local search algorithms for the PTSP. In this paper, we customize two metaheuristics, an iterated local search algorithm and a memetic algorithm, to solve the PTSP. This customization consists in adopting the estimation approach to evaluate the solution cost, exploiting a recently developed estimation-based local search algorithm, 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.

1 Introduction

Designing effective algorithms for stochastic routing problems is a difficult task. The difficulty is due to the element of uncertainty in the data, which increases the complexity of finding an optimal solution in a large search space. Exact techniques can solve only small instances to optimality. This motivated researchers and practitioners to focus on metaheuristics, an important class of stochastic local search (SLS) methods (Hoos and Stützle, 2005). Unfortunately, the literature on metaheuristics for tackling stochastic routing problems is rather underdeveloped when compared to the deterministic case even if it is receiving increasing attention.

The PROBABILISTIC TRAVELING SALESMAN PROBLEM (PTSP) (Jaillet, 1985) is a central problem in stochastic routing. It is an \mathcal{NP} -hard problem that has a number of practical applications not only in transportation but also in strategic

planning and scheduling (Bertsimas, 1988). 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 the pruned tour: this pruned tour is obtained only after knowing the nodes that require being visited and skipping the nodes that do not require being visited according to some predefined rules.

Based on the way in which the expected cost is determined, the optimization algorithms for the PTSP can be grouped into two classes: *analytical computation* algorithms, which use closed form expressions for computing the expected cost, and *estimation-based* algorithms, which use Monte Carlo simulation for estimating the expected cost. In this paper, we tackle the PTSP by using effective estimation-based metaheuristics, a goal which is a natural extension of our two earlier research efforts. First, we developed `2.5-opt-EEais` (Birattari et al., 2008; Balaprakash et al., 2009a), a new state-of-the-art iterative improvement algorithm for the PTSP that uses an estimation-based approach to compute the cost difference between two solutions. Second, we showed that the integration of two PTSP-specific algorithmic components, `2.5-opt-EEais` as local search and an estimation-based approach to evaluate the solution cost of artificial ants, into an ant colony optimization (ACO) algorithm (Dorigo and Stützle, 2004) is highly beneficial in terms of computation time and solution quality (Balaprakash et al., 2009b). Here, we extend our work to metaheuristics that are known to have high performance on the related TRAVELING SALESMAN PROBLEM (TSP). In particular, we integrate the two PTSP-specific components into iterated local search (ILS) (Lourenço et al., 2002) and memetic algorithms (MAs) (Moscato, 1989, 1999). We present an experimental study to compare the two algorithms to the recently developed estimation-based ACO algorithm and we show that for various instance classes they can improve upon the ACO algorithm. As a control algorithm, we consider a random restart local search (RRLS) algorithm. In fact, the results show that all metaheuristics significantly outperform RRLS. A further comparison to the so far best performing analytical computation metaheuristics for the PTSP clearly establishes the estimation-based algorithms as the new state-of-the-art for the PTSP.

The paper is organized as follows. In Section 2, we describe the PTSP and its solution approaches. In Section 3, we discuss the proposed estimation-based metaheuristics. In Section 4, we evaluate their performances. In Section 5, we conclude the paper.

2 The probabilistic traveling salesman problem

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

- a set $V = \{1, 2, \dots, n\}$ 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 the nodes i and j ;

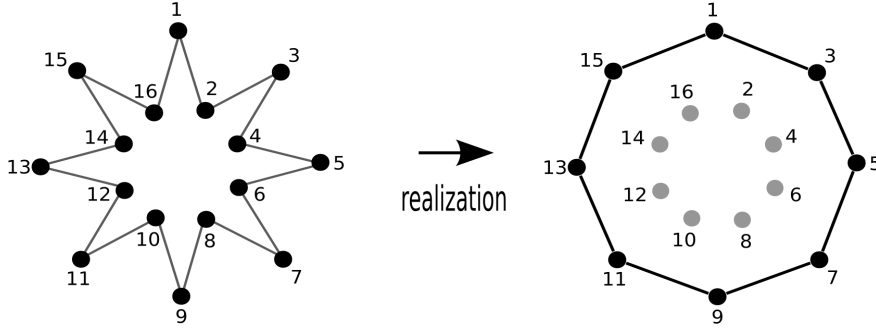


Figure 1: The left plot shows an *a priori* solution for a PTSP instance with 16 nodes. The order in which the nodes are visited in the *a priori* solution is: 1, 2, 3, ..., 15, 16, and 1. Let us assume that, according to a realization of ω , the nodes 1, 3, 5, 7, 9, 11, 13, and 15 are to be visited. The right plot shows the *a posteriori* solution that visits the nodes following the *a priori* solution but skipping the nodes 2, 4, 6, 8, 10, 12, 14, and 16.

- 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 ω that follows an n -variate Bernoulli distribution. A realization of ω 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. A PTSP instance is called homogeneous if all probability values in the set P are the same; it is called heterogeneous, if for at least two nodes the values are different.

The PTSP is usually tackled by *a priori* optimization (Jaillet, 1985; Bertsimas et al., 1990), which comprises two stages. First, an *a priori* solution, a Hamiltonian tour, is determined before the realization of ω is available. Once the nodes that require being visited are known, in the second stage the *a posteriori* solution is derived from the *a priori* solution by visiting the nodes in the same order as in the *a priori* solution and by excluding the nodes that do not require being visited. The goal is to find an *a priori* solution with minimum expected *a posteriori* solution cost. See Figure 1 for an illustration of *a priori* and *a posteriori* solutions.

The analytical computation approach computes the cost $F(x)$ of an *a priori* solution $x = (\pi(1), \pi(2), \dots, \pi(n), \pi(n+1) = \pi(1))$, where π is a permutation

of the set V , using the following closed-form expression (Jaillet, 1985):

$$\begin{aligned}
F(x) &= \sum_{i=1}^n \sum_{j=i+1}^n c_{\pi(i)\pi(j)} p_{\pi(i)} p_{\pi(j)} \prod_{k=i+1}^{j-1} (1 - p_{\pi(k)}) \\
&+ \sum_{j=1}^n \sum_{i=1}^{j-1} c_{\pi(j)\pi(i)} p_{\pi(i)} p_{\pi(j)} \prod_{k=j+1}^n (1 - p_{\pi(k)}) \prod_{k=1}^{i-1} (1 - p_{\pi(k)}). \quad (1)
\end{aligned}$$

For the homogeneous PTSP, Equation 1 can be written as

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

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

The empirical estimation approach for the PTSP falls into the so-called sample average approximation (Kleywegt et al., 2002), which consists in estimating the cost $F(x)$ on the basis of sample costs $f(x, \omega_1), f(x, \omega_2), \dots, f(x, \omega_M)$ of *a posteriori* solutions obtained from M independent realizations $\omega_1, \omega_2, \dots, \omega_M$ of the random variable ω :

$$\hat{F}_M(x) = \frac{1}{M} \sum_{r=1}^M f(x, \omega_r). \quad (2)$$

As it can easily be shown, $\hat{F}_M(x)$ is an *unbiased* estimator of $F(x)$. Note that the number M of realizations is crucial for the effectiveness of this approach—we will revisit this issue in Section 3.2.

The development of metaheuristics to solve the PTSP has received considerable attention in recent years. This is in part due to the fact that the state-of-the-art exact technique, a branch and cut algorithm based on integer two-stage stochastic programming (Laporte et al., 1994), has solved to optimality only instances of size up to 50. Much of the early research in the development of metaheuristics for the PTSP focused on algorithms that use analytical computation. Bianchi et al. (2002a,b) proposed pACS, an ant colony system (ACS) (Dorigo and Gambardella, 1997) that adopts Equation 1 to compute the cost of solutions. Branke and Guntch (2003, 2004) and Liu (2007, 2008) used a truncated version of Equation 1 in pACS and in a scatter search algorithm, respectively. Bianchi (2006) and Bianchi and Gambardella (2007) integrated 1-shift, an analytical computation local search algorithm, into pACS and showed that the resulting pACS+1-shift algorithm is very effective. Recently, Marinakis and Marinaki (2009) proposed HybMSPSO, a particle swarm optimization algorithm built on the top of a PTSP-specific greedy randomized adaptive search procedure (Marinakis et al., 2008). HybMSPSO also adopts Equation 1 to compute the solution cost. The authors show that HybMSPSO obtains slightly better solutions than pACS+1-shift. However, there are 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., 2006; Birattari, 2009).

Concerning estimation-based algorithms, Gutjahr (2003, 2004) proposed a general purpose, estimation-based ACO algorithm called S-ACO and a variant S-ACOA. While in S-ACO, the number of realizations needed for cost estimation is increased linearly in dependence of the iteration number, in S-ACOA, the number of realizations is determined based on a statistical test. Gutjahr used the PTSP to calibrate the algorithm parameters. 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 on-line based on the F-Race procedure (Birattari et al., 2002; Birattari, 2009). Recently, in Balaprakash et al. (2009b), we extended ACS with our effective estimation-based iterative improvement algorithm, `2.5-opt-EEais` (Balaprakash et al., 2009a) and an ANOVA-Race procedure. This procedure is based on a parametric statistical test for multiple comparisons to determine the number of realizations. We showed that the estimation-based ACS+`2.5-opt-EEais` is more effective than pACS+1-shift.

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. In Balaprakash et al. (2009a), we implemented a simple iterated local search algorithm that adopts the state-of-the-art iterative improvement algorithm, `2.5-opt-EEais`, as local search. This algorithm is primarily used to show that the advantage of using `2.5-opt-EEais` over 1-shift remains once they are included into a metaheuristic. Therefore, the algorithm is not compared to current state-of-the-art algorithms. Note that the iterated local search that we discuss in this paper is an improved variant of the one proposed in Balaprakash et al. (2009a).

To summarize, ACO dominates the literature with pACS+1-shift and estimation-based ACS being the best available metaheuristics for the analytical computation approach and the estimation-based approach, respectively. Although the effectiveness of HybMSPSO is not quite clear, high quality solutions have been reported. Therefore, we consider it as a state-of-the-art algorithm for the PTSP and we include it in our analysis.

3 Estimation-based approach

In this section, first we summarize the `2.5-opt-EEais` algorithm, which is used as the underlying local search heuristic for all metaheuristics. Then, we briefly describe the implemented metaheuristics and we highlight the customizations performed to tackle the PTSP.

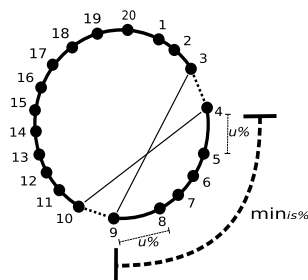


Figure 2: In this example, the two edges $\langle 3, 4 \rangle$ and $\langle 9, 10 \rangle$ are deleted and replaced with $\langle 3, 9 \rangle$ and $\langle 4, 10 \rangle$ by a 2-exchange move. Assume that min_{is} and u are both set to 40. Since the number of nodes in the segment $[4, \dots, 9]$ is less than 40% of 20 (that is, eight), importance sampling is used to bias 40% of 6 (that is, two) nodes between 4 and 9 on each end of the segment. The nodes biased are 4, 5, 8 and 9.

3.1 The 2.5-opt-EEais algorithm

2.5-opt-EEais (Balaprakash et al., 2009a) is a state-of-the-art iterative improvement algorithm for the PTSP. It adopts the 2.5-exchange neighborhood, which combines the 2-exchange and the node-insertion neighborhoods (Bentley, 1992). The cost differences between neighboring solutions are computed using delta evaluation based on an estimation approach that includes three variance reduction techniques, namely method of common random numbers, adaptive sample size, and importance sampling (Rubinstein, 1981). The algorithm has three parameters that affect the PTSP-specific importance sampling procedure. This procedure, which is crucial for tackling instances with very low probability values, works as follows. In 2-exchange moves, when the number of nodes in one of the two segments is less than $min_{is}\%$ of the instance size, $u\%$ nodes of the shorter segment are biased to probability p' . See Figure 2 for an example, where the shorter of the two segments is marked by $min_{is}\%$. In node-insertion moves, the insertion node is biased to probability value p'' . Whenever a node i is biased, the delta evaluation procedure ignores realizations sampled with the original probability p_i and considers instead realizations sampled with the biased probability that is larger than p_i . The cost difference estimate obtained in this way is then corrected for the artificial bias using the likelihood ratio. The effectiveness of this algorithm is also due to the adoption of the following neighborhood reduction techniques: fixed-radius search, candidate lists, and don't look bits (Bentley, 1992; Johnson and McGeoch, 1997). For more details, we refer the reader to Balaprakash et al. (2009a).

3.2 Estimation-based metaheuristics

A straightforward approach to make a metaheuristic estimation-based is to estimate the cost of solutions using Equation 2. An important issue in using the estimation-based approach within a metaheuristic is determining if one solution

is better than another: since the cost of a solution is estimated, there is an inherent variance associated with the cost estimate. To reduce the variance of the cost estimate, we adopt two variance reduction techniques: (i) the method of common random numbers and (ii) an adaptive sample size procedure.

The method of common random numbers is one of the most widely used techniques to reduce the variance of an estimator. For the PTSP, this technique uses a same set of realizations to evaluate the cost of solutions produced at each iteration of a metaheuristic.

The estimation problem becomes crucial for PTSP instances with low probability values. This is due to the fact that the estimator of the cost of solutions has a very high coefficient of variation: with respect to the expected value, the variance is very high. In this case, the adoption of a large number of realizations improves the estimation because the variance of the estimator decreases with $O(1/\sqrt{M})$, where M is the number of realizations. However, for instances with high probability values, the use of a large number of realizations results in a waste of computation time. This issue can be addressed by using an adaptive sample size procedure, which selects the most appropriate number of realizations for each estimation with respect to the variance. This adaptive sample size procedure can be realized using Student's t-test: Given two solutions, the cost estimate of each solution is computed on a realization-by-realization basis. As soon as the t-test rejects the null hypothesis that the cost estimates of the two solutions are equal, the computation is stopped. If no statistical evidence is gathered, the computation is continued until a maximum number M of realizations, where M is a parameter of the procedure. Finally, the solution with the lower cost estimate is selected as the best. Note that Gutjahr (2004) used a very similar procedure within S-ACOa.

The aforementioned adaptive sample size procedure can be adopted easily to compare two solutions in algorithms such as iterated local search (ILS). However, in memetic algorithms (MAs), a set of solutions needs to be compared at each iteration. For this purpose, we use ANOVA-Race, which we developed for the estimation-based ACS. ANOVA-Race is a racing algorithm based on analysis of variance (ANOVA) (Fisher, 1925), which is implemented as follows: a given set of solutions is sequentially evaluated on a number of realizations. The ANOVA test is used to gather a statistical evidence that the cost estimate of a solution is worse than at least another one. As soon as this evidence is obtained, the inferior solution is discarded and not considered for further evaluation. The inferior solution is identified using Tukey's honestly significant differences test (Tukey, 1949). The procedure stops when either one single solution remains, or when any of the surviving candidate solutions is evaluated on a maximum number M of realizations. If more than one solution survives the race, the solution with the least cost estimate is selected as the best one.

3.2.1 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. 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.

3.2.2 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. 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 (Bianchi et al., 2005; Bianchi and Campbell, 2007; Balaprakash et al., 2009a; Birattari et al., 2008). 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.

3.2.3 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. 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 heuristics 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_{ab} and ps (see Section 3.2.2) 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.

3.2.4 Ant colony optimization

As an ACO algorithm for the PTSP, we choose ACS-EE (Balaprakash et al., 2009b), a recent state-of-the-art ACO algorithm for the PTSP. It is an extension of the ant colony system (ACS) algorithm (Dorigo and Gambardella, 1997) designed for the TSP. 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 the node j that maximizes the product $\tau_{ij}\eta_{ij}^\beta$; otherwise the 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.

4 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 meta-heuristics and to compare them to the state-of-the-art analytical computation algorithms for the PTSP.

4.1 Experimental setup

The PTSP instances used for the experiments are obtained as follows: first we generated TSP instances with the DIMACS instance generator (Johnson et al., 2001); from these TSP instances, PTSP instances are obtained by associating a probability value to each node using a beta distribution as described by 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 ω 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 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 our previous study on `2.5-opt-EEais` (Balaprakash et al., 2009a), 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 OpteronTM244 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 one thousand 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 (Birattari et al., 2008). However, the realizations are selected randomly from the given set at each iteration. Equation 1 is used for the post-evaluation

Table 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. For an explanation of the parameters, see Section 3.2.

algorithm	parameters	range	selected value		
			Class-I	Class-II	Class-III
2.5-opt-EEais	$min_{i,s}$	[0.0, 50.0]	42.0	46.0	2.40
	w	[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

of the best-so-far solutions found by all estimation-based metaheuristics.

Due to space limitations, 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 and the complete results can be inspected in Balaprakash et al. (2008).

4.2 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 a parameter tuning algorithm, Iterative F-Race (Balaprakash et al., 2007). 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 Table 1.

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. (2008).

4.3 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

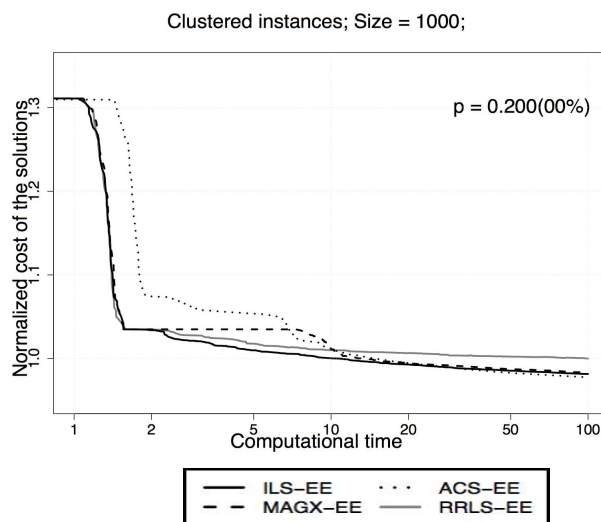


Figure 3: An exemplary run time development plot on clustered PTSP instances of 1000 nodes for 100 CPU seconds. The plot represents the cost of the solutions obtained by ILS-EE, MAGX-EE, ACS-EE, and RRLS-EE. The obtained solution costs of the algorithms are normalized by the final solution cost reached by RRLS-EE. The normalization is done on an instance-by-instance basis for 50 instances; the normalized solution cost is then aggregated.

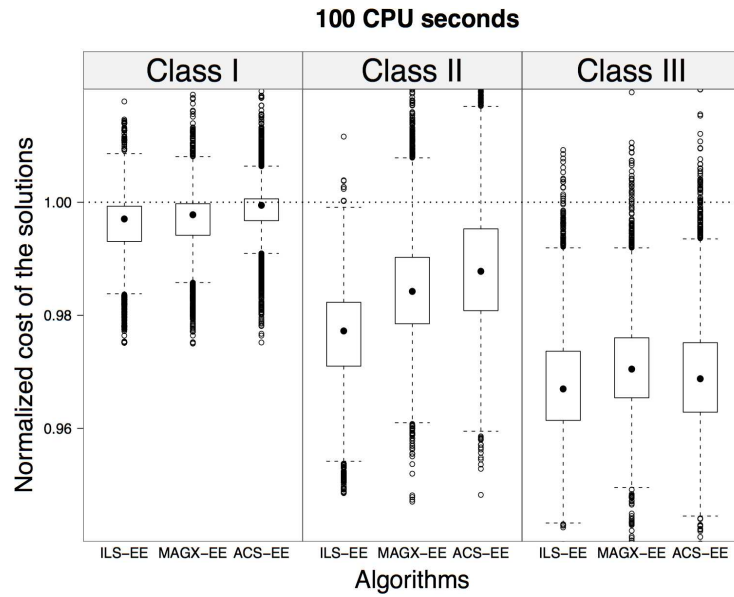
parameter configurations and the set of all test instances. In order to group the results obtained on different instances on each instance class, the cost of the solutions obtained by ILS-EE, MAGX-EE, and ACS-EE are normalized by the final solution cost reached by RRLS-EE. The normalization is done on an instance-by-instance basis for 50 instances for each probability level.

Figure 3 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 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. (2008). The improvements 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.

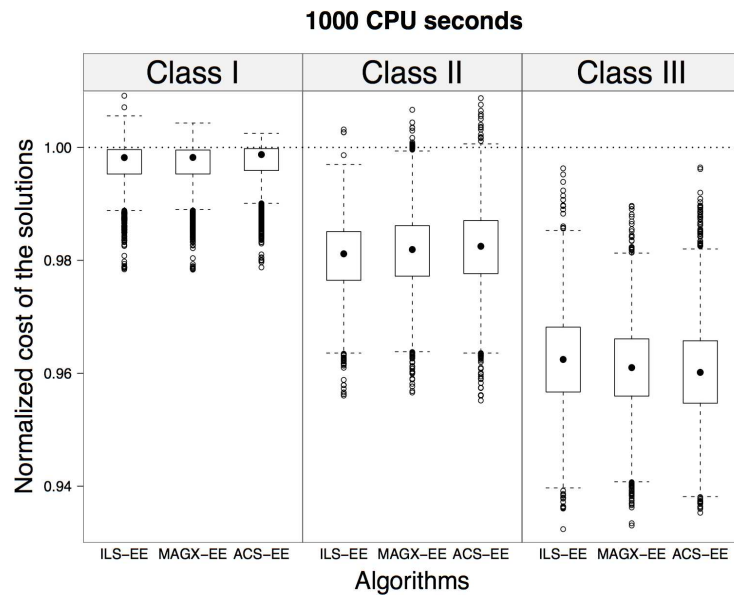
Figure 4 shows the box plot of the solution cost of the algorithms after 100 and 1000 CPU seconds. The solution cost of RRLS-EE is significantly worse than that of all other algorithms across most probability levels. An interesting observation from the plots is that the difference in the solution cost 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 initial 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 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. Considering the relatively short computation time, the effectiveness of ILS-EE can be explained as follows: since `2.5-opt-EEais` is applied on a single solution at each iteration unlike MAGX-EE and ACS-EE, ILS-EE can do more iterations and exploit better the best solution found so far.

The results for 1000 CPU seconds are given in Table 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.



(a)



(b)

Figure 4: 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.

Table 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	
		<i>p</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>	<i>d</i>
			CI	CI	CI	CI	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]	<i>+0.19</i>	[+0.14, +0.24]
	0.300(16%)	-0.17	[-0.23, -0.11]	-0.03	[-0.09, +0.03]	<i>+0.13</i>	[+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]	<i>+0.16</i>	[+0.09, +0.22]	<i>+0.32</i>	[+0.27, +0.37]
	0.400(16%)	-0.18	[-0.25, -0.10]	<i>+0.15</i>	[+0.08, +0.23]	<i>+0.33</i>	[+0.28, +0.38]
	0.400(50%)	-0.29	[-0.36, -0.21]	-0.14	[-0.22, -0.07]	<i>+0.15</i>	[+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]	<i>+0.19</i>	[+0.11, +0.26]	<i>+0.37</i>	[+0.32, +0.42]
	0.500(16%)	-0.20	[-0.28, -0.13]	<i>+0.21</i>	[+0.13, +0.28]	<i>+0.41</i>	[+0.36, +0.46]
0.500(50%)	-0.14	[-0.22, -0.06]	<i>+0.23</i>	[+0.15, +0.31]	<i>+0.37</i>	[+0.31, +0.43]	
0.500(83%)	-0.54	[-0.62, -0.46]	-0.28	[-0.36, -0.20]	<i>+0.27</i>	[+0.20, +0.33]	
overall		-0.31	[-0.33, -0.29]	-0.19	[-0.21, -0.16]	<i>+0.12</i>	[+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 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 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 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 3.2.

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

4.4 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, 2009). 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 the estimation-based algorithms, we choose the parameter values that produced the lowest average cost across the 10 tuning runs for 1000 CPU seconds in Section 4.3. They are listed in Table 4.

A problem in pACS+1-shift is that the underlying local search, 1-shift, suffers from numerical precision problems for large instances (Birattari et al., 2008). This problem can be addressed by resorting to computationally expensive arbitrary precision arithmetics (Fousse et al., 2007). For the given stopping criterion, the usage of the arbitrary precision arithmetics in pACS+1-shift

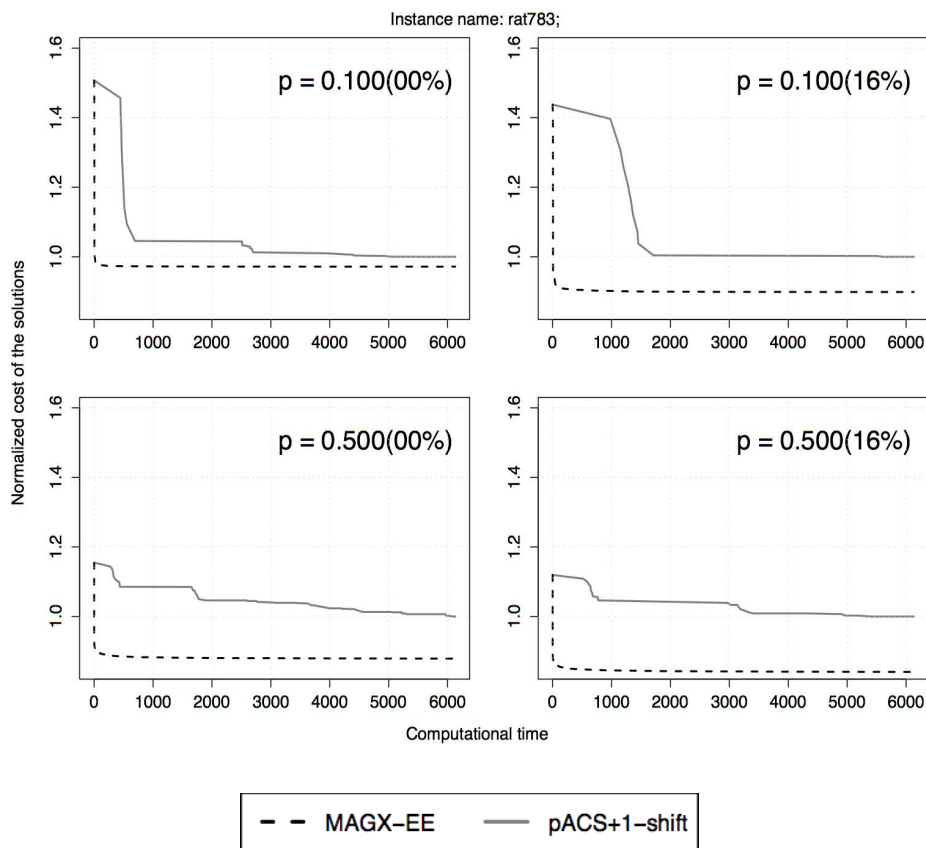


Figure 5: Experimental results on the instance `rat783`. The plots represent the development of the solution cost over time for MAGX-EE and pACS+1-shift. The obtained solution costs of the two algorithms are normalized by the final solution cost reached by pACS+1-shift. The normalization is performed on a run-by-run basis for 10 runs; the normalized solution cost is then aggregated.

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 5 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. (2008).

Table 5 reports the average difference between the final solution costs ob-

tained 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. (2008). 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 improvements 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.

For the comparison between the estimation-based algorithms and HybMSPSO, we adopt the instances used by Marinakis and Marinaki (2009), 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 (2009). 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 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. (2008).

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%. 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 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 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: 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 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 (2009). Typographic conventions are the same as in Table 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]	-0.43	[+0.00, +0.00]		
	0.200(00%)	-0.07	[-0.10, -0.05]	-0.08	[-0.10, -0.07]	-0.09	[-0.09, -0.09]		
	0.300(00%)	<i>+0.06</i>	[+0.03, +0.10]	+0.01	[-0.01, +0.04]	<i>+0.01</i>	[+0.00, +0.00]		
	0.400(00%)	-0.92	[-0.99, -0.85]	-1.00	[-1.00, -0.99]	-1.00	[+0.00, +0.00]		
	0.500(00%)	-0.03	[-0.10, +0.04]	-0.08	[-0.08, -0.07]	-0.08	[+0.00, +0.00]		
	0.600(00%)	-1.91	[-1.92, -1.90]	-1.91	[-1.92, -1.91]	-1.92	[+0.00, +0.00]		
	0.800(00%)	-2.57	[-2.58, -2.56]	-2.58	[-2.59, -2.57]	-2.58	[+0.00, +0.00]		
	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]	-0.80	[+0.00, +0.00]		
	0.200(00%)	-0.97	[-1.01, -0.94]	-1.03	[-1.06, -1.01]	-0.99	[-0.99, -0.99]		
	0.500(00%)	-0.83	[-0.86, -0.79]	-0.85	[-0.87, -0.82]	-0.88	[+0.00, +0.00]		
	0.900(00%)	-0.02	[-0.05, +0.01]	-0.07	[-0.09, -0.04]	<i>+0.00</i>	[+0.00, +0.00]		
rat783	0.100(00%)	-10.14	[-10.24, -10.05]	-10.23	[-10.32, -10.15]	-10.10	[+0.00, +0.00]		
	0.200(00%)	-4.60	[-4.77, -4.42]	-4.64	[-4.75, -4.54]	-4.29	[-4.30, -4.29]		
	0.500(00%)	-3.35	[-3.47, -3.23]	-3.51	[-3.61, -3.42]	-3.39	[-3.40, -3.39]		
	0.900(00%)	-0.32	[-0.38, -0.26]	-0.32	[-0.52, -0.12]	-0.38	[+0.00, +0.00]		

Table 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 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]	-0.48	[+0.00, +0.00]
	0.200(00%)	-0.41	[-0.43, -0.38]	-0.41	[-0.43, -0.40]	-0.42	[-0.43, -0.42]
	0.300(00%)	-0.22	[-0.26, -0.18]	-0.27	[-0.30, -0.24]	-0.28	[+0.00, +0.00]
	0.400(00%)	-0.08	[-0.15, -0.01]	-0.16	[-0.16, -0.15]	-0.16	[+0.00, +0.00]
	0.500(00%)	-0.05	[-0.12, +0.02]	-0.09	[-0.09, -0.09]	-0.09	[+0.00, +0.00]
	0.600(00%)	-0.07	[-0.08, -0.06]	-0.07	[-0.08, -0.07]	-0.08	[+0.00, +0.00]
	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]	-0.04	[+0.00, +0.00]
	0.900(00%)	-0.00	[+0.00, +0.00]	-0.00	[+0.00, +0.00]	-0.00	[+0.00, +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]	-1.06	[+0.00, +0.00]
	0.200(00%)	-1.22	[-1.26, -1.19]	-1.28	[-1.31, -1.26]	-1.24	[-1.24, -1.24]
	0.300(00%)	-1.44	[-1.47, -1.40]	-1.48	[-1.51, -1.45]	-1.51	[-1.52, -1.51]
	0.400(00%)	-1.25	[-1.28, -1.21]	-1.28	[-1.30, -1.25]	-1.32	[-1.32, -1.31]
	0.500(00%)	-0.92	[-0.95, -0.88]	-0.94	[-0.96, -0.91]	-0.97	[+0.00, +0.00]
	0.600(00%)	-0.51	[-0.57, -0.44]	-0.54	[-0.59, -0.50]	-0.57	[+0.00, +0.00]
	0.700(00%)	-0.41	[-0.45, -0.38]	-0.47	[-0.47, -0.47]	-0.47	[+0.00, +0.00]
	0.800(00%)	-0.22	[-0.27, -0.18]	-0.27	[-0.27, -0.27]	-0.27	[+0.00, +0.00]
	0.900(00%)	-0.08	[-0.11, -0.05]	-0.12	[-0.14, -0.10]	-0.05	[+0.00, +0.00]
	1.000(00%)	$+0.05$	[+0.03, +0.07]	$+0.03$	[+0.01, +0.05]	$+0.03$	[+0.00, +0.00]
rat783	0.100(00%)	-10.18	[-10.27, -10.09]	-10.27	[-10.36, -10.19]	-10.14	[+0.00, +0.00]
	0.200(00%)	-8.39	[-8.57, -8.22]	-8.44	[-8.54, -8.34]	-8.10	[-8.11, -8.10]
	0.300(00%)	-6.25	[-6.34, -6.15]	-6.32	[-6.46, -6.17]	-6.69	[-6.70, -6.69]
	0.400(00%)	-4.72	[-4.85, -4.58]	-4.64	[-4.78, -4.49]	-4.64	[-4.64, -4.63]
	0.500(00%)	-3.39	[-3.51, -3.27]	-3.56	[-3.65, -3.46]	-3.43	[-3.44, -3.43]
	0.600(00%)	-2.10	[-2.21, -1.99]	-2.26	[-2.38, -2.13]	-2.25	[+0.00, +0.00]
	0.700(00%)	-1.17	[-1.32, -1.03]	-1.45	[-1.59, -1.32]	-1.29	[-1.29, -1.28]
	0.800(00%)	-0.81	[-0.90, -0.72]	-0.82	[-0.93, -0.71]	-0.93	[+0.00, +0.00]
	0.900(00%)	-0.56	[-0.62, -0.50]	-0.56	[-0.76, -0.37]	-0.62	[+0.00, +0.00]
	1.000(00%)	$+0.81$	[+0.71, +0.91]	$+0.76$	[+0.61, +0.91]	$+0.88$	[+0.87, +0.88]

4.5 PTSP-specific algorithms vs. TSP approximation

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 8. The estimation-based algorithms ILS-

EE, ACS-EE, and MAGX-EE obtain solution costs, which are significantly less than that 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`.

5 Conclusions

In this paper, we customized iterated local search and memetic 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, in which we also included a recently developed estimation-based ant colony system. We used short and long computation times as stopping criteria. First, we performed a rigorous parameter tuning 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 highly effective when compared to the previous estimation-based ant colony system. On the other hand, for long computation time, both the iterated local search and the memetic algorithm outperform the ant colony system on instances with low average probability values (up to 0.2). Nevertheless, the 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 paper is the development of new state-of-the-art algorithms for the PTSP. The estimation-based algorithms are particularly effective for large instances. Compared to the best analytical computation algorithms, the proposed 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 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.

Acknowledgments

The authors thank Leonora Bianchi for providing the source code of `pACS+1-shift`. This research has been supported by `COMP2SYS`, 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`, which are ARC projects funded by the French Community of Belgium. The authors acknowledge support from the fund for scientific research `F.R.S.-FNRS`

of the French Community of Belgium.

References

- 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>.
- P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In T. Bartz-Beielstein, M. Blesa, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, editors, *Hybrid Metaheuristics, HM 2007*, volume 4771 of *LNCS*, pages 113–127, Berlin, Germany, 2007. Springer Verlag.
- P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Estimation-based metaheuristics for the probabilistic traveling salesman problem. IRIDIA Supplementary page, 2008. URL <http://iridia.ulb.ac.be/supp/IridiaSupp2008-019/>.
- 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, 2009a.
- 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), 2009b.
- J. L. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.
- D. Bertsimas. *Probabilistic Combinatorial Optimization Problems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1988.
- D. Bertsimas, P. Jaillet, and A. Odoni. A priori optimization. *Operations Research*, 38(6):1019–1033, 1990.
- 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.-P. 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(1):206–219, 2005.
- M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*, volume 197 of *Studies in Computational Intelligence*. Springer Verlag, Berlin, Germany, 2009.
- M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In W. B. Langdon et. al, editor, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002*, 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, 2005.
- 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, 2006.
- 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.
- 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. 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.
- 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.
- R. A. Fisher. *Statistical Methods for Research Workers*. Oliver and Boyd, London, UK, 1925.
- L. Fousse, G. Hanrot, V. Lefèvre, P. Pélicier, 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/>.
- W. J. Gutjahr. A converging ACO algorithm for stochastic combinatorial optimization. In A. Albrecht and K. Steinhof, 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.
- H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco, CA, 2005.
- P. Jaillet. *Probabilistic Traveling Salesman Problems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1985.
- 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, United Kingdom, 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/>.

- 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.
- G. Laporte, F. Louveaux, and H. Mercure. A priori optimization of the probabilistic traveling salesman problem. *Operations Research*, 42:543–549, 1994.
- 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.
- H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research and Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.
- Y. Marinakis and M. Marinaki. A hybrid multi-swarm particle swarm optimization algorithm for the probabilistic traveling salesman problem. *Computers and Operations Research*, (in press), 2009.
- Y. Marinakis, A. Migdalas, and P. Pardalos. Expanding neighborhood search–GRASP for the probabilistic traveling salesman problem. *Optimization Letters*, 2(3):351–361, 2008.
- P. Merz and B. Freisleben. Memetic algorithms for the traveling salesman problem. *Complex Systems*, 13(4):297–345, 2001.
- P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. C3P Report 826, Caltech Concurrent Computation Program, 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.
- 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.
- R. Y. Rubinstein. *Simulation and the Monte Carlo Method*. John Wiley & Sons, New York, NY, 1981.
- 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/>.
- J. W. Tukey. Comparing individual means in the analysis of variance. *Biometrics*, 5(2):99–114, 1949.