Swarm Intell (2009) 3: 223–242
DOI 10.1007/s11721-009-0031-y

Estimation-based ant colony optimization and local search for the probabilistic traveling salesman problem

Prasanna Balaprakash · Mauro Birattari · Thomas Stützle · Zhi Yuan · Marco Dorigo

Received: 13 October 2008 / Accepted: 10 May 2009 / Published online: 4 June 2009

Abstract The use of ant colony optimization for solving stochastic optimization problems has received a significant amount of attention in recent years. In this paper, we present a study of enhanced ant colony optimization algorithms for tackling a stochastic optimization problem, the PROBABILISTIC TRAVELING SALESMAN PROBLEM. In particular, we propose an empirical estimation approach to evaluate the cost of the solutions constructed by the ants. Moreover, we use a recent estimation-based iterative improvement algorithm as a local search. Experimental results on a large number of problem instances show that the proposed ant colony optimization algorithms outperform the current best algorithm tailored to solve the given problem, which also happened to be an ant colony optimization algorithm. As a consequence, we have obtained a new state-of-the-art ant colony optimization algorithm for the PROBABILISTIC TRAVELING SALESMAN PROBLEM.

Keywords Ant colony optimization · Empirical estimation · Estimation-based local search · Probabilistic traveling salesman problem

1 Introduction

Ant colony optimization (ACO) (Dorigo and Stützle 2004) has become a very successful and widely used swarm intelligence method (Bonabeau et al. 1999; Dorigo and Birattari 2007)

P. Balaprakash (✉) · M. Birattari · T. Stützle · Z. Yuan · M. Dorigo
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
e-mail: pbalapra@ulb.ac.be

M. Birattari
e-mail: mbiro@ulb.ac.be

T. Stützle
e-mail: stuetzle@ulb.ac.be

Z. Yuan
e-mail: zyuan@ulb.ac.be

M. Dorigo
e-mail: mdorigo@ulb.ac.be

for solving hard optimization problems. Its success has been proved not only by the large number of problems to which it has been applied, but also by the very good performance ACO algorithms have achieved in many fields, especially for routing problems with complex features, such as stochastic information, or time-varying data (Caro and Dorigo 1998).

In this paper, we study the application of ACO algorithms to the PROBABILISTIC TRAVELING SALESMAN PROBLEM (PTSP) (Jaillet 1985), which is also known as the traveling salesman problem with stochastic customers (Gendreau et al. 1996). It is a stochastic extension of the classical traveling salesman problem (TSP). In the PTSP, it is unknown in advance whether a node requires being visited, but its probability of requiring a visit is given. The most widely used approach to tackle the PTSP is to construct an a priori solution before knowing which nodes require being visited. An a priori solution is a permutation of all the nodes of the given instance. Once the set of nodes that require being visited is known, the a posteriori solution is derived by visiting the nodes that require being visited in the order prescribed by the a priori solution and by skipping the nodes that do not require being visited. The objective of the PTSP is to find an a priori solution, such that the expected cost of its associated a posteriori solution is minimized.

The PTSP is an $\mathcal{NP}$-hard problem. The stochastic nature and the complexity of the problem limit the applicability of exact algorithms. The so far best performing exact solution technique was proposed by Laporte et al. (1994), who formulated the problem as an integer linear stochastic program and solved it by a branch-and-cut approach. The experimental results showed that instances of size up to 50 can be solved to optimality.

Recent approaches to the PTSP mainly involve the application of stochastic local search (SLS) methods (Hoos and Stützle 2005), among which ACO algorithms appear to be currently the best-performing. SLS methods for the PTSP can be grouped into two main classes: *analytical computation* and *empirical estimation*. In analytical computation algorithms, the exact cost of an a priori solution is computed using a closed-form expression derived by Jaillet (1985). A prominent subclass of analytical computation algorithms is analytical approximation, where the cost of the a priori solution is approximated using a truncated version of the closed-form expression. In empirical estimation algorithms, the cost of the a priori solution is estimated by Monte Carlo simulation.

Much of the early ACO algorithms for the PTSP are based on analytical computation. Bianchi et al. (2002a, 2002b) adopted the closed-form expression in an ant colony system (ACS) and compared it with a version of ACS for the TSP. The preliminary results showed that the PTSP-specific approach is more effective than its TSP counterpart when the instance probability values are less than 0.5. Branke and Guntsch (2004) explored the idea to employ an ad-hoc approximation to replace the exact PTSP objective function, and showed that the computation time can be significantly reduced without major loss in solution quality. Bianchi (2006) and Bianchi and Gambardella (2007) proposed `pACS+1-shift`, which integrates the PTSP-specific ACS with `1-shift` (Bertsimas and Howell 1993; Bianchi et al. 2005), a local search tailored for the PTSP. The experimental results showed that `pACS+1-shift` significantly outperforms all other algorithms proposed so far in the literature, and it is up to now considered as the best-performing SLS method for the PTSP.

Gutjahr (2003, 2004) proposed S-ACO, a general-purpose estimation-based ACO algorithm for tackling stochastic combinatorial optimization problems, and took the PTSP as a test bed. 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 that increases with the number of iterations according to a static scheme. Gutjahr (2004) also studied a variant of S-ACO called S-ACOa, in which the number of realizations needed for comparing the iteration-best with

the best-so-far solution is determined dynamically based on a statistical test. Birattari et al. (2006) proposed ACO/F-Race, which adopts the F-Race procedure (Birattari 2004, 2009). In this algorithm, 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 the 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 not typically 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 estimation-based solution evaluation approach 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`.

In our recent research on the PTSP, we first developed `2.5-opt-EEs` (Birattari et al. 2008), a new local search algorithm that uses an estimation-based approach to speed up the cost difference computation between neighbor solutions. `2.5-opt-EEs` reaches significantly better solutions for a wide range of instances of size from 100 to 1000 nodes and it is by two to three orders of magnitude faster than `1-shift`. Only on instances with low probability values, `2.5-opt-EEs` showed a slightly worse performance than `1-shift`. To address this issue, we integrated two variance reduction techniques, namely adaptive sample size and importance sampling into `2.5-opt-EEs`, obtaining in this way the new algorithm `2.5-opt-EEais` (Balaprakash et al. 2009), which fully outperforms `1-shift`. To test whether the observed behavior extends to SLS methods, we performed some preliminary experiments with a simple iterated local search algorithm that uses `2.5-opt-EEais` or an improved variant of `1-shift`. The results showed that the iterated local search algorithm with `2.5-opt-EEais` is very effective with respect to both solution quality and computation time (Balaprakash et al. 2009). These results indicate that there is also a significant potential to improve over `pACS+1-shift`. The aforementioned factors motivated us to develop a new algorithm that adopts the estimation-based approach in the ACO framework, with the goal of effectively solving the PTSP.

In order to assess the impact of each algorithmic component that we use, we adopt the following systematic bottom-up design: We use `pACS+1-shift` as a starting point; in Sect. 4, we replace `1-shift` with `2.5-opt-EEais` in `pACS` and we show that `pACS+2.5-opt-EEais` outperforms `pACS+1-shift`; in Sect. 5, 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 Sect. 6, 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. It should be noted that all the four estimation-based ACO variants outperform the previously best ACO algorithm, `pACS+1-shift`.

2 The probabilistic traveling salesman problem

A PTSP instance can be defined on a fully connected weighted graph G with $V = \{1, 2, \ldots, n\}$ being a set of nodes and $P = \{p_i : i \in V\}$ being a set of probabilities, where p_i

is the probability that node i requires being visited. The events that two distinct nodes i and j require being visited are assumed to be independent. We denote $\langle i, j\rangle$ the edge connecting two distinct nodes i and j and c_{ij} the travel cost imposed on this edge. The travel costs are assumed to be symmetric—for all pairs of nodes i, j we have $c_{ij} = c_{ji}$. The probabilistic information 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 need a visit. A PTSP instance is called homogeneous if all probabilities in the set P are the same, and it is called heterogeneous otherwise.

The PTSP is usually tackled by the a priori optimization approach in two stages. First, before the realization of ω is known, a Hamiltonian tour containing all the nodes is constructed, which is called an a priori solution; once the nodes that require being visited are known, the a posteriori solution is obtained by following the nodes in the order of the a priori solution and by excluding the nodes that need not be visited. See Fig. 1 for an illustration of a priori and a posteriori solutions. The goal is to find an a priori solution with the minimum expected a posteriori solution cost.

Suppose $x = (\pi(1), \pi(2), \ldots, \pi(n), \pi(n+1) = \pi(1))$ is an a priori solution for the PTSP, where π is a permutation of the set V. The analytical computation approach for evaluating the expected cost $F(x)$ of the a priori solution x uses the following closed-form expression (Jaillet 1985):

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

Note that for a homogenous instance with probability p and size n, (1) reduces to $F(x) = \sum_{i=1}^{n} \sum_{j=1}^{n-1} p^2 (1-p)^{j-1}\ c_{\pi(i),\pi(1+((i+j-1) \bmod n))}$.

In the empirical estimation approach for the PTSP, the cost $F(x)$ is empirically estimated on the basis of sample costs of a posteriori solutions $f(x, \omega_1), f(x, \omega_2), \ldots, f(x, \omega_M)$ ob-

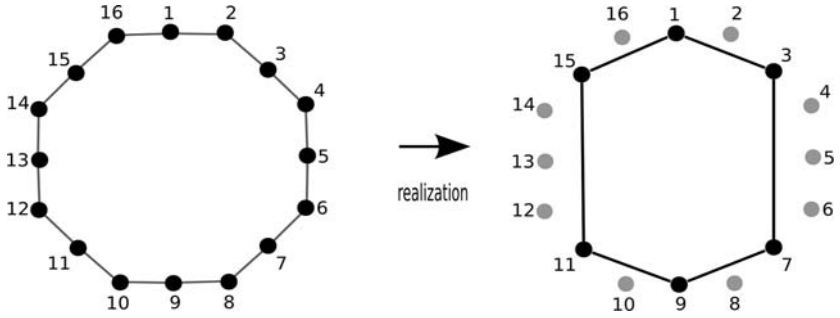

Fig. 1 An a priori solution for a PTSP instance with 16 nodes, where the nodes are visited in the following order: 1, 2, 3, ..., 15, 16, and 1. Let us assume that the nodes 1, 3, 7, 9, 11, and 15 are prescribed to be visited by a realization of $\boldsymbol{\omega}$. The a posteriori solution visits the nodes following the a priori solution but skipping the gray nodes that do not require to be visited

tained from M independent realizations $\omega_1, \omega_2, \ldots, \omega_M$ of the random variable $\boldsymbol{\omega}$:

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

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

3 The `pACS+1-shift` algorithm

`pACS+1-shift` (Bianchi 2006; Bianchi and Gambardella 2007) is currently the best performing ant colony optimization algorithm for the PTSP. It is a standard ACS algorithm (Dorigo and Gambardella 1997) in which, at each iteration, m ants construct solutions in the following way: With a probability q_0, ant k at node i chooses to move to the node j that maximizes the product $\tau_{ij}\eta_{ij}^{\beta}$; with probability $1 - q_0$, the next node j is chosen with probability $p_{ij}^k = \tau_{ij}\eta_{ij}^{\beta} / \sum_{l \in N_i^k} \tau_{il}\eta_{il}^{\beta}$ (the random proportional rule); 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 nodes for which it is feasible to move from node i. When an ant moves from node i to node j, the pheromone value associated with edge $\langle i, j \rangle$ is updated to $\tau_{ij} = (1 - \varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0$, where $\varphi \in (0, 1]$ is a parameter, and τ_0 is the initial value of the pheromone. 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 $\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij}^{\text{best}}$, where $\rho \in (0, 1]$ is a parameter and $\Delta\tau_{ij}^{\text{best}} = 1/C^{\text{best}}$. The value of C^{best} is set to the cost of the best-so-far solution. The solutions generated at each iteration are evaluated by (1).

`1-shift` local search, a PTSP-specific iterative improvement algorithm, is applied to all solutions constructed by the ants prior to the pheromone update. The algorithm proceeds in two phases: The first phase consists in exploring a swap-neighborhood, where the set of neighbors of a given solution contains all the solutions that can be obtained by swapping two consecutive nodes. The second phase explores the node-insertion neighborhood in a fixed lexicographic order. The cost difference of neighboring solutions is obtained by delta evaluation, a technique that considers only the cost contribution of solution components that are not common between the two solutions. This is done using computationally expensive closed-form expressions, which are based on complex mathematical derivations (Bianchi et al. 2005; Bianchi 2006; Bianchi and Campbell 2007).

4 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`.

`2.5-opt-EEais` (Balaprakash et al. 2009) is the state-of-the-art iterative improvement algorithm for the PTSP. `2.5-opt-EEais` differs from `1-shift` in the following three elements: It adopts an empirical estimation technique in the delta evaluation; it uses the 2.5-exchange neighborhood relation that combines the 2-exchange and node-insertion neighborhoods (Bentley 1992); and it exploits typical TSP neighborhood reduction techniques such as fixed-radius search, candidate lists, and don't look bits (Martin et al. 1991; Bentley 1992;

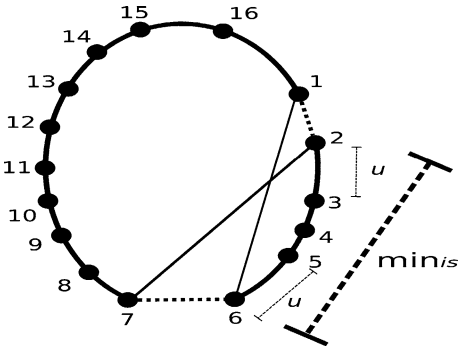

Fig. 2 In this example, the two edges $\langle 1, 2\rangle$ and $\langle 6, 7\rangle$ are deleted and replaced with $\langle 1, 6\rangle$ and $\langle 2, 7\rangle$ by a 2-exchange move. Assume that $\min_{\text{is}}$ and u are set to 50 and 40, respectively. Since the number of nodes in the segment $[2, \ldots, 6]$ is less than 50% of 16, which is eight, importance sampling is used to bias 40% of 5, that is, two nodes on *each end* of the segment $[2, \ldots, 6]$: on the end that starts with node 2, the biased nodes are 2 and 3; on the other end that starts with node 6, the biased nodes are 5 and 6

Johnson and McGeoch 1997). The effectiveness of the algorithm is further enhanced by the usage of variance reduction techniques such as the method of common random numbers, adaptive sample size, and importance sampling. In particular, importance sampling is essential for the algorithm to effectively tackle instances with probability values up to 0.2. Moreover, the adoption of this procedure is useful for instances with high probability values although it may result in slightly higher computation time when compared to an appropriate fixed sample size of 100 (Balaprakash et al. 2009). The importance sampling procedure is implemented as follows: In a 2-exchange move, whenever the number of nodes in the shorter segment (a 2-exchange move always leads to two tour segments) is less than $\min_{\text{is}}$ % of the instance size, u% nodes on *each end* of the shorter segment are biased with probability p'. See Fig. 2 for an example. Whenever a node i is biased with probability p', the delta evaluation procedure ignores realizations sampled with the original probability p_i and considers instead realizations in which the probability of node i requiring being visited is p'. The cost difference estimate obtained in this way is a biased one, which is then corrected to an unbiased one using the likelihood ratio. For the node-insertion move, only the insertion node is biased with a value p''. For a more detailed explanation of `2.5-opt-EEais`, we refer the reader to Balaprakash et al. (2009). We denote `pACS+2.5-opt-EEais` to be the algorithm obtained by combining `pACS` with `2.5-opt-EEais`.

We tuned the four parameters of `2.5-opt-EEais` through a parameter tuning algorithm, Iterative F-Race (Balaprakash et al. 2007). For tuning, we used homogeneous instances obtained as follows: TSP instances are generated with the DIMACS instance generator (Johnson et al. 2001) from which the PTSP instances are obtained by associating a probability value to each node. We used clustered instances of 1000 nodes, in which the nodes are arranged in a number of clusters in a $10^6 \times 10^6$ square. We considered values for p from 0.050 to 0.200 with increments of 0.025 and from 0.3 to 0.5 with increments of 0.1. We focus on probability values up to 0.5 because Bianchi and Gambardella (2007) showed that the instances with probability values greater than 0.5 can effectively be solved as a TSP by the concorde solver (Applegate et al. 2001). The generated instances are grouped into three classes according to the value of p: $\{0.050, 0.075, 0.100\}$ (Class-I), $\{0.150, 0.175, 0.200\}$ (Class-II), $\{0.300, 0.400, 0.500\}$ (Class-III); for each probability level we generated 30 instances. The parameters of `2.5-opt-EEais` are fine tuned on each instance class. The

Table 1 Parameters values for `2.5-opt-EEais`

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

Table 2 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. See footnote 1 for an explanation of the contents and of the typographic conventions adopted in the table

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]

initial solution for `2.5-opt-EEais` is generated using the nearest neighbor heuristic. Table 1 shows the range of each parameter given to the tuning algorithm and the selected value. For `pACS`, we adopted the parameter values suggested by Bianchi (2006), 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™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 2 and Fig. 3.

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`

[1]For a given comparison of algorithms A vs. B (in Table 2, algorithm A and B are `pACS+2.5-opt-EEais` and `pACS+1-shift`, respectively) the table reports the observed relative difference 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%.

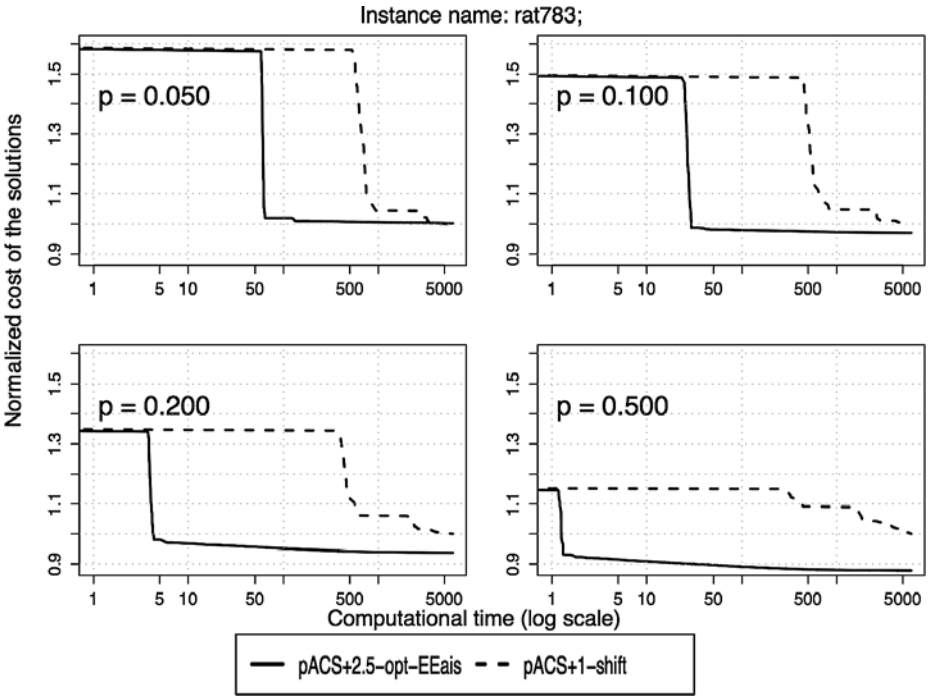

Fig. 3 Experimental results on the instance rat783. The plots represent the development of the solution cost over time for `pACS+2.5-opt-EEais` and `pACS+1-shift`. The obtained solution costs of the two algorithms are normalized by the final solution cost reached by `pACS+1-shift`. The normalization is done on a run-by-run basis for 30 runs; the normalized solution cost is then aggregated

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; we refer the reader to Balaprakash et al. (2008) for the complete set of results and for the absolute values.

5 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 (2) instead of (1). 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 $\boldsymbol{\omega}$. Estimating solution costs with low variance is crucial to the effectiveness of the estimation approach. As in `2.5-opt-EEais`, here we address this issue using two variance reduction techniques: (i) the method of common random numbers and (ii) an adaptive sample size.

As in `2.5-opt-EEais`, the adoption of the method of common random numbers for ACS consists in using the same set of realizations to evaluate the solutions produced at each iteration. The adaptive sample size in `2.5-opt-EEais` is implemented using the sequential application of a parametric statistical test, Student's t-test, which is appropriate for comparing two solutions. However, since in ACS more than two solutions are compared at

each iteration, we use a parametric statistical test based on analysis of variance (ANOVA) (Fisher 1925) and Tukey's honestly significant difference (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 Tukey's 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. Note that the aforementioned cost evaluation procedure takes place after the solutions constructed by the ants are improved by `2.5-opt-EEais`. 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 Sect. 4. 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` (Birattari et al. 2008; Balaprakash et al. 2009), 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 (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. (2008).

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. 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 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 3 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 coefficient of variation (Balaprakash et al.

Table 3 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 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	**−0.32** [−0.56, −0.09]	0.600	−0.02 [−0.12, +0.08]
0.700	**−0.49** [−0.76, −0.21]	0.700	−0.09 [−0.26, +0.08]
0.800	**−0.62** [−0.81, −0.43]	0.800	**−0.21** [−0.31, −0.10]
0.900	**−0.99** [−1.22, −0.77]	0.900	**−0.15** [−0.29, −0.02]

2009). 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 Sect. 4 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 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.

6 Comparison between estimation-based ACO algorithms

So far ACS is widely adopted to tackle the PTSP (Bianchi et al. 2002a, 2002b; 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: $\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{\text{best}}$, where ρ is a parameter and $\Delta\tau_{ij}^{\text{best}}$ is set to either $1/C^{\text{best}}$—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: $\tau_{ij} = \tau_{ij} + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{\text{best}}$, where r is the rank of the solution obtained by sorting m solutions by increasing cost, $\Delta\tau_{ij}^{r} = 1/C^{r}$, and $\Delta\tau_{ij}^{\text{best}} = 1/C^{\text{best}}$ 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.

6.1 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 4. 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. (2008).

6.2 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 Sect. 4 using Iterative F-Race.

Table 4 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 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]

The selected values are shown in Table 5. 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 6 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%.

6.3 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 7. For the absolute values, we refer the

Table 5 Fine-tuned parameter 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 6 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 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]

reader to Balaprakash et al. (2008). 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. (2008), we observed that for 1000 CPU seconds all the algorithms obtain average solution costs that are smaller than that of

Table 7 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 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 8 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 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 9 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 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]

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 8 and 9, 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 7 and 8. The general trend is similar to that of shorter computation times: There is no clear winner among the considered algorithms.

7 Conclusions and future work

The main contribution of this paper is the development and the empirical analysis of new state-of-the-art ACO algorithms for the PTSP. We used the current best performing ACO algorithm `pACS+1-shift` as a starting point. We showed that the adoption of the state-of-the-art 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. In a nutshell, we showed that the proposed estimation-based approach is an effective alternative to the analytical computation techniques when applying ACO and local search to the PTSP. Note that this conclusion contradicts the previous results reported in the ACO literature for the PTSP. 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.

Our future work consists in customizing effective TSP-specific SLS methods such as iterated local search, memetic algorithm and comparing them with the proposed estimation-based ACO algorithms. Further research effort will be devoted to design estimation-based SLS methods to solve stochastic vehicle routing problems. Another promising research direction is to investigate the application of estimation-based SLS methods to multi-objective stochastic routing problems.

Acknowledgements The authors thank Leonora Bianchi for providing the source code of `pACS+1-shift`. This research has been 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, 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, of which PB and ZY are research fellows, MB and TS are research associates, and MD is a research director.

References

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

Balaprakash, P., Birattari, M., & Stützle, T. (2007). 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, & M. Sampels (Eds.), *LNCS: Vol. 4771. Hybrid metaheuristics*, HM 2007 (pp. 113–127). Berlin: Springer.

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

Balaprakash, P., Birattari, M., Stützle, T., & Dorigo, M. (2009). 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.

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

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

Bianchi, L. (2006). *Ant colony optimization and local search for the probabilistic traveling salesman problem: a case study in stochastic combinatorial optimization*. Ph.D. thesis, Université Libre de Bruxelles, Brussels, Belgium.

Bianchi, L., & Campbell, A. (2007). 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.

Bianchi, L., & Gambardella, L. M. *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.

Bianchi, L., Gambardella, L., & Dorigo, M. (2002a). Solving the homogeneous probabilistic travelling salesman problem by the ACO metaheuristic. In M. Dorigo, G. Di Caro, & M. Sampels (Eds.), *LNCS: Vol. 2463. Ant algorithms, third international workshop*, ANTS 2002 (pp. 176–187). Berlin: Springer.

Bianchi, L., Gambardella, L. M., & Dorigo, M. (2002b). An ant colony optimization approach to the probabilistic traveling salesman problem. In J. J. Guervós, P. Adamidis, H. Beyer, J. L. Martín, & H. P. Schwefel (Eds.), *LNCS: Vol. 2439. 7th international conference on parallel problem solving from nature*, PPSN VII (pp. 883–892). Berlin: Springer.

Bianchi, L., Knowles, J., & Bowler, N. (2005). 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.

Birattari, M. (2004). *The problem of tuning metaheuristics as seen from a machine learning perspective*. Ph.D. thesis, Université Libre de Bruxelles, Brussels, Belgium.

Birattari, M. (2009). *Tuning metaheuristics: a machine learning perspective*. *Studies in computational intelligence* (Vol. 197). Berlin: Springer.

Birattari, M., Balaprakash, P., & Dorigo, M. (2006). The ACO/F-RACE algorithm for combinatorial optimization under uncertainty. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. J. Gutjahr, R. F. Hartl, & M. Reimann (Eds.), *Operations research/computer science interfaces series: Vol. 44. Metaheuristics—progress in complex systems optimization* (pp. 189–203). Berlin: Springer.

Birattari, M., Balaprakash, P., Stützle, T., & Dorigo, M. (2008). 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.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems*. London: Oxford University Press.

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

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

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

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

Dorigo, M., & Birattari, M. (2007). Swarm intelligence. *Scholarpedia*, *2*(9), 1462.

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

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization*. Cambridge: MIT Press.

Fisher, R. A. (1925). *Statistical methods for research workers*. Edinburgh: Oliver and Boyd.

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

Gutjahr, W. J. (2003). A converging ACO algorithm for stochastic combinatorial optimization. In A. Albrecht & K. Steinhofl (Eds.), *LNCS: Vol. 2827. Stochastic algorithms: foundations and applications* (pp. 10–25). Berlin: Springer.

Gutjahr, W. J. (2004). S-ACO: An ant based approach to combinatorial optimization under uncertainty. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, & T. Stützle (Eds.), *LNCS: Vol. 3172. Ant colony optimization and swarm intelligence, 5th international workshop*, ANTS 2004 (pp. 238–249). Berlin: Springer.

Hoos, H., & Stützle, T. (2005). *Stochastic local search: foundations and applications*. San Mateo: Morgan Kaufmann.

Jaillet, P. (1985). *Probabilistic traveling salesman problems*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA.

Johnson, D. S., & McGeoch, L. A. (1997). The travelling salesman problem: a case study in local optimization. In E. H. L. Aarts & J. K. Lenstra (Eds.), *Local search in combinatorial optimization* (pp. 215–310). Wiley: New York.

Johnson, D. S., McGeoch, L.A., Rego, C, & Glover, F. (2001). 8th DIMACS implementation challenge. URL http://www.research.att.com/~dsj/chtsp/.

Laporte, G., Louveaux, F., & Mercure, H. (1994). A priori optimization of the probabilistic traveling salesman problem. *Operations Research*, *42*, 543–549.

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

Stützle, T. (2002). ACOTSP: A software package of various ant colony optimization algorithms applied to the symmetric traveling salesman problem. URL http://www.aco-metaheuristic.org/aco-code/.

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

Tukey, J. W. (1949). Comparing individual means in the analysis of variance. *Biometrics*, *5*(2), 99–114.