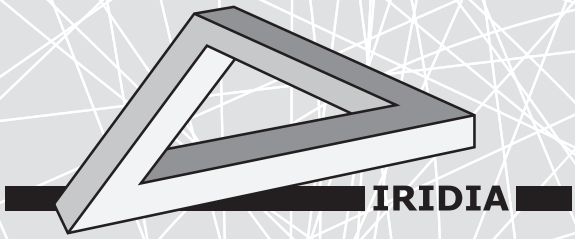


Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Estimation-based Local Search for Stochastic Combinatorial Optimization

Mauro Birattari, Prasanna Balaprakash,
Thomas Stützle, and Marco Dorigo

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2007-003

February 2007

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/2007-003

Estimation-based Local Search for Stochastic Combinatorial Optimization

Mauro Birattari mbiro@ulb.ac.be
Prasanna Balaprakash pbalapra@ulb.ac.be
Thomas Stützle stuetzle@ulb.ac.be
Marco Dorigo mdorigo@ulb.ac.be

IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

February 12, 2007

In recent years, much attention has been devoted to the development of metaheuristics and local search algorithms for tackling stochastic combinatorial optimization problems. This paper focuses on local search algorithms; their effectiveness is greatly determined by the evaluation procedure that is used to select the best of several solutions in the presence of uncertainty. In this paper, we propose an effective evaluation procedure that makes use of *empirical estimation* techniques. We illustrate our approach and assess its performance on the probabilistic traveling salesman problem. Experimental results on a large set of instances show that our approach can lead to a very fast and highly effective local search algorithms.

Keywords: stochastic combinatorial optimization; suboptimal algorithms; iterative improvement; simulation;

---

1 Introduction

In a large number of practically relevant combinatorial optimization problems, the objective function is affected by uncertainty. Examples include portfolio management, vehicle routing, resource allocation, scheduling, and the modeling and simulation of large molecular systems in bio-informatics (Fu, 2002). In order to tackle these problems, it is customary that a setting is considered in which the cost of each solution is a random variable, and the goal is to find a solution that minimizes some statistics of the latter. For a number of practical and theoretical reasons, the optimization is performed with respect to the expectation (Fu, 1994, 2002). In this context, two approaches have been discussed in the literature: *analytical computation* and *empirical estimation*. While the former explicitly relies on the underlying probabilistic model for computing the expectation through a complex analytical development, the latter simply estimates the expectation through Monte Carlo simulation.

Designing efficient algorithms for solving stochastic combinatorial optimization problems is a challenging task. The main difficulty is that the computational complexity associated to the combinatorial explosion of potential solutions is exacerbated by the added element of uncertainty in the data. We refer the reader to Fu (1994) and Bianchi (2006) for surveys on solution techniques for stochastic combinatorial optimization problems. Extensive computational results from the literature have shown that local search is an effective approach for stochastic combinatorial optimization (Pichitlamken and Nelson, 2003; Gutjahr, 2004; Bianchi et al., 2006). However, a main challenge in applying local search lies in designing an effective evaluation procedure that conclusively determines if one solution is better than another.

In this paper, we focus on a very basic local search algorithm known as iterative improvement, which starts from some initial solution and then moves to an improving neighboring solution until a local optimum is found. In this algorithm, the cost of solutions can be evaluated in two ways: (i) full evaluation that computes the cost of each solution from scratch; (ii) partial evaluation that computes only the cost difference between a particular solution and every neighboring solution. The former is applicable to all classes of stochastic combinatorial optimization problems. The latter, widely known as *delta evaluation*, is highly profitable in terms of computation time whenever partial evaluation of solutions is feasible (Bertsimas, 1988).

The *delta evaluation* strategies proposed in the literature for iterative improvement algorithms are based on *analytical computation* (Bertsimas, 1988; Bianchi, 2006). In this strategy, the cost difference between two solutions is given by a closed-form expression that is obtained through problem-specific knowledge and rather heavy mathematical derivations. The main drawbacks of this techniques are that (i) they are not general-purpose; and (ii) they cannot be applied to problems in which the cost difference cannot be expressed in an analytical way (Fu, 1994). Several research results from the simulation literature suggest that the *empirical estimation* approach can overcome the difficulties posed by *analytical computation*. Surprisingly, to the best of our knowledge, the idea of using estimation techniques in *delta evaluation* has never been thoroughly investigated.

The goal of this paper is to present an iterative improvement algorithm that performs *delta evaluation* using *empirical estimation* techniques. We use the PROBABILISTIC TRAVELING SALESMAN PROBLEM as an example to illustrate the proposed approach and to assess its performance.

The paper is organized as follows. In Section 2, we give a formal definition of stochastic combinatorial optimization problems and introduce the PTSP as an example. In Section 3, we review the state-of-the-art iterative improvement algorithms for the PTSP. In Section 4, we introduce the *estimation-based* iterative improvement algorithm for the PTSP and we study its performance in Section 5. In Section 6, we conclude the paper.

2 Stochastic Combinatorial Optimization Problems

In this paper, we consider stochastic combinatorial optimization problems that can be described as:

$$\text{Minimize} \quad F(x) = E\big[f(x,\Omega)\big], \qquad \text{subject to } x \in S, \tag{1}$$

where x is a solution, S is the finite set of feasible solutions, the operator E denotes the mathematical expectation, and f is the cost function, which depends on x and on a multivariate random variable Ω. The presence of the latter makes $f(x,\Omega)$ a random variable. The goal is to find a feasible solution that minimizes the expected cost.

A paradigmatic example of a stochastic combinatorial optimization problem is the PROBABILISTIC TRAVELING SALESMAN PROBLEM (PTSP) (Jaillet, 1985). Formally, an instance of the PTSP is defined on a complete graph $G = (V, A, C, P)$, where $V = \{1, 2, \ldots, n\}$ is a set of nodes, $A = \{\langle i, j\rangle : i, j \in V, i \neq j\}$ is the set of edges that completely connects the nodes, $C = \{c_{ij} : \langle i, j\rangle \in A\}$ is a cost-matrix that gives the travel cost associated with each edge $\langle i, j\rangle \in A$, and $P = \{p_i : i \in V\}$ is a set of probabilities that for each node i specifies its probability p_i of requiring a visit. Hence, for the PTSP the random variable Ω is described by an n-variate Bernoulli distribution and a realization of Ω is a binary vector of size n where a `1` in position i indicates that node i requires visit and a `0` indicates that it does not. We assume that the cost matrix C is symmetric.

Usually, the PTSP is tackled by *a priori* optimization (Jaillet, 1985; Bertsimas et al., 1990), which consists of two stages: In the first stage, a solution, which can be represented as a permutation of the nodes, is determined before the actual realization of the random variable Ω is available. This is the so-called *a priori* solution. In the second stage, after the realization of the random variable is known, an *a posteriori* solution is obtained from the *a priori* solution by visiting the nodes prescribed by the given realization in the order in which they appear in the *a priori* solution. The nodes that do not require visit are simply skipped. Figure 1 shows an example.

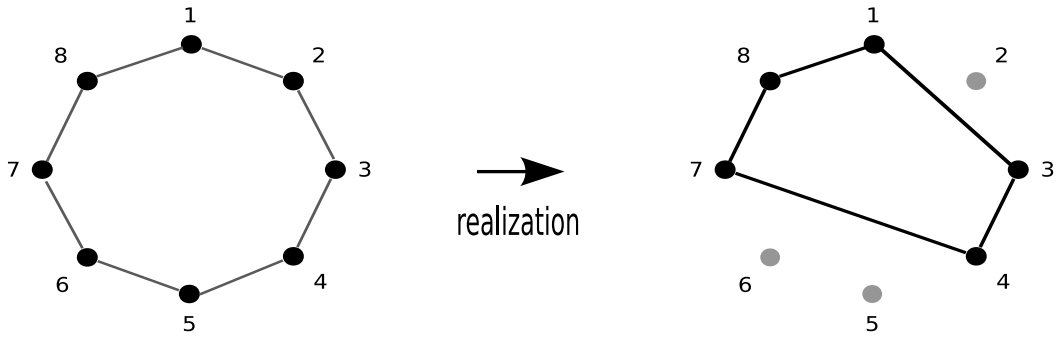


Figure 1: An *a priori* solution for a PTSP instance with 8 nodes. The nodes in the *a priori* solution are visited in following order: 1, 2, 3, 4, 5, 6, 7, 8, and 1. Assume that a realization of Ω prescribes that nodes 1, 3, 4, 7, and 8 are to be visited. The resulting *a posteriori* solution is obtained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

The goal in the PTSP is to find an *a priori* solution that minimizes the expected cost of the *a posteriori* solution, where the expectation is computed with respect to a given n-variate Bernoulli distribution. Note that when $P = \{p_i = p : i \in V\}$, the PTSP instance is called homogeneous, otherwise, if for at least two nodes i and j we have $p_i \neq p_j$, we are faced with a heterogeneous PTSP.

3 Local Search for the PTSP

Local search is a method for searching a given space of solutions. It consists in moving from one solution to another neighboring one according to an acceptance criterion. Many local search methods exist and the one that has received the most attention in the PTSP literature is iterative improvement. Iterative improvement algorithms start from some initial solution and repeatedly try to move from a current solution x to a lower cost neighboring solution x'. A solution that does not have any improving neighboring solution is a local minimum and the iterative improvement search terminates with such a solution. In the PTSP literature, mainly the following two neighborhood structures were considered:

- ***2-exchange neighborhood***: Two solutions are neighbors if, and only if, they differ in exactly two edges. In other words, the neighborhood of a solution is the set of solutions obtained by deleting any two edges $\langle a, b\rangle$ and $\langle c, d\rangle$ and by replacing them with $\langle a, c\rangle$ and $\langle b, d\rangle$. See Figure 2(a) for an example.
- ***Node-insertion neighborhood***: Two solutions are neighbors if, and only if, they differ in the position of exactly one node. In other words, the neighborhood of a solution is the set of solutions obtained by deleting a node a and inserting it elsewhere in the solution. See Figure 2(b) for an example.

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

Iterative improvement algorithms for the PTSP are similar to the usual iterative improvement algorithms for the TSP: the cost difference between two TSP neighboring solutions x and x' is computed by considering the cost contribution of solution components that are not common to x

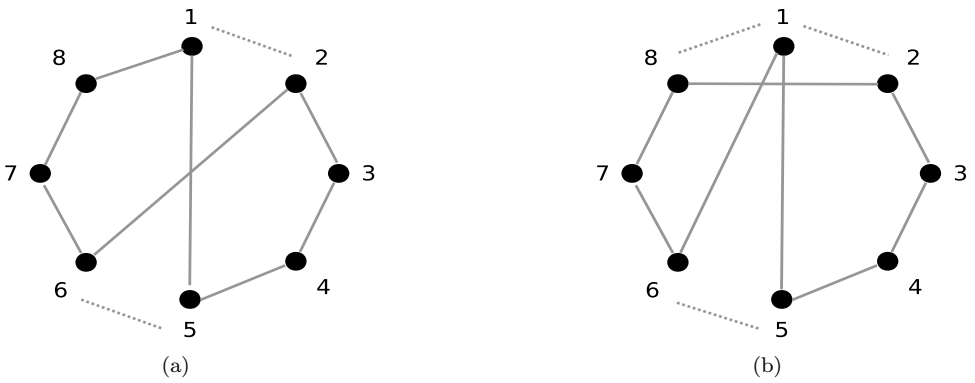


Figure 2: Plot 2(a) shows a *2-exchange* move that is obtained by deleting two edges $\langle 1, 2\rangle$ and $\langle 5, 6\rangle$ of the solution and by replacing them with $\langle 1, 5\rangle$ and $\langle 2, 6\rangle$. Plot 2(b) shows a *node-insertion* move obtained by deleting node 1 from its current position in the solution and inserting it between nodes 5 and 6.

and x'. In the case of a *2-exchange* move, the cost difference between the neighboring solutions is simply given by $c_{a,c} + c_{b,d} - c_{a,b} - c_{c,d}$. This technique is widely known as *delta evaluation*. The only difference between PTSP and TSP versions is that, in the former, the random variable Ω has to be taken into account in the *delta evaluation*. In the rest of this section, we describe how *delta evaluation* is performed in state-of-the-art iterative improvement algorithms for the PTSP.

3.1 State-of-the-art iterative improvement algorithms for the PTSP

For the homogenous PTSP, Bertismas (Bertsimas, 1988; Bertsimas and Howell, 1993) derived closed-form *delta evaluation* expressions based on *analytical computation* for the *2-exchange neighborhood* and the *node-insertion neighborhood*. Equipped with these expressions, the author also proposed two iterative improvement algorithms, namely, `2-p-opt` and `1-shift`. For both algorithms, the total time complexity of the neighborhood exploration and evaluation is $O(n^2)$. For the heterogeneous case, Chervi (1988) proposed closed-form *delta evaluation* expressions for `2-p-opt` and `1-shift`, where each algorithm explores and evaluates the neighborhood in $O(n^3)$. Bianchi et al. (2005) and Bianchi and Campbell (2007) proved that the expressions derived by Bertsimas (1988); Chervi (1988); Bertsimas and Howell (1993) are incorrect and corrected the errors. Furthermore, for the heterogeneous PTSP, the authors showed that the neighborhoods in `2-p-opt` and `1-shift` can be explored and evaluated in $O(n^2)$ rather than $O(n^3)$.

In her Ph.D. thesis (Bianchi, 2006), Bianchi considered also the possibility of using an *estimation-based* approach for the *delta evaluation* in `2-p-opt` and `1-shift`. However, based on an asymptotic analysis, the author speculated that this approach might be much more computationally expensive than *analytical computation* techniques and for this reason, the idea of using an *estimation-based* approach has been abandoned without any empirical investigation. Note that the experimental results of the *estimation-based* approach presented in this paper contradict Bianchi's conjecture.

3.2 `2-p-opt` and `1-shift`

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

Table 1: For a given probability p, the maximum size ($n_{critical}$) of the PTSP instance that can be handled by `2-p-opt` without numerical problems on a 32-bit GNU system.

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

an improvement is found, the search is restarted from the first phase: the *swap-neighborhood* exploration. `1-shift` differs from `2-p-opt` only in the second phase: it uses the *node-insertion neighborhood* with the *best-improvement* rule.

We refer the reader to Bianchi et al. (2005); Bianchi and Campbell (2007) for the *delta evaluation* expressions that are used in `2-p-opt` and `1-shift`. Even though the time complexity is $O(n^2)$ for both `2-p-opt` and `1-shift`, the asymptotic notation captures only the growth rates with respect to the number of neighboring solutions and does not reflect the large multiplicative constant. Indeed, a closer look at the *delta evaluation* expressions presented in (Bianchi et al., 2005; Bianchi and Campbell, 2007) reveals that there is a large constant of proportionality hidden in the asymptotic notation.

The final picture we reach is that the state-of-the-art iterative improvement algorithms for the PTSP use neighborhood-specific *delta evaluation* expressions, which are based on *analytical computation* techniques. The advantage of this framework is that the values of the computed cost differences are exact. However, from a practical perspective, this framework has some limitations.

3.3 Limitations of the state-of-the-art iterative improvement algorithms for the PTSP

Theoretically, the *delta evaluation* expressions proposed in (Bianchi et al., 2005; Bianchi, 2006) for the PTSP could be applied to solve instances of any size. However, in practice, these expressions suffer from numerical precision problems when applied to large instances. In fact, to use the *delta evaluation* expressions in `2-p-opt`, the term $(1-p)^{(k-n)}$ has to be computed, where p is the probability, k is the number of nodes between two considered edges, and n is the size of the instance. For some values of p, k, and n, this term can result in an overflow. As an illustration, consider a typical 32-bit GNU system, where, according to the IEEE 754 standard (1985), double precision floating point variables can take a maximum value of about `1e+308` (Griffith, 2002). Given a homogeneous PTSP instance of probability p with n nodes, the condition for the numerical overflow is $(1-p)^{(k-n)} >$ `1e+308`. From this condition, one can obtain, after basic transformations, a critical value for n, above which the computation of the *delta evaluation* expression is not possible:

$$n_{critical} = 1 - \frac{308}{\log_{10}(1-p)}. \tag{2}$$

Table 1 shows the probability levels and the corresponding maximum size of instances that can be tackled by `2-p-opt` without any numerical overflow in a 32-bit GNU system. Note that the very same numerical problem occurs in `1-shift`. Also the *analytical computation* algorithms for the heterogeneous PTSP are affected by this problem.

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

4 Estimation-based iterative improvement algorithms for the PTSP

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

$$\hat{F}_M(x) = \frac{1}{M}\sum_{r=1}^{M} f(x,\omega_r). \tag{3}$$

As it can be shown easily, $\hat{F}_M(x)$ is an *unbiased* estimator of $F(x)$. In iterative improvement algorithms for the PTSP, we need to compare two neighboring solutions x and x' to select the one of lower cost. This can be achieved by determining the sign of the cost difference $F(x')-F(x)$. For x', an *unbiased* estimator $\hat{F}_{M'}(x')$ of $F(x')$ can be obtained from a sample $f(x',\omega'_1), f(x',\omega'_2), \ldots, f(x',\omega'_{M'})$ of costs of *a posteriori* solutions through M' independent realizations of Ω. Eventually, $\hat{F}_{M'}(x') - \hat{F}_M(x)$ is an *unbiased* estimator of $F(x') - F(x)$.

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

$$\begin{aligned}\hat{F}_M(x') - \hat{F}_M(x) &= \frac{1}{M}\sum_{r=1}^{M} f(x',\omega_r) - \frac{1}{M}\sum_{r=1}^{M} f(x,\omega_r) \\ &= \frac{1}{M}\sum_{r=1}^{M} \Big(f(x',\omega_r) - f(x,\omega_r)\Big).\end{aligned} \tag{4}$$

In this paper, we use the same set of M realizations for all steps of the iterative improvement algorithms. Other approaches could be adopted: for example, M realizations could be sampled anew for each step of the algorithm or even for each comparison. A discussion about this issue is given in Section 5.4.

Using Equation 4, given two neighboring solutions x and x' and a realization ω, a naïve approach to compute the cost difference between two *a posteriori* solutions consists in computing first the complete cost of each *a posteriori* solution and then the difference between them. However, a more efficient algorithm can be obtained by adopting the idea of *delta evaluation*: Given the *a priori* solutions and a realization ω, such an algorithm requires identifying the edges that are not common to the two *a posteriori* solutions.

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

In general, for every edge $\langle i,j\rangle$ that is deleted from x, one needs to find the corresponding edge $\langle i^*,j^*\rangle$ in the *a posteriori* solution of x. We call this edge the *a posteriori edge* and it is obtained as follows: If node i requires visit, then $i^* = i$, otherwise, i^* is the first predecessor of i in x such that $\omega[i^*] = 1$. If node j requires visit, then $j^* = j$, otherwise, j^* is the first successor of j such that $\omega[j^*] = 1$. Recall that in a *2-exchange* move, the edges $\langle a,b\rangle$ and $\langle c,d\rangle$ are deleted from x and replaced by $\langle a,c\rangle$ and $\langle b,d\rangle$. For a given realization ω and the corresponding *a-posteriori-edges*, $\langle a^*,b^*\rangle$, $\langle c^*,d^*\rangle$, the cost difference between the two *a posteriori* solutions is given by $c_{a^*,c^*} + c_{b^*,d^*} - c_{a^*,b^*} - c_{c^*,d^*}$. Figure 4 shows the *a-posteriori-edges* for the example given in Figure 3.

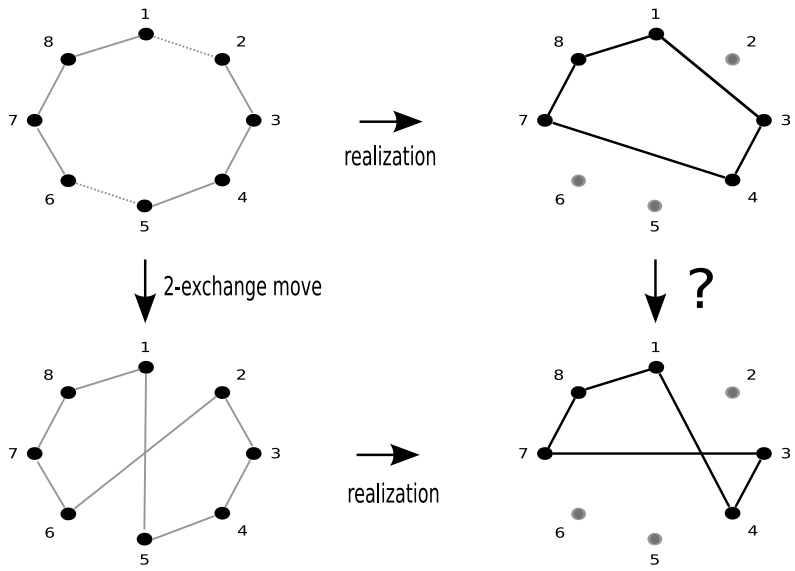


Figure 3: In this example, a *2-exchange* move is obtained by deleting the edges $\langle 1,2\rangle$ and $\langle 5,6\rangle$ from the *a priori* solution and by replacing them with $\langle 1,5\rangle$ and $\langle 2,6\rangle$. Assume that a realization of Ω prescribes that nodes 1, 3, 4, 7, and 8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

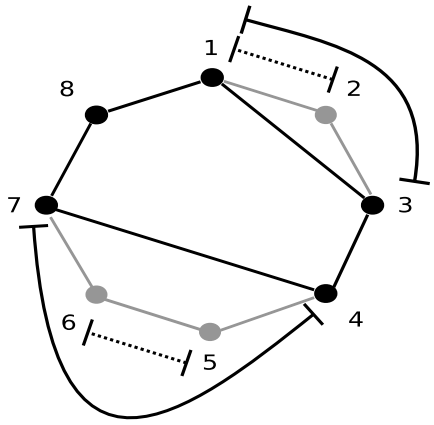


Figure 4: The steps performed for finding the *a-posteriori-edges*. Assume that, without loss of generality, the nodes are visited in the order: 1, 2, 3, 4, 5, 6, 7, 8, and 1. The edges $\langle 1,2\rangle$ and $\langle 5,6\rangle$ are deleted and the gray colored nodes do not require visit. The first *successor* of node 2 that requires visit is 3; the first *predecessor* of node 5 that requires visit is 4; the first *successor* of node 6 that requires visit is 7. The *a-posteriori-edges* are therefore given as $\langle 1,3\rangle$ and $\langle 4,7\rangle$.

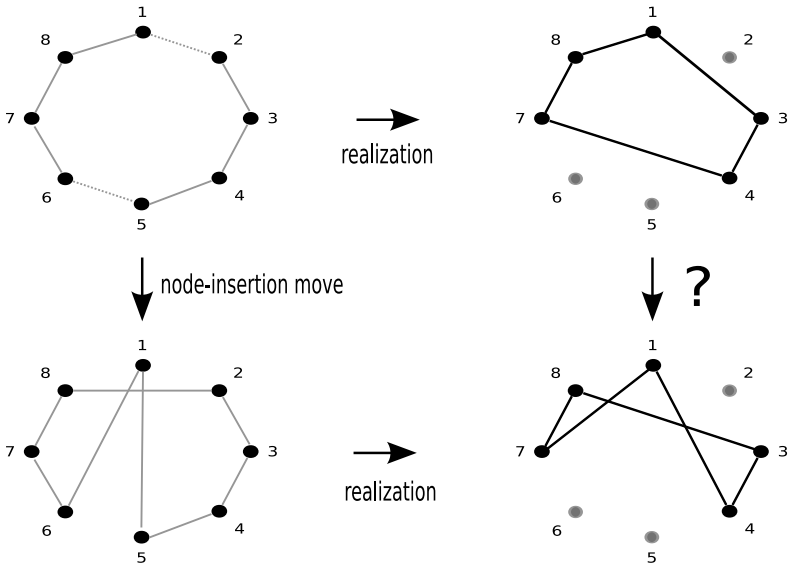


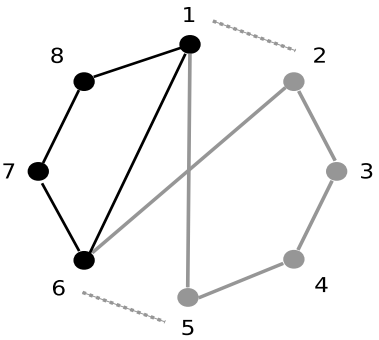
Figure 5: In this example, the *node-insertion* move is obtained by deleting node 1 and inserting it between nodes 5 and 6. Consequently, the edges $\langle 8,1\rangle$, $\langle 1,2\rangle$, and $\langle 5,6\rangle$ are deleted from the *a priori* solution and replaced by $\langle 8,2\rangle$, $\langle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

The procedure described can be directly extended to the *node-insertion* move. See Figure 5 for an example.

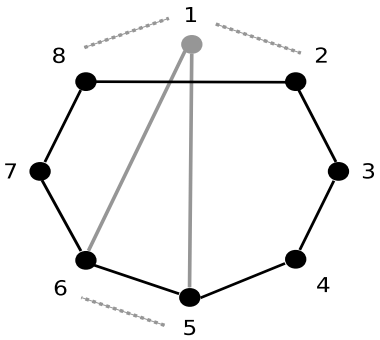
It is worth discussing here some degenerate cases: in a *2-exchange* move that deletes the edges $\langle a,b\rangle$ and $\langle c,d\rangle$, and where no node between the nodes b and c or between nodes a and d requires visit, the difference between the two *a posteriori* solutions is zero—see Figure 6(a); in a *node-insertion* move, if the insertion node does not require vist, then the cost difference between the two *a posteriori* solutions is zero—see Figure 6(b). In this second case, one can avoid unnecessary computations by checking whether the insertion node requires visit before finding the *a-posteriori-edges*.

The proposed approach has a number of advantages: First, the *estimation-based delta evaluation* procedure is general and can be applied to any neighborhood structure without requiring neighborhood-specific, complex and error-prone mathematical derivations. In virtue of this versatility, rather than using the *node-insertion neighborhood* or the *2-exchange neighborhood* structure, we use a hybrid neighborhood structure that includes the *node-insertion neighborhood* on top of the *2-exchange neighborhood* structure. In the TSP literature (Bentley, 1992), this hybrid neighborhood is widely known as the *2.5-exchange neighborhood*: when checking for a *2-exchange* move on any two edges $\langle a,b\rangle$ and $\langle c,d\rangle$, it is also checked whether deleting any one of the nodes of an edge, say for example a, and inserting it between nodes c and d results in an improved solution (Bentley, 1992). Second, unlike `2-p-opt` and `1-shift`, the proposed approach does not impose any constraints on the order in which the neighborhood should be explored. This allows for an easy integration of the classical TSP neighborhood reduction techniques such as *fixed-radius search*, *candidate lists* and *don't look bits* (Martin et al., 1991; Bentley, 1992; Johnson and McGeoch, 1997). We denote the proposed algorithm `2.5-opt-EEs` where `EE` and `s` stand for *empirical estimation* and *speedup*, respectively. Note that `2.5-opt-EEs` uses the *first-improvement* rule.

In order to implement `2.5-opt-EEs` effectively, a specific data structure is needed. We use a data structure in which data items can be accessed both as elements of a doubly circularly linked list and as elements of a one dimensional array, both of size n. Each data item has an integer variable to store a node of the *a priori* solution. This structure is efficient for finding *a*



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



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

Figure 6: Some degenerate cases that can occur in the evaluation of cost differences.

posteriori-edges: predecessor and successor of a node are simply obtained by following the links pointing towards the previous data item and next data item, respectively. To access data items as the elements of the array, a data item representing node i is stored at position i of the array and this arrangement is kept unchanged throughout the search process. Consequently, given a node i, its data item can be accessed in $O(1)$ time. Moreover, each data item stores an array of size M—the realization array—which is indexed from 1 to M. Element r of the realization array, $1 \leqslant r \leqslant M$, is either `1` or `0` indicating whether the node requires visit or not in realization ω_r. Figure 7 shows the data structure that is used in `2.5-opt-EEs`. Whenever, an improved solution is found, only the links of the particular data items whose nodes are involved in the exchange move are modified. Furthermore, each data item comprises also two auxiliary fields for the neighborhood reduction techniques: one integer variable for the *don't look bit* and one integer array of size L for the *candidate list* of each node.

Concerning the computational complexity of the *estimation-based* iterative improvement algorithm, Bianchi (2006) reached the conclusion that the time complexity of a complete neighborhood scan is $O(Mpn^3)$. Indeed, the number of solutions in the *2.5-exchange neighborhood* is $O(n^2)$; the maximum number of steps for finding the *a-posteriori-edges* for a given realization is n; and the number of realizations considered is M. Bianchi included in the complexity also the probability p that a node requires visit, but the inclusion of this term is not completely justified and is not thoroughly discussed in her work (Bianchi, 2006). The main result of the analysis of Bianchi is that the time complexity grows with the cube of n. On the basis of this result, Bianchi decided that the *estimation-based* approach does not deserve any further attention. However, the above analysis does not hold for `2.5-opt-EEs`: The use of a *candidate list* of size L reduces the neighborhood size from $O(n^2)$ to $O(nL)$, which in turn reduces the worst-case time complexity of a neighborhood scan to $O(n^2LM)$. Furthermore, it should be observed that, since we explicitly deal with a probabilistic model, a more informative average-case analysis can be derived easily: The expected number of steps for finding an *a posteriori edge* is $(1-p)/p$. As a result, the average-case time complexity of a complete neighborhood scan is $O(nLMp^{-1})$.

5 Experimental analysis

In this section, we present the experimental settings considered and our empirical results. We base our analysis on *homogeneous* PTSP instances that we obtained from TSP instances generated with the DIMACS instance generator (Johnson et al., 2001). We carried out experiments with two classes of instances. In the first class, nodes are *uniformly distributed* in a $10^6 \times 10^6$ square;

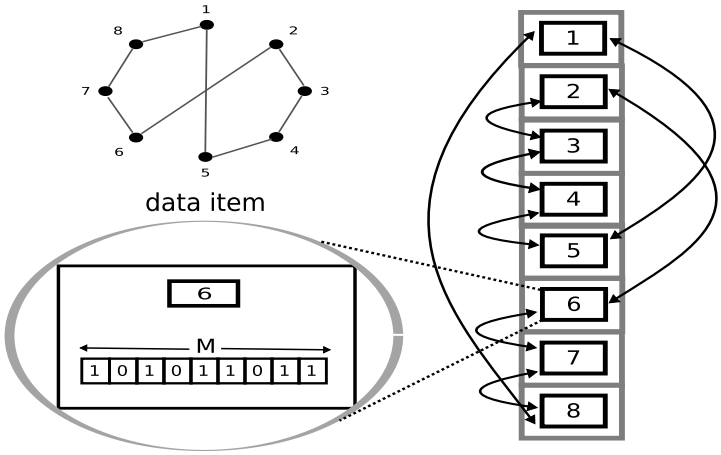


Figure 7: Assume that we have an *a priori* solution in which the nodes are visited in the order: 1, 5, 4, 3, 2, 6, 7, 8, 1. This is encoded in the doubly circularly linked list data structure as shown in the plot. Also note that the data items can be accessed as the elements of the one dimensional array.

in the second, nodes are arranged in a number of *clusters*, in a square of the same size. Due to the limitations of the state-of-the-art PTSP algorithms that were discussed in Section 3.3, we compare `2.5-opt-EEs` with `2-p-opt` and `1-shift` using instances up to 300 nodes. For each instance class, we generated 100 TSP instances of 100, 200 and 300 nodes. From each TSP instance, we obtained 9 PTSP instances by letting the probability range in $[0.1, 0.9]$ with a step size of 0.1. Due to space limitations, we report only the results obtained on the *clustered* instances with 300 nodes. The general trends of the experimental results obtained on the other instances are similar; we refer the reader to Birattari et al. (2007) for the complete set of results.

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

The nearest-neighbor heuristic is used to generate initial solutions. The *candidate list* is set to size 40 and is constructed with the *quadrant nearest-neighbor* strategy (Penky and Miller, 1994; Johnson and McGeoch, 1997). Each iterative improvement algorithm is run until it reaches a local optimum. The number of realizations in `2.5-opt-EEs` is set to 100. In order to highlight this fact, we denote the algorithm `2.5-opt-EEs-100`.

For the *homogenous* PTSP with probability p and size n, given an *a priori* solution x, the exact cost $F(x)$ of x can be computed using the formula

$$F(x) = \sum_{u=1}^{n} \sum_{v=1}^{n-1} p^2 (1-p)^{v-1} c_{(x(u),x(v))}, \tag{5}$$

where $x(u)$ and $x(v)$ are the nodes of index u and v in x, respectively (Jaillet, 1985). We use this formula for post-evaluation purposes: for each algorithm, whenever an improved solution is found, we record the solution. In order to compare the cost of the *a priori* solutions reached by the algorithms, we use Equation 5 to compute the cost of the recorded solutions obtained from each algorithm.

In addition to tables, we visualize the results using *runtime development plots*. These plots show how the cost of solutions develops over computation time and they can be used to compare the performance of several algorithms over time. In these plots, the x-axis indicates computation time and the y-axis indicates the cost of the solutions found, averaged over 100 instances. For comparing several algorithms, one of them has been taken as a reference: for each instance, the computation time and the cost of the solutions of the algorithms are normalized by the average

computation time and average cost of the local optima obtained by the reference algorithm. For convenience, the x-axis is in logarithmic scale. We report one such plot for each probability level under consideration.

5.1 Experiments on neighborhood reduction techniques

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

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

The results given in Figure 8 show that `2.5-opt-ACs` *dominates* `2-p-opt` and `1-shift` with the only exception being for the values of p ranging between 0.5 and 0.9: at the *early stages* of the search and for a very short time range, the average cost of the solutions obtained by `1-shift` is slightly lower than that of `2.5-opt-ACs`. Concerning the time required to reach local optima, irrespective of the probability value, `2.5-opt-ACs` is faster than `2-p-opt` by approximately a factor of four. In the case of `1-shift`, the same tendency holds when $p \geq 0.5$. However, for small values of p, the difference in speed between `2.5-opt-ACs` and `1-shift` is small. Concerning the average cost of local optima found, `2.5-opt-ACs` is between 2% and 5% better than `2-p-opt`. We can observe the same trend also in `1-shift`; an exception is for $p \leq 0.3$, where the difference between the average cost of the local optima obtained by `2.5-opt-ACs` and `1-shift` is very small. For details, see Table 2.

In order to test that the observed difference between the cost of local optima are significant in a statistical sense, we use a Wilcoxon test. Table 3 shows the p-values. The cost of the local optima obtained by `2.5-opt-ACs` is significantly lower than that of `1-shift` and `2-p-opt` for all probability values, the only exception being $p \leq 0.2$, where the difference between the cost of the local optima obtained by `2.5-opt-ACs` and `1-shift` is not significant.

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

5.2 Experiments to assess the estimation approach

In this section, we compare `2.5-opt-EEs-100` with `2.5-opt-ACs`. The two algorithms differ only in the *delta evaluation* procedure they adopt: *empirical estimation* versus *analytical computation*. The experimental results are illustrated using a run time development plot and are shown in Figure 9. Moreover, in order to show how significant is the difference between the cost of the solutions obtained by `2.5-opt-EEs-100` and `2.5-opt-ACs` over time, in Figure 10 we provide a plot of some quantiles of the distribution of the difference between the normalized values of the

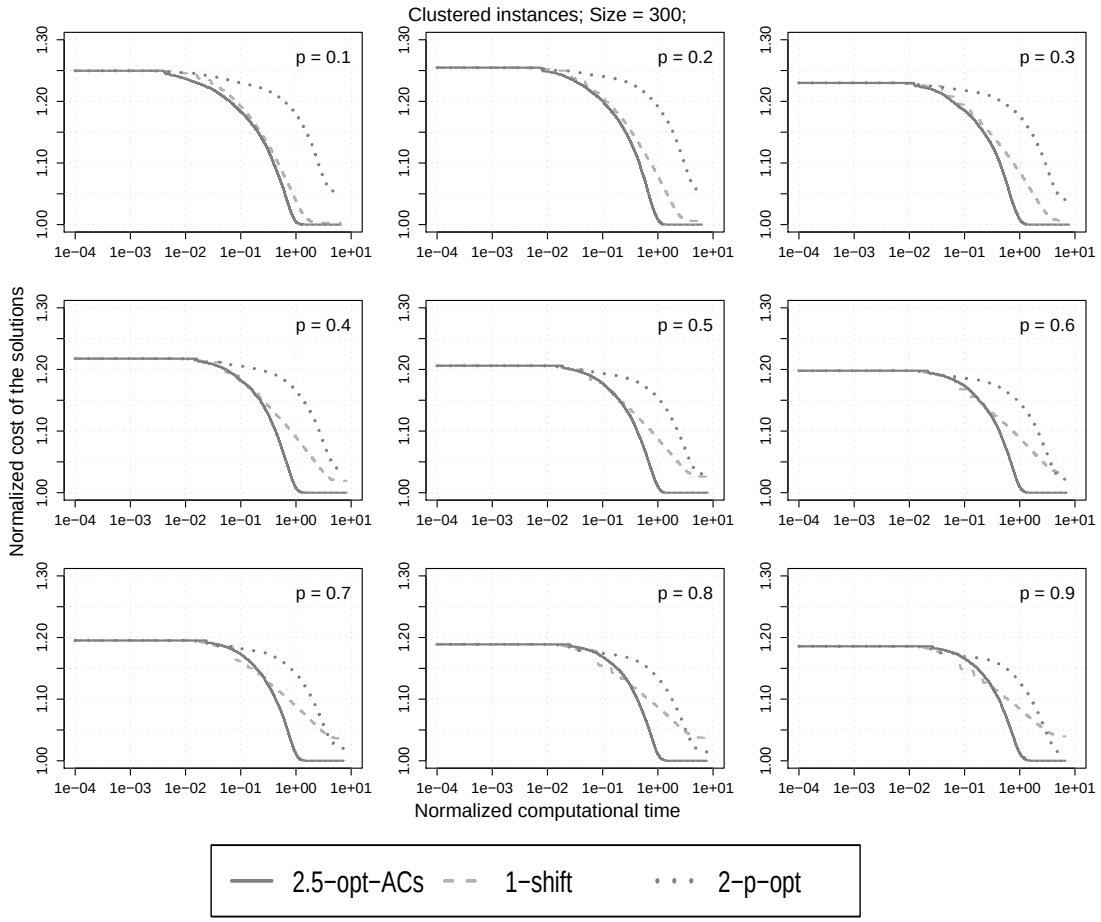


Figure 8: Experimental results on clustered homogeneous PTSP instances of size 300. The plots represent the average cost of the solutions obtained by `2-p-opt` and `1-shift` normalized by the one obtained by `2.5-opt-ACs`. Each algorithm is stopped when it reaches a local optimum.

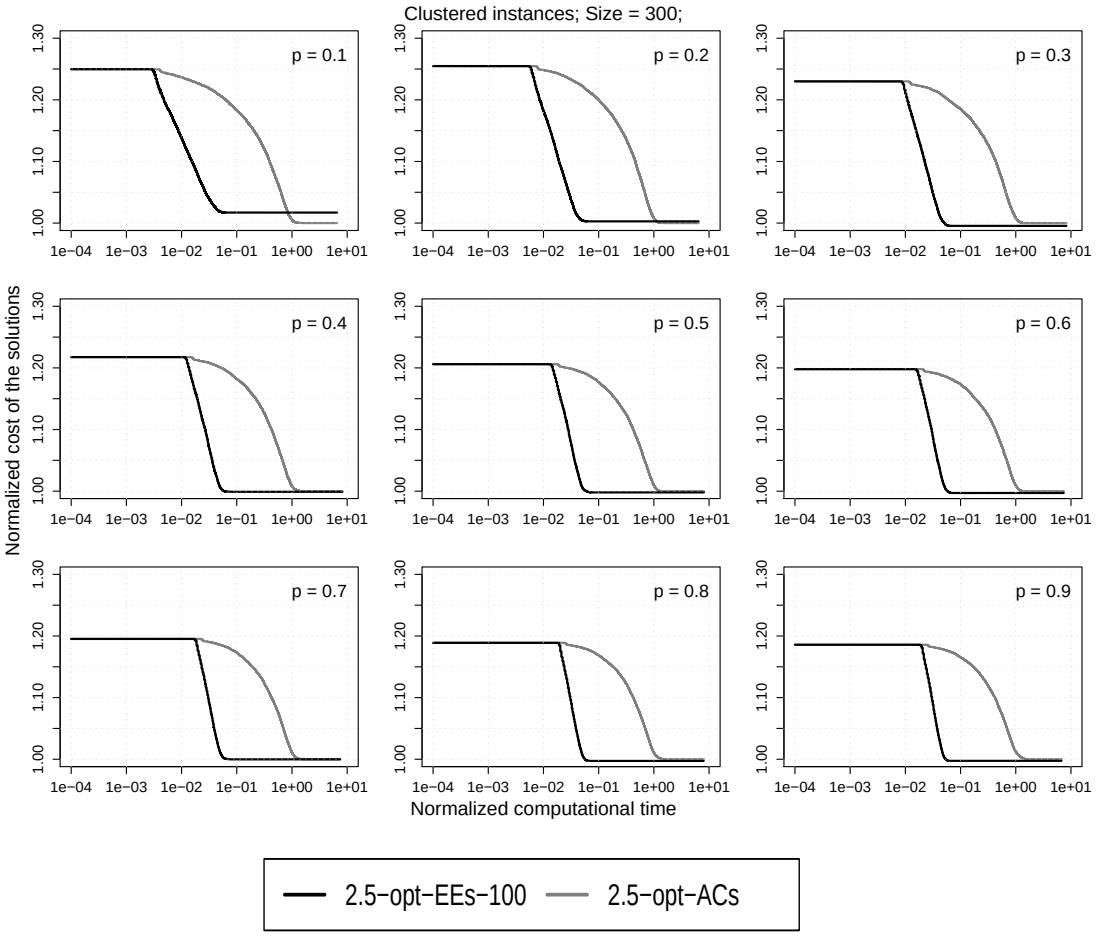


Figure 9: Experimental results on clustered homogeneous PTSP instances of size 300. The plots represent the average cost of the solutions obtained by `2.5-opt-EEs-100` normalized by the one obtained by `2.5-opt-ACs`. Each algorithm is stopped when it reaches a local optimum.

solution costs of `2.5-opt-EEs-100` and `2.5-opt-ACs`. In this plot, a value greater than zero indicates that `2.5-opt-EEs-100` is better than `2.5-opt-ACs`, and vice versa.

Concerning the average cost of local optima, the two algorithms are similar with the only exception of $p = 0.1$, where the average cost of the local optima obtained by `2.5-opt-EEs-100` is approximately 2% higher than that of `2.5-opt-ACs`. Concerning the time required to reach local optima, irrespective of the probability value, `2.5-opt-EEs-100` is approximately 1.5 orders of magnitude faster than `2.5-opt-ACs`. See Tables 2 for the absolute values and Table 3 for the p-values of the Wilcoxon test.

In Table 4, we report the observed relative difference between the cost of the local optima obtained by the two algorithms and a 95% confidence bound on this relative difference. This bound is obtained through a one-sided paired Wilcoxon test. For the sake of completeness, we also present these data for what concerns the comparison of `2.5-opt-EEs-100` with `1-shift` and `2-p-opt`.

Table 4 confirms that, concerning the average cost of the local optima found, `2.5-opt-EEs-100` is essentially equivalent to `2.5-opt-ACs`. To be more precise, for $p = 0.1$, `2.5-opt-ACs` obtains solutions, the average cost of which is lower than the one of those obtained by `2.5-opt-EEs-100`.

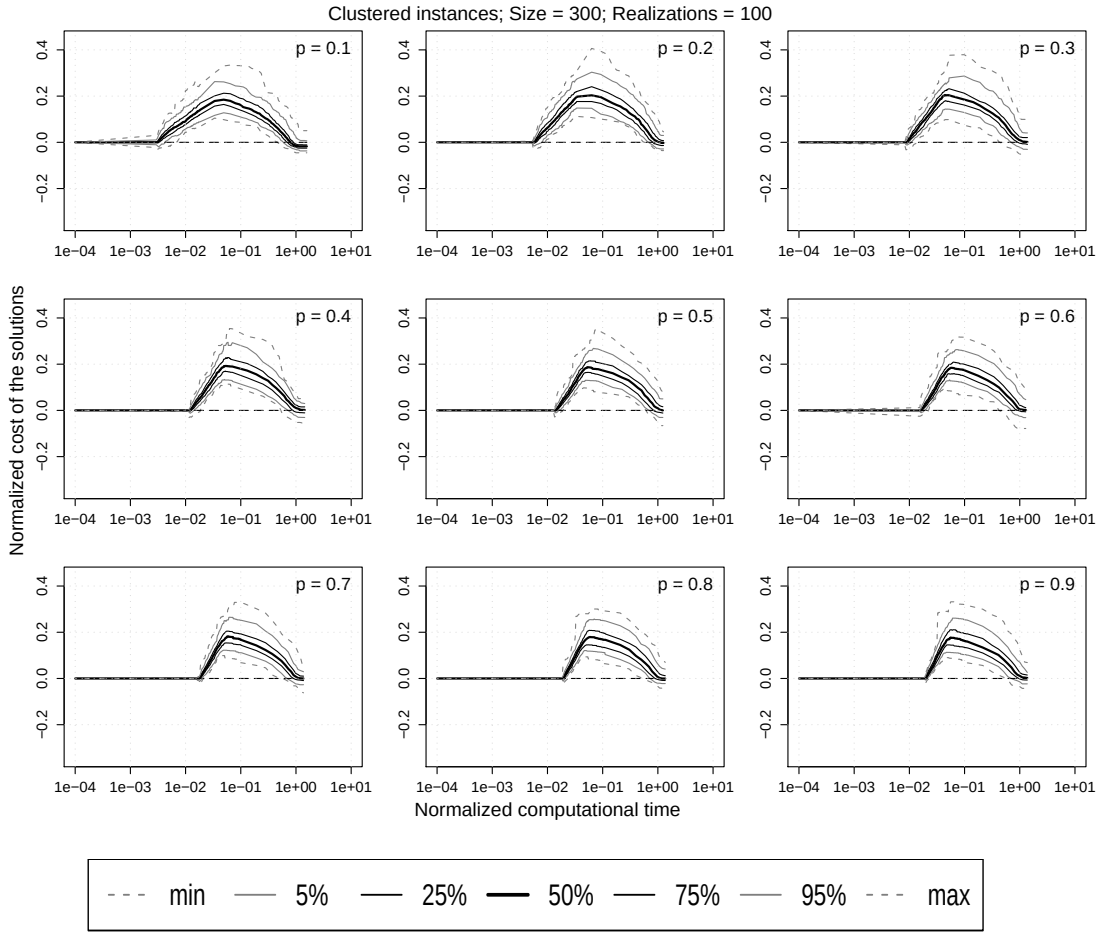


Figure 10: Experimental results on clustered homogeneous PTSP instances of size 300. For the normalized values shown in Figure 9, the difference between `2.5-opt-EEs-100` and `2.5-opt-ACs` over time is computed and the 5%, 25%, 50%, 75%, and 95% quantiles of the distribution of the differences are plotted. For a curve, a value greater than zero indicates that `2.5-opt-EEs-100` is better than `2.5-opt-ACs` and vice versa.

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

	Algorithm	Solution Cost		Computation Time	
		mean	s.d.	mean	s.d.
$p = 0.1$	`2.5-opt-EEs-100`	2776865	456487	0.120	0.014
	`2.5-opt-ACs`	2730221	454321	6.453	1.067
	`1-shift`	2738026	450970	11.771	2.404
	`2-p-opt`	2870013	462383	22.440	5.831
$p = 0.2$	`2.5-opt-EEs-100`	3595283	467721	0.086	0.011
	`2.5-opt-ACs`	3585254	471967	3.413	0.540
	`1-shift`	3606878	467069	10.103	1.773
	`2-p-opt`	3775106	474269	13.848	3.254
$p = 0.3$	`2.5-opt-EEs-100`	4239788	499001	0.064	0.008
	`2.5-opt-ACs`	4259032	501810	2.214	0.399
	`1-shift`	4286461	481061	8.478	1.856
	`2-p-opt`	4429328	497857	9.842	2.462
$p = 0.4$	`2.5-opt-EEs-100`	4764400	517350	0.052	0.006
	`2.5-opt-ACs`	4769245	519437	1.672	0.303
	`1-shift`	4861195	518759	7.154	1.452
	`2-p-opt`	4936775	511292	7.389	1.867
$p = 0.5$	`2.5-opt-EEs-100`	5190835	537186	0.046	0.005
	`2.5-opt-ACs`	5201221	557895	1.421	0.230
	`1-shift`	5336979	544328	5.933	1.355
	`2-p-opt`	5352456	553028	5.978	1.616
$p = 0.6$	`2.5-opt-EEs-100`	5552434	564439	0.042	0.004
	`2.5-opt-ACs`	5568872	586369	1.231	0.204
	`1-shift`	5744938	577165	5.358	1.215
	`2-p-opt`	5689353	567193	5.315	1.204
$p = 0.7$	`2.5-opt-EEs-100`	5874044	579475	0.038	0.004
	`2.5-opt-ACs`	5875100	579015	1.127	0.209
	`1-shift`	6087249	597356	4.851	1.056
	`2-p-opt`	5993481	589339	4.776	1.146
$p = 0.8$	`2.5-opt-EEs-100`	6154030	586937	0.037	0.004
	`2.5-opt-ACs`	6170881	593782	1.042	0.190
	`1-shift`	6401881	611155	4.399	0.972
	`2-p-opt`	6259008	598772	4.366	1.094
$p = 0.9$	`2.5-opt-EEs-100`	6412335	602907	0.036	0.004
	`2.5-opt-ACs`	6428845	602878	1.020	0.220
	`1-shift`	6683735	628833	4.112	0.937
	`2-p-opt`	6491451	604388	4.093	0.954

Table 3: The p-values of the comparison of `2.5-opt-EEs-100`, `2.5-opt-ACs`, `2-p-opt` and `1-shift` on clustered instances of size 300. Each algorithm is allowed to run until it reaches a local optimum. The statistical test adopted is the paired Wilcoxon test with p-values adjusted by Holm's method. The confidence level is 95%. Values in bold mean that the algorithm in the row performs significantly better than the algorithm in the column, while values in italic mean that the algorithm in the column performs significantly better than the algorithm in the row.

		p-values			
		`2.5-opt-EEs-100`	`2.5-opt-ACs`	`1-shift`	`2-p-opt`
$p = 0.1$	`2.5-opt-EEs-100`	-	*0.000*	*0.000*	**0.000**
	`2.5-opt-ACs`	**0.000**	-	0.402	**0.000**
	`1-shift`	**0.000**	0.402	-	**0.000**
	`2-p-opt`	*0.000*	*0.000*	*0.000*	-
		`2.5-opt-EEs-100`	`2.5-opt-ACs`	`1-shift`	`2-p-opt`
$p = 0.2$	`2.5-opt-EEs-100`	-	0.153	0.542	**0.000**
	`2.5-opt-ACs`	0.153	-	0.153	**0.000**
	`1-shift`	0.542	0.153	-	**0.000**
	`2-p-opt`	*0.000*	*0.000*	*0.000*	-
		`2.5-opt-EEs-100`	`2.5-opt-ACs`	`1-shift`	`2-p-opt`
$p = 0.3$	`2.5-opt-EEs-100`	-	0.094	**0.000**	**0.000**
	`2.5-opt-ACs`	0.094	-	**0.017**	**0.000**
	`1-shift`	*0.000*	*0.017*	-	**0.000**
	`2-p-opt`	*0.000*	*0.000*	*0.000*	-
		`2.5-opt-EEs-100`	`2.5-opt-ACs`	`1-shift`	`2-p-opt`
$p = 0.4$	`2.5-opt-EEs-100`	-	0.432	**0.000**	**0.000**
	`2.5-opt-ACs`	0.432	-	**0.000**	**0.000**
	`1-shift`	*0.000*	*0.000*	-	**0.000**
	`2-p-opt`	*0.000*	*0.000*	*0.000*	-
		`2.5-opt-EEs-100`	`2.5-opt-ACs`	`1-shift`	`2-p-opt`
$p = 0.5$	`2.5-opt-EEs-100`	-	0.679	**0.000**	**0.000**
	`2.5-opt-ACs`	0.679	-	**0.000**	**0.000**
	`1-shift`	*0.000*	*0.000*	-	0.217
	`2-p-opt`	*0.000*	*0.000*	0.217	-
		`2.5-opt-EEs-100`	`2.5-opt-ACs`	`1-shift`	`2-p-opt`
$p = 0.6$	`2.5-opt-EEs-100`	-	0.153	**0.000**	**0.000**
	`2.5-opt-ACs`	0.153	-	**0.000**	**0.000**
	`1-shift`	*0.000*	*0.000*	-	*0.000*
	`2-p-opt`	*0.000*	*0.000*	**0.000**	-
		`2.5-opt-EEs-100`	`2.5-opt-ACs`	`1-shift`	`2-p-opt`
$p = 0.7$	`2.5-opt-EEs-100`	-	0.892	**0.000**	**0.000**
	`2.5-opt-ACs`	0.892	-	**0.000**	**0.000**
	`1-shift`	*0.000*	*0.000*	-	*0.000*
	`2-p-opt`	*0.000*	*0.000*	**0.000**	-
		`2.5-opt-EEs-100`	`2.5-opt-ACs`	`1-shift`	`2-p-opt`
$p = 0.8$	`2.5-opt-EEs-100`	-	0.517	**0.000**	**0.000**
	`2.5-opt-ACs`	0.517	-	**0.000**	**0.000**
	`1-shift`	*0.000*	*0.000*	-	*0.000*
	`2-p-opt`	*0.000*	*0.000*	**0.000**	-
		`2.5-opt-EEs-100`	`2.5-opt-ACs`	`1-shift`	`2-p-opt`
$p = 0.9$	`2.5-opt-EEs-100`	-	0.160	**0.000**	**0.000**
	`2.5-opt-ACs`	0.160	-	**0.000**	**0.000**
	`1-shift`	*0.000*	*0.000*	-	*0.000*
	`2-p-opt`	*0.000*	*0.000*	**0.000**	-

Table 4: Comparison of the average cost obtained by `2.5-opt-EEs-100` and by `2.5-opt-ACs`, `1-shift`, and `2-p-opt`, on clustered instances of size 300. Each algorithm is allowed to run until it reaches a local optimum. For each level of probability, the table reports the observed relative difference and a 95% confidence bound on the relative difference. This bound is obtained through a one-side paired Wilcoxon test. Concerning the relative difference, if the value is positive, `2.5-opt-EEs-100` obtained an average cost that is larger than the one obtained by the other algorithm considered; if it is negative, `2.5-opt-EEs-100` reached solutions of lower average cost. In both cases, a value is typeset in boldface if it is significantly different from zero according to a two-side paired Wilcoxon test, at a confidence of 95%. Concerning the bound, a positive value $+d$% indicates that, at a confidence of 95%, `2.5-opt-EEs-100` is not more than d% worse than the other algorithm considered; a negative value $-d$% indicates that, at a confidence of 95%, `2.5-opt-EEs-100` at least d% better.

	p-values					
	`2.5-opt-EEs-100` vs. `2.5-opt-ACs`		`2.5-opt-EEs-100` vs. `1-shift`		`2.5-opt-EEs-100` vs. `2-p-opt`	
p	Difference	Bound	Difference	Bound	Difference	Bound
0.1	$+\mathbf{1.71}\%$	$+1.98\%$	$+\mathbf{1.42}\%$	$+1.86\%$	$-\mathbf{3.25}\%$	-2.88%
0.2	$+0.28\%$	$+0.60\%$	-0.32%	$+0.26\%$	$-\mathbf{4.76}\%$	-4.42%
0.3	-0.45%	-0.01%	$-\mathbf{1.09}\%$	-0.53%	$-\mathbf{4.28}\%$	-3.90%
0.4	-0.10%	$+0.18\%$	$-\mathbf{1.99}\%$	-1.50%	$-\mathbf{3.49}\%$	-3.10%
0.5	-0.20%	$+0.27\%$	$-\mathbf{2.74}\%$	-2.21%	$-\mathbf{3.02}\%$	-2.64%
0.6	-0.30%	$+0.03\%$	$-\mathbf{3.35}\%$	-2.99%	$-\mathbf{2.40}\%$	-2.17%
0.7	-0.02%	$+0.24\%$	$-\mathbf{3.50}\%$	-2.96%	$-\mathbf{1.99}\%$	-1.64%
0.8	-0.27%	$+0.17\%$	$-\mathbf{3.87}\%$	-3.47%	$-\mathbf{1.68}\%$	-1.36%
0.9	-0.26%	$+0.04\%$	$-\mathbf{4.06}\%$	-3.66%	$-\mathbf{1.22}\%$	-0.98%

The difference is significant in a statistical sense but it is nonetheless relatively small: with a confidence of 95%, we can state that the average cost of the solutions obtained by `2.5-opt-EEs-100` is at most 1.98% higher than the one obtained by `2.5-opt-ACs`.[1] For $p = 0.2$, the observed average cost obtained by `2.5-opt-EEs-100` is slightly higher than the one of `2.5-opt-ACs` but the difference is not significant in a statistical sense. Moreover, the average cost obtained by `2.5-opt-EEs-100` is not more than 0.60% higher than the one of `2.5-opt-ACs`. For probabilities larger than 0.2, on average, `2.5-opt-EEs-100` obtains slightly better results, even if our experiments were not able to detect statistical significance. Nonetheless, should ever the average cost obtained by `2.5-opt-EEs-100` be higher than the one obtained by `2.5-opt-ACs`, the difference would be at most 0.27% for all values of probability larger than 0.2.

Similar conclusions can be drawn for what concerns the comparison between `2.5-opt-EEs-100` and `1-shift`. For $p = 0.1$, `1-shift` obtains a lower average cost than that of `2.5-opt-EEs-100`. The difference is significant in a statistical sense but it is relatively small: the average cost obtained by `2.5-opt-EEs-100` is within a bound of 1.86% of the one obtained by `1-shift`. For $p = 0.2$, the difference between the two algorithms is not significant and the average cost obtained by `2.5-opt-EEs-100` is not more than 0.26% higher than the one of `1-shift`. For probabilities larger than 0.2, `2.5-opt-EEs-100` performs significantly better than `1-shift` and, the average cost of the solutions obtained by `2.5-opt-EEs-100` is between at least 0.53% (for $p = 0.3$) and at least 3.66% (for $p = 0.9$) lower that the one of `2.5-opt-ACs`.

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

In order to highlight the impact of the speed factor of `2.5-opt-EEs-100` on the cost of the solutions, we can observe the cost of the solutions obtained by `2.5-opt-ACs` in the time needed

[1] In the same way, we make the rest of the statements concerning the observed relative difference with 95% confidence.

by 2.5-opt-EEs-100 to find the local optima. From the results, irrespective of the value of p, we can observe that the average cost of the solutions obtained by 2.5-opt-EEs-100 is between 16% and 18% lower than that of 2.5-opt-ACs. Clearly, the speed factor gives 2.5-opt-EEs-100 a significant advantage over 2.5-opt-ACs. Even though 2.5-opt-ACs and 2.5-opt-EEs-100 adopt the same neighborhood exploration and neighborhood reduction techniques, the higher speed of 2.5-opt-EEs-100 is due to the simplicity of the *estimation-based delta evaluation* procedure.

The poorer solution cost of 2.5-opt-EEs-100 for $p = 0.1$ can be attributed to the number of realizations used to estimate the cost difference between two solutions. Intuitively, the variance of the PTSP cost difference estimator depends on p and M. The smaller the value of p, the higher the variance of the cost difference estimator. For $p = 0.1$ and $M = 100$, the variance of the cost difference estimator is very high, which eventually results in a misleading estimation of the cost difference between two solutions. As a consequence, 2.5-opt-EEs-100 stops prematurely. We address this issue in Section 5.3.

5.3 Experiments on large instances

In this section, we study the performance of 2.5-opt-EEs-100 when applied to large instances. We considered PTSP instances with 1000 nodes, which are generated in the same way as described before. Since algorithms based on the *analytical computation* techniques suffer from numerical problems for $p > 0.5$ (for details, see Table 1), we compare solution cost and computation time of 2.5-opt-EEs-100 to 2.5-opt-ACs, 1-shift and 2-p-opt only for $p \leq 0.5$. In addition to these results, in the very same setting, we also study the impact of the number of realizations considered on the performance of 2.5-opt-EEs. For this purpose, we consider samples of size 10, 100, and 1000 and we denote the algorithms 2.5-opt-EEs-10, 2.5-opt-EEs-100, and 2.5-opt-EEs-1000. The results are given in Figure 11 and Table 5.

Let us first focus on the performance of 2.5-opt-EEs-100: from the results, we can observe that the percentage difference between the average cost of the solutions obtained 2.5-opt-EEs-100 and 2.5-opt-ACs exhibits a trend similar to the one observed on instances of size 300. However, the difference between the computation times of the two algorithms is very high. Regarding the time required to reach local optima, irrespective of the value of p, 2.5-opt-EEs-100 is approximately 2.3, 2.5 and 3 orders of magnitude faster than 2.5-opt-ACs, 1-shift and 2-p-opt, respectively. Concerning the average cost of local optima, 2.5-opt-EEs-100 achieves an average cost similar to that of 2.5-opt-ACs with the exception of $p = 0.1$, where the average cost of local optima obtained by 2.5-opt-EEs-100 is approximately 3% higher than that of 2.5-opt-ACs. However, 2.5-opt-EEs-100 completely dominates 1-shift and 2-p-opt.

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

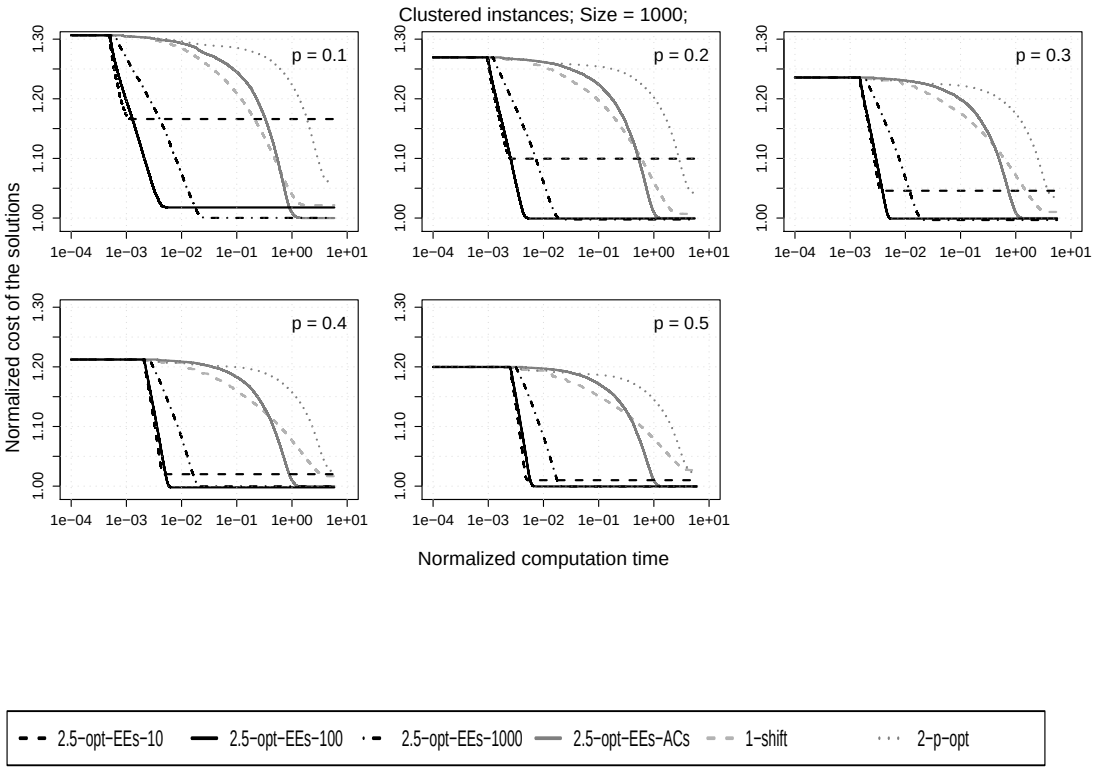


Figure 11: Experimental results on clustered homogeneous PTSP instances of size 1000. The plots represent the cost of the solutions obtained by `2.5-opt-EEs-10`, `2.5-opt-EEs-100`, `2.5-opt-EEs-1000`, `1-shift`, and `2-p-opt` normalized by the one obtained by `2.5-opt-ACs`. Each algorithm is stopped when it reaches a local optimum. Note that the algorithms based on the *analytical computation* techniques do not produce meaningful results for $p > 0.5$ due to the numerical problem (for details, see Table 1). The algorithms based on the *empirical estimation* do not suffer from this problem.

Table 5: Experimental results for `2.5-opt-EEs-10`, `2.5-opt-EEs--100`, `2.5-opt-EEs-1000`, `2.5-opt-ACs`, `2-p-opt` and `1-shift` on clustered instances of size 1000. Each algorithm is allowed to run until it reaches a local optimum. The table gives mean and standard deviation (s.d.) of final solution cost and computation time in seconds. The results are given for 100 instances at each probability level. The symbol × indicates that the algorithms do not produce meaningful results due to the numerical problem (for details, see Table 1). The algorithms based on the *empirical estimation* do not suffer from this problem.

	Algorithm	Solution Cost		Computation Time	
		mean	s.d.	mean	s.d.
$p = 0.1$	`2.5-opt-EEs-10`	5909938	461029	0.462	0.030
	`2.5-opt-EEs-100`	5158215	448385	1.839	0.190
	`2.5-opt-EEs-1000`	5069560	424448	9.487	1.170
	`2.5-opt-ACs`	5068223	450709	443.952	70.934
	`1-shift`	5178144	469977	635.757	84.010
	`2-p-opt`	5365486	449318	1464.535	341.993
$p = 0.2$	`2.5-opt-EEs-10`	7364518	513937	0.524	0.032
	`2.5-opt-EEs-100`	6692459	486598	1.024	0.092
	`2.5-opt-EEs-1000`	6681179	475423	3.981	0.447
	`2.5-opt-ACs`	6697814	480609	229.288	33.165
	`1-shift`	6744906	494658	547.263	71.878
	`2-p-opt`	6978843	477590	859.276	159.102
$p = 0.3$	`2.5-opt-EEs-10`	8263425	554699	0.507	0.030
	`2.5-opt-EEs-100`	7894854	547385	0.722	0.051
	`2.5-opt-EEs-1000`	7875735	511413	2.658	0.306
	`2.5-opt-ACs`	7901717	524412	149.881	22.702
	`1-shift`	7982498	531787	451.773	58.575
	`2-p-opt`	8175022	547812	552.554	95.447
$p = 0.4$	`2.5-opt-EEs-10`	9025641	585240	0.459	0.031
	`2.5-opt-EEs-100`	8830141	596439	0.600	0.041
	`2.5-opt-EEs-1000`	8846373	594006	2.026	0.198
	`2.5-opt-ACs`	8848198	549139	106.665	17.015
	`1-shift`	8995824	567472	374.877	50.938
	`2-p-opt`	9060142	551377	422.211	76.872
$p = 0.5$	`2.5-opt-EEs-10`	9693061	630069	0.422	0.020
	`2.5-opt-EEs-100`	9592605	623310	0.526	0.038
	`2.5-opt-EEs-1000`	9591076	635788	1.689	0.149
	`2.5-opt-ACs`	9597432	599270	89.272	14.155
	`1-shift`	9856073	579796	316.049	44.883
	`2-p-opt`	9799426	594452	338.203	63.679
$p = 0.6$	`2.5-opt-EEs-10`	10282713	609678	0.391	0.018
	`2.5-opt-EEs-100`	10238620	648265	0.481	0.027
	`2.5-opt-EEs-1000`	10264821	637652	1.444	0.129
	`2.5-opt-ACs`	×	×	×	×
	`1-shift`	×	×	×	×
	`2-p-opt`	×	×	×	×
$p = 0.7$	`2.5-opt-EEs-10`	10849469	680716	0.370	0.017
	`2.5-opt-EEs-100`	10809915	627073	0.446	0.024
	`2.5-opt-EEs-1000`	10782974	669036	1.280	0.108
	`2.5-opt-ACs`	×	×	×	×

	1-shift	×	×	×	×
	2-p-opt	×	×	×	×
$p = 0.8$	2.5-opt-EEs-10	11323598	678658	0.355	0.015
	2.5-opt-EEs-100	11325374	667811	0.424	0.020
	2.5-opt-EEs-1000	11326928	665397	1.126	0.086
	2.5-opt-ACs	×	×	×	×
	1-shift	×	×	×	×
	2-p-opt	×	×	×	×
$p = 0.9$	2.5-opt-EEs-10	11754423	697164	0.346	0.013
	2.5-opt-EEs-100	11764130	697751	0.406	0.019
	2.5-opt-EEs-1000	11764290	705736	1.016	0.073
	2.5-opt-ACs	×	×	×	×
	1-shift	×	×	×	×
	2-p-opt	×	×	×	×

5.4 Experiments on sampling strategies

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

In `2.5-opt-EEs-100-ri` and `2.5-opt-EEs-100-rc`, for $p = 0.1$, the search cycles between solutions due to the high variance of the cost difference estimator. To avoid this problem, we implemented a mechanism that for $p = 0.1$ memorizes moves in order to reject them in successive search steps. The results on clustered instances with 300 nodes are given in Figure 12.

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

5.5 Experiments with iterated local search

In this section, we study the behavior of `2.5-opt-EEs` and `2.5-opt-ACs` embedded into iterated local search (ILS) (Lourenço et al., 2002), a metaheuristic on which many state-of-the-art algorithms for the TSP are based (Hoos and Stützle, 2005). We implemented a standard ILS algorithm that accepts only improving local optima and that uses a random double-bridge move for the perturbation of local optima. We denote the two algorithms (ILS versions) `ILS-2.5-opt-EEs` and `ILS-2.5-opt-ACs`, respectively. For `ILS-2.5-opt-EEs`, we use 100 and 1000 realizations; we denote these algorithms `ILS-2.5-opt-EEs-100` and `ILS-2.5-opt-EEs-1000`. Note that the set of realizations is kept unchanged throughout the search.

The stopping criteria for the considered algorithms is the following: `ILS-2.5-opt-ACs` is run until it performs 25 perturbations and the time needed for completion is recorded; this time is then taken as the time limit for `ILS-2.5-opt-EEs`. The results on clustered instances with 1000

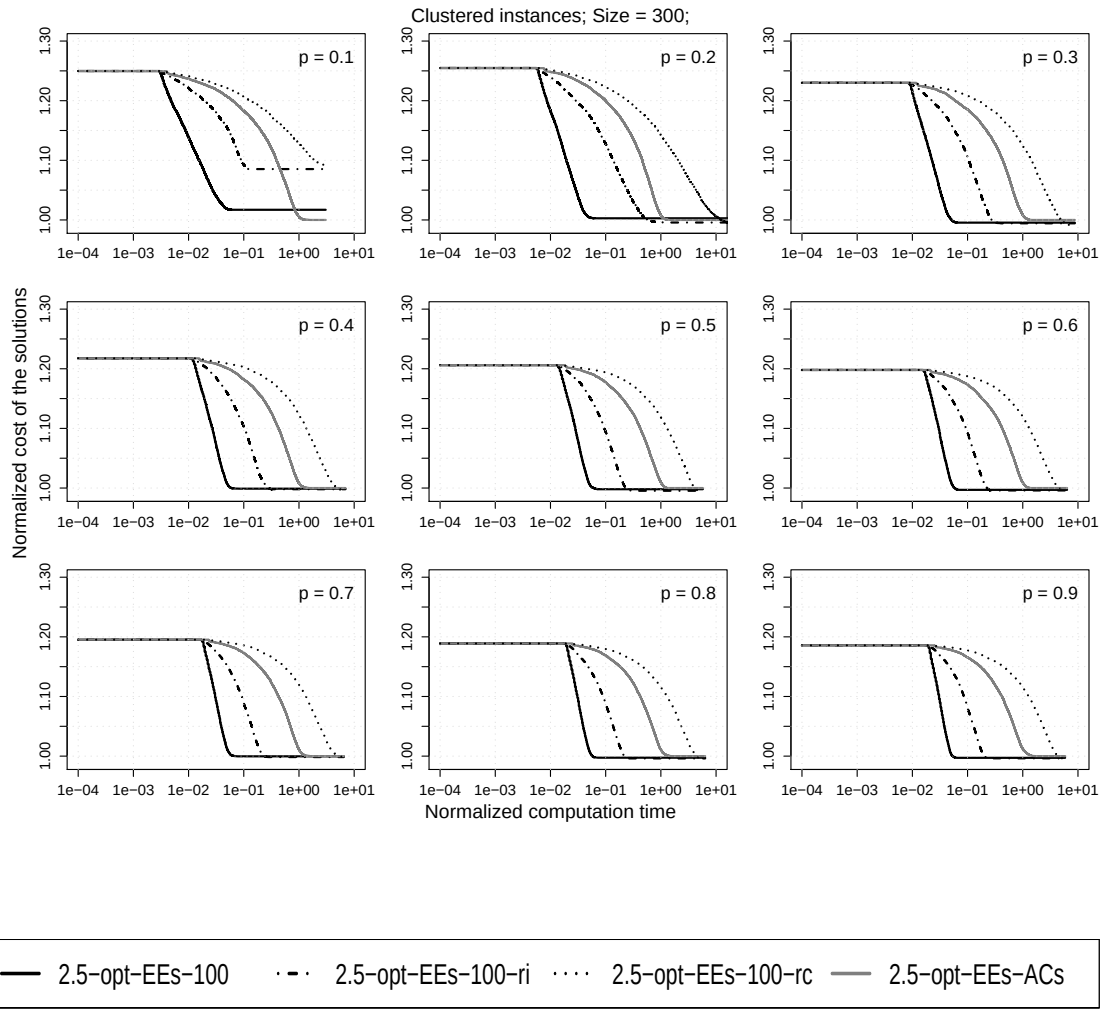


Figure 12: Experimental results on clustered homogeneous PTSP instances of size 300. The plots represent the average cost of the solutions obtained by `2.5-opt-EEs-100`, `2.5-opt-EEs-100-ri`, `2.5-opt-EEs-100-rc` normalized by the one obtained by `2.5-opt-ACs`. Each algorithm is stopped when it reaches a local optimum.

Table 6: Experimental results for `ILS-2.5-opt-EEs-100`, `ILS-2.5-opt-EEs-1000`, and `ILS--2.5-opt-ACs` on clustered instances of size 1000. The table gives mean and standard deviation (s.d.) of final solution cost and computation time in seconds. The results are given for 100 instances at each probability level. Note that the algorithms based on the *analytical computation* techniques do not produce meaningful results for $p > 0.5$ due to the numerical problem (for details, see Table 1). The algorithms based on the *empirical estimation* do not suffer from this problem.

	Algorithm	Solution Cost		Computation Time	
		mean	s.d.	mean	s.d.
$p = 0.1$	`ILS-2.5-opt-EEs-100`	4979874	399713	3261.500	362.575
	`ILS-2.5-opt-EEs-1000`	4843776	394002		
	`ILS-2.5-opt-ACs`	4877318	396493		
$p = 0.2$	`ILS-2.5-opt-EEs-100`	6236305	433606	1933.850	200.543
	`ILS-2.5-opt-EEs-1000`	6212680	435542		
	`ILS-2.5-opt-ACs`	6376838	452242		
$p = 0.3$	`ILS-2.5-opt-EEs-100`	7244262	470112	1306.395	144.981
	`ILS-2.5-opt-EEs-1000`	7259346	471874		
	`ILS-2.5-opt-ACs`	7511449	492782		
$p = 0.4$	`ILS-2.5-opt-EEs-100`	8073761	501743	1017.768	115.292
	`ILS-2.5-opt-EEs-1000`	8120875	502695		
	`ILS-2.5-opt-ACs`	8450032	532982		
$p = 0.5$	`ILS-2.5-opt-EEs-100`	8787387	532259	842.500	97.555
	`ILS-2.5-opt-EEs-1000`	8845621	525797		
	`ILS-2.5-opt-ACs`	9235088	560814		

nodes are given in Figure 13 and Table 6.

Concerning the average cost of the solutions obtained, `ILS-2.5-opt-EEs-100` is between 1% and 3% better than `ILS-2.5-opt-ACs`, except for $p = 0.1$, where the average cost of `ILS-2.5-opt--ACs` is approximately 1% lower than that of `ILS-2.5-opt-EEs-100`. On the other hand, `ILS--2.5-opt-EEs-1000` outperforms `ILS-2.5-opt-ACs` for all values of p: the average cost reached by the former is between 1% and 4% lower than that of the latter.

The results of the comparison of `ILS-2.5-opt-EEs-100` and `ILS-2.5-opt-EEs-1000` show that the average cost reached by the latter is between 0.2% and 2% better than that of the former for $p \leq 0.2$. Since the variance of the cost difference estimator is high at this probability range, the use of a large number of realizations results in a more precise estimator, which eventually leads to solutions of lower cost. Nevertheless, for $p \geq 0.3$, the average cost of `ILS-2.5-opt-EEs-100` is between 0.2% and 0.6% better than that of `ILS-2.5-opt-EEs-1000`. Since the variance of the cost difference estimator is low at this probability range, the use of 100 realizations instead of 1000 allows `ILS-2.5-opt-EEs-100` to perform more iterations than `ILS-2.5-opt-EEs-1000`, which in turn results in solutions of lower cost.

Note that the observed differences between the algorithms are statistically significant according to a paired Wilcoxon test with p-values adjusted by Holm's method, with a confidence of 95%. Since all the p-values obtained from the pairwise comparisons are less than `2.2e-16`, we do not present them in a table.

We also implemented another sampling strategy for the *estimation-based* ILS in which the set of realizations is sampled anew for each iteration of ILS (after each perturbation). However, the results did not show any significant difference from the one presented in this section.

6 Conclusions and Future Work

We introduced an *estimation-based* iterative improvement algorithm for the PTSP. The main novelty of our approach consists in using the *empirical estimation* techniques in the *delta evaluation* procedure. The proposed approach is conceptually simple, easy to implement, scalable to large instance sizes and can be applied to problems in which the cost difference cannot be expressed in

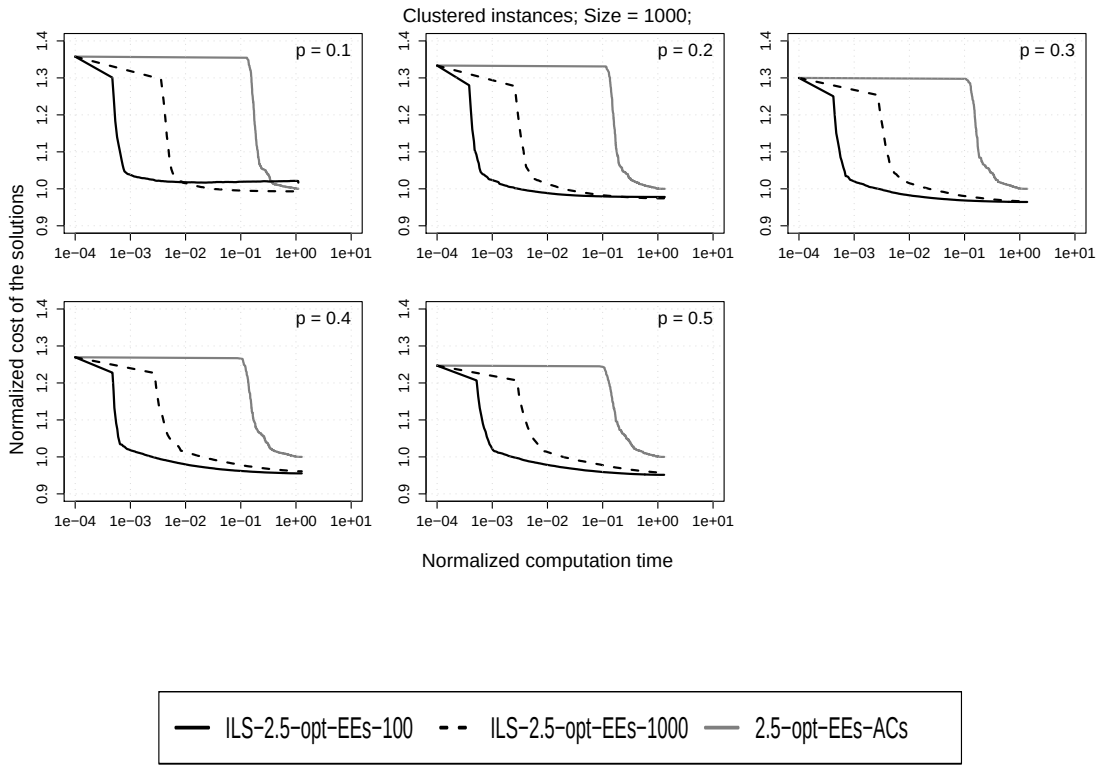


Figure 13: Experimental results on clustered homogeneous PTSP instances of size 1000. The plots represent the cost of the solutions obtained by `ILS-2.5-opt-EEs-100` and `ILS-2.5-opt--EEs-1000` normalized by the one obtained by `ILS-2.5-opt-ACs`. Note that the algorithms based on the *analytical computation* techniques do not produce meaningful results for $p > 0.5$ due to the numerical problem (for details, see Table 1). The algorithms based on the *empirical estimation* do not suffer from this problem.

a closed-form. Moreover, we have shown that the TSP-specific neighborhood reduction techniques are also effective for the PTSP. Furthermore, we identified a major practical limitation in applying the current state-of-the-art iterative improvement algorithms to large PTSP instances.

There are a large number of avenues for further research. The performance of the proposed approach is affected by the number of realizations considered. In this context, our future work will aim at developing adaptive sampling procedures that save computational time by selecting the most appropriate number of realizations with respect to the variance of the cost difference estimator.

Given the promising results obtained by the iterated local search presented in Section 5.5, further research will be devoted to assess the behavior of the proposed approach when used as an embedded heuristic in metaheuristics such as ant colony optimization and genetic algorithms.

From the application perspective, the *estimation-based* iterative improvement algorithms will be applied to more complex problems such as stochastic vehicle routing, stochastic scheduling, and TSP with time windows and stochastic service time.

Acknowledgments

The authors thank Leonora Bianchi for providing the source code of `2-p-opt` and `1-shift`. This research has been supported by COMP^{2}SYS, a Marie Curie Early Stage Research Training Site funded by the European Community's Sixth Framework Programme under contract number MEST-CT-2004-505079, and by the ANTS project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Prasanna Balaprakash, Thomas Stützle, and Marco Dorigo acknowledge support from the Belgian FNRS of which they are an Aspirant, a Research Associate, and a Research Director, respectively. The information provided is the sole responsibility of the authors and does not reflect the opinion of the sponsors. The European Community is not responsible for any use that might be made of data appearing in this publication.

References

754, IEEE. 1985. IEEE Standard for Binary Floating-Point Arithmetic.

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

Bertsimas, D. 1988. Probabilistic combinatorial optimization problems. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA.

Bertsimas, D., L. Howell. 1993. Further results on the probabilitsic traveling salesman problem. *European Journal of Operations Research* **65** 68–95.

Bertsimas, D., P. Jaillet, A. Odoni. 1990. A priori optimization. *Operations Research* **38** 1019–1033.

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., M. Birattari, M. Chiarandini, M. Manfrin, M. Mastrolilli, L. Paquete, O. Rossi-Doria, T. Schiavinotto. 2006. Hybrid metaheuristics for the vehicle routing problem with stochastic demands. *Journal of Mathematical Modelling and Algorithms* **5** 91–110.

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

Bianchi, L., J. Knowles, N. Bowler. 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** 206–219.

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

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

Fu, M. C. 1994. Optimization via simulation: A review. *Annals of Operations Research* **53** 199–248.

Fu, M. C. 2002. Optimization for simulation: theory vs. practice. *INFORMS Journal on Computing* **14** 192–215.

Griffith, A. 2002. *GCC: The Complete Reference*. McGraw Hill/Osborne Media.

Gutjahr, W.J. 2004. S-ACO: An ant based approach to combinatorial optimization under uncertainity. M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, T. Stützle, eds., *Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2004*, *LNCS*, vol. 3172. Springer-Verlag, Berlin, Germany, 238–249.

Hoos, H., T. Stützle. 2005. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann.

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

Johnson, D. S., L. A. McGeoch. 1997. The travelling salesman problem: A case study in local optimization. E. H. L. Aarts, J. K. Lenstra, eds., *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, United Kingdom, 215–310.

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

Lourenço, H. R., O. Martin, T. Stützle. 2002. Iterated local search. F. Glover, G. Kochenberger, eds., *Handbook of Metaheuristics*, *International Series in Operation Research and Management Science*, vol. 57. Kluwar Academic Publishers, Norwell, MA, 321–353.

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

Penky, J. F., D.L. Miller. 1994. A staged primal-dual algorithm for finding a minimum cost perfect two-matching in an undirected graph. *ORSA Jornal on Computing* **6** 68–81.

Pichitlamken, J., B. L. Nelson. 2003. A combined procedure for optimization via simulation. *ACM Transactions on Modeling and Computer Simulation* **13** 155–179.