INFORMS Journal on Computing
Vol. 20, No. 4, Fall 2008, pp. 644–658
ISSN 1091-9856 | EISSN 1526-5528 | 08 | 2004 | 0644

DOI 10.1287/ijoc.1080.0276

Estimation-Based Local Search for Stochastic Combinatorial Optimization Using Delta Evaluations: A Case Study on the Probabilistic Traveling Salesman Problem

Mauro Birattari, Prasanna Balaprakash, Thomas Stützle, Marco Dorigo
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium 1050
{mbiro@ulb.ac.be, pbalapra@ulb.ac.be, stuetzle@ulb.ac.be, mdorigo@ulb.ac.be}

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 this approach and we assess its performance on the probabilistic traveling salesman problem. Experimental results on a large set of instances show that the proposed approach can lead to a very fast and highly effective local search algorithm.

Key words: stochastic combinatorial optimization; suboptimal algorithms; iterative improvement; simulation
History: Accepted by Michel Gendreau, Area Editor for Heuristic Search and Learning; received February 2007; revised September 2007; accepted February 2008. Published online in *Articles in Advance* August 20, 2008.

1. Introduction

When tackling a large number of practically relevant combinatorial optimization problems, only a part of the information needed for evaluating the quality of a solution might be available. Examples include portfolio management, vehicle routing, resource allocation, scheduling, and the modeling and simulation of large molecular systems in bioinformatics (Fu 2002). 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*. Whereas the former relies on a complex analytical development for computing the expectation, the latter simply estimates it 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 with 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). Moreover, certain algorithms based on local search can even be shown to converge to the optimum with probability one (Gutjahr 2003). 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, which computes the cost of each solution from scratch, and (ii) partial evaluation, which 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 feasible (Bertsimas 1988).

The delta evaluation strategies proposed in the stochastic optimization literature for iterative

improvement algorithms are based on analytical computation (Bertsimas 1988, Bianchi 2006). In this case, the cost difference between two solutions is given by a closed-form expression that is obtained through problem-specific knowledge and rather complex analytical developments. The main drawbacks of these 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 (PTSP) as an example to illustrate the proposed approach and to assess its performance.

The empirical estimation approach for stochastic combinatorial optimization falls into the *sample average approximation* framework (Kleywegt et al. 2002): the given stochastic optimization problem is transformed into a so-called sample average optimization problem, which is obtained by considering several realizations of the random variable and by approximating the cost of a solution with a sample average function. In the context of stochastic routing problems, where the PTSP is considered as a paradigmatic example, this framework has been shown to be very effective (Verweij et al. 2003). The main novelty of the approach we propose in this paper is the adoption of delta evaluation within the sample average approximation framework. This is particularly useful in certain classes of stochastic combinatorial optimization problems, where the local modifications in a solution entail only local modifications in the estimation of its cost.

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

2. Stochastic Combinatorial Optimization Problems

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

$$\begin{aligned} &\text{Minimize} \quad F(x) = E[f(x, \boldsymbol{\omega})], \\ &\text{subject to} \quad x \in S, \end{aligned} \tag{1}$$

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

A paradigmatic example of a stochastic combinatorial optimization problem is the 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 set of costs associated with edges, 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 $\boldsymbol{\omega}$ is described by an n-variate Bernoulli distribution and a realization of $\boldsymbol{\omega}$ is a binary vector of size n where a "1" in position i indicates that node i requires a visit and a "0" indicates that it does not. We assume that the costs are symmetric; that is, traveling from a node i to j has the same cost as traveling from node j to i. A solution to the PTSP is a permutation of the nodes.

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 is determined before the actual realization of the random variable $\boldsymbol{\omega}$ is available. This is the so-called a priori solution. In the second stage, after the realization of the random variable is known, an a posteriori solution is obtained from the a priori solution by visiting the nodes prescribed by the given realization in the order in which they appear in the a priori solution. The nodes that do not require a visit are simply skipped. Figure 1 shows an example. It should be noted that because the travel costs are symmetric, the cost of the a posteriori solution is invariant with respect to the orientation. For example, in Figure 1, the cost of the a posteriori solution does not change by visiting the nodes in the counter-clockwise direction.

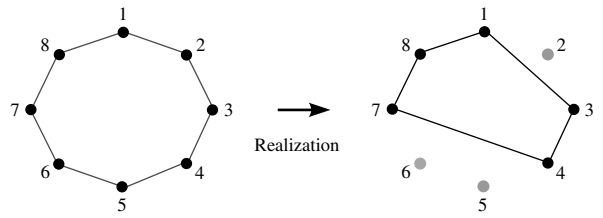

Figure 1 An A Priori Solution for a PTSP Instance with Eight Nodes

Notes. 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 $\boldsymbol{\omega}$ prescribes that nodes 1, 3, 4, 7, and 8 are to be visited. The resulting a posteriori solution is obtained by visiting the nodes in the order in which they appear in the a priori solution and by skipping the nodes 2, 5, and 6, which do not require being visited.

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 $P = \{p_i = p\colon i \in V\}$, 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 of moving from one solution to a 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***: 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***: 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). Whereas in the former an improving move is immediately applied as soon as it is detected, in the latter the whole neighborhood is examined and a move that gives the best improvement is chosen.

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

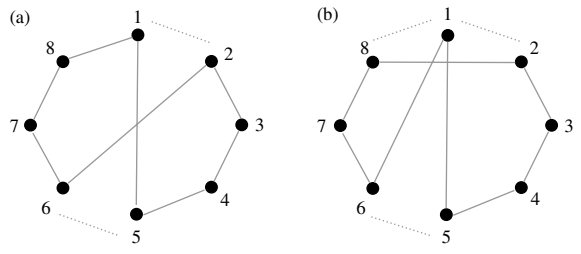

Figure 2 **Plot 2(a) Shows a 2-Exchange Move: Two Edges $\langle 1, 2\rangle$ and $\langle 5, 6\rangle$ Are Deleted and Replaced with $\langle 1, 5\rangle$ and $\langle 2, 6\rangle$; Plot 2(b) Shows a Node-Insertion Move: Node 1 Is Moved and Inserted Between Nodes 5 and 6**

3.1. State-of-the-Art Iterative Improvement Algorithms for the PTSP

For the homogeneous PTSP, Bertsimas (1988) 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: `2-p-opt` and `1-shift`. For both algorithms, the total time complexity of the neighborhood exploration and evaluation is $O(n^2)$. For the heterogeneous case, Chervi (1988) proposed closed-form delta evaluation expressions for `2-p-opt` and `1-shift`, where each algorithm explores and evaluates the neighborhood in $O(n^3)$. Bianchi et al. (2005) and Bianchi and Campbell (2007) proved that the expressions derived by Bertsimas (1988) and Chervi (1988) are incorrect, and corrected the errors. Furthermore, for the heterogeneous PTSP, 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) also considered 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. The experimental results 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 of exploring a special case of the 2-exchange neighborhood, the *swap-neighborhood*, where the neighbors of the current solution are all those that can be obtained by swapping two consecutive nodes. If the swap-neighborhood is fully explored and no improvement is found, a second phase explores the 2-exchange neighborhood with the first-improvement rule. It should be noted that the

neighborhood is explored in a fixed lexicographic order by considering pairs of edges that are separated by a fixed number k of nodes. Starting with $k = 2$, the lexicographic exploration proceeds by incrementing k, and whenever an improvement is found, the search is restarted from the first phase: the swap-neighborhood exploration. Note that the swap-neighborhood is explored with the first-improvement rule. `1-shift` differs from `2-p-opt` only in the second phase: it uses the node-insertion neighborhood with the best-improvement rule.

We refer the reader to Bianchi et al. (2005) and to Bianchi and Campbell (2007) for the 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 the aforementioned papers (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 that 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. Issues with the State-of-the-Art Iterative Improvement Algorithms for the PTSP

Theoretically, the delta evaluation expressions proposed by Bianchi et al. (2005) can be applied to solve PTSP instances of any size. However, in practice, these expressions suffer from numerical problems when applied to large instances. In fact, to use the delta evaluation expressions in `2-p-opt`, the term $(1-p)^{(k-n)}$ has to be computed, where p is the probability, k is the number of nodes between two considered edges, and n is the size of the instance. For some values of p, k, and n, this term can result in an overflow. As an illustration, consider a typical 32-bit GNU system, where, according to the IEEE 754 standard (IEEE 1985), double-precision floating-point variables can take a maximum value of about $1e+308$ (Griffith 2002). Given a homogeneous PTSP instance of probability p with n nodes, the condition for the numerical overflow is $(1-p)^{(k-n)} > 1e+308$. From this condition, one can obtain, after basic transformations, a critical value for n, above which the computation of the delta evaluation expression suffers from precision problems: $n_{\text{critical}} = 1 - (308/\log_{10}(1-p))$.

Table 1 **For a Given Probability p, the Maximum Size of the Instance that Can be Handled by `2-p-opt` on a 32-Bit System Without Resorting to Arbitrary Precision Arithmetics**

p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
n_{critical}	6,733	3,180	1,990	1,390	1,025	775	591	442	309

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 on a 32-bit system. The very same problem occurs in `1-shift` and in the analytical computation algorithms for the heterogeneous PTSP. This issue, which has never been pointed out before in the literature, has to be addressed by resorting to methods for arbitrary precision arithmetics. As we show in §5.3, this might entail a major computational overhead.

A second issue with 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 speed up the search significantly.

4. Estimation-Based Iterative Improvement Algorithms

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

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

As 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 $\boldsymbol{\omega}$. Eventually, $\hat{F}_{M'}(x') - \hat{F}_M(x)$ is an unbiased estimator of $F(x') - F(x)$.

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 of using the

same set of realizations of $\boldsymbol{\omega}$ for estimating the costs $F(x')$ and $F(x)$. Consequently, we have $M' = M$, and the estimator $\hat{F}_M(x') - \hat{F}_M(x)$ of the cost difference is given by

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

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 §5.4.

Using Equation (3), 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 of computing first the complete cost of each a posteriori solution and then the difference between them. However, a more efficient algorithm can be obtained by adopting the idea of delta evaluation: Given the a priori solutions and a realization ω, such an algorithm requires identifying the edges that are not common to the two a posteriori solutions.

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

In general, for every edge $\langle i, j\rangle$ that is deleted from x, one needs to find the corresponding edge $\langle i^*, j^*\rangle$ in the a posteriori solution. We call this edge the a posteriori edge. It is obtained as follows: if node i has to be visited, then $i^* = i$; otherwise, i^* is the first predecessor of i in x such that $\omega[i^*] = 1$. If node j has to be visited, then $j^* = j$; otherwise, j^* is the first successor of j such that $\omega[j^*] = 1$. Recall that in a 2-exchange move, the edges $\langle a, b\rangle$ and $\langle c, d\rangle$ are replaced by $\langle a, c\rangle$ and $\langle b, d\rangle$. For the a posteriori edges $\langle a^*, b^*\rangle$ and $\langle c^*, d^*\rangle$, the cost difference between the two a posteriori solutions is $c_{a^*,c^*} + c_{b^*,d^*} - c_{a^*,b^*} - c_{c^*,d^*}$. Figure 4 shows the a posteriori edges for the example given in Figure 3. 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 edges $\langle a, b\rangle$ and $\langle c, d\rangle$, and where no node between the nodes b and c or between nodes a and d requires being visited, the difference between the two a posteriori solutions is zero—see Figure 6(a); in a node-insertion move, if the insertion node does not require being visited, then the cost difference between the two a posteriori solutions is zero—see Figure 6(b). In this second case, one can avoid unnecessary computations by checking whether the insertion node requires being visited before finding the a posteriori edges.

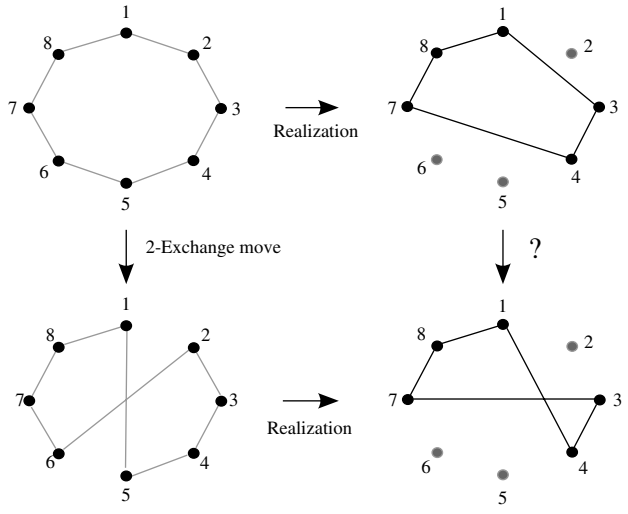

Figure 3 In This Example, a 2-Exchange Move Is Obtained by Replacing $\langle 1, 2\rangle$ and $\langle 5, 6\rangle$ in the A Priori Solution with $\langle 1, 5\rangle$ and $\langle 2, 6\rangle$

Notes. Assume that a realization of $\boldsymbol{\omega}$ prescribes that nodes 1, 3, 4, 7, and 8 are to be visited. The edges that are not common to the a posteriori solutions are $\langle 1, 3\rangle$, $\langle 4, 7\rangle$, and $\langle 1, 4\rangle$, $\langle 3, 7\rangle$. The delta evaluation procedure needs to identify these edges without considering the complete a posteriori solutions.

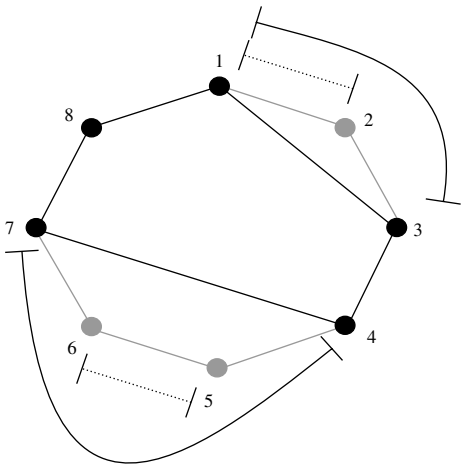

Figure 4 The Steps Performed for Finding the A Posteriori Edges

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

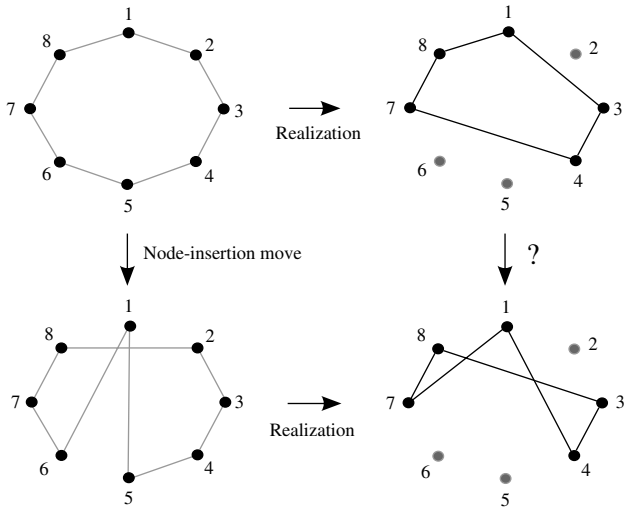

Figure 5 In This Example, the Node-Insertion Move Is Obtained by Inserting Node 1 Between Nodes 5 and 6

Notes. Consequently, the edges $\langle 8, 1\rangle$, $\langle 1, 2\rangle$, and $\langle 5, 6\rangle$ in the a priori solution are replaced with $\langle 8, 2\rangle$, $\langle 5, 1\rangle$, and $\langle 1, 6\rangle$. Assume that a realization of $\boldsymbol{\omega}$ prescribes that nodes 1, 3, 4, 7, and 8 are to be visited. The edges that are not common to the a posteriori solutions are $\langle 1, 3\rangle$, $\langle 4, 7\rangle$, $\langle 8, 1\rangle$ and $\langle 8, 3\rangle$, $\langle 4, 1\rangle$, $\langle 1, 7\rangle$.

The proposed approach has a number of advantages: First, the estimation-based delta evaluation procedure—in particular, the procedure for finding a posteriori edges—can be applied to any neighborhood structure without requiring the complex error-prone mathematical derivations that have to be performed when adopting the analytical computation approach. In virtue of this versatility, rather than using the node-insertion neighborhood or the 2-exchange neighborhood structure, we can use a hybrid neighborhood structure that includes the node-insertion neighborhood on top of the 2-exchange neighborhood structure. In the TSP literature (Bentley 1992), this hybrid neighborhood is widely known as the 2.5-exchange neighborhood: when checking for a 2-exchange move on any two edges $\langle a, b\rangle$ and $\langle c, d\rangle$, it is also checked whether deleting any one of the nodes of an edge, say for example a, and inserting it between nodes c and d results in an improved solution (Bentley 1992).

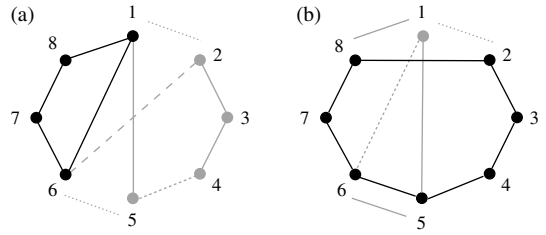

Figure 6 Some Degenerate Cases that Can Occur in the Evaluation of Cost Differences

Notes. (a) Assume that a realization of $\boldsymbol{\omega}$ prescribes that nodes 1, 6, 7, and 8 are to be visited. The 2-exchange neighboring solutions shown in Figure 2(a) lead to the same a posteriori solution. The cost difference is therefore zero. (b) Assume that a realization of $\boldsymbol{\omega}$ prescribes that nodes 2, 3, 4, 5, 6, 7, and 8 are to be visited. The node-insertion neighboring solutions shown in Figure 2(b) lead to the same a posteriori solution. Because the two a posteriori solutions are the same, the cost difference is zero.

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

We denote the proposed algorithm `2.5-opt-EEs`, where `EE` and `s` stand for *empirical estimation* and *speedup*, respectively. Note that `2.5-opt-EEs` uses the first-improvement rule. To implement `2.5-opt-EEs` effectively, a specific data structure is needed. We use a structure in which data items can be accessed both as elements of a doubly circularly linked list and as elements of a one-dimensional array, both of size n. Each data item comprises an integer variable to store a node of the a priori solution. This structure is efficient for finding a posteriori edges: predecessor and successor of a node are simply obtained by following the links pointing towards the previous item and next item, respectively. To access data items as elements of the array, a data item representing node i is stored at position i of the array, and this arrangement is kept unchanged throughout the search process. Consequently, given a node i, its data can be accessed in $O(1)$ time. Moreover, each data item stores an array of size M—the realization array—which is indexed from one to M. Element r of the realization array is either one or zero, indicating whether the node requires being visited or not in realization ω_r. Figure 7 shows the data structure. Whenever an improved solution is found, only the links of the particular data items whose nodes are involved in the exchange move are modified. Furthermore, each data item comprises also two auxiliary fields for the neighborhood reduction techniques: one integer variable for the don't look bit and one integer array of size L for the candidate list of each node.

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

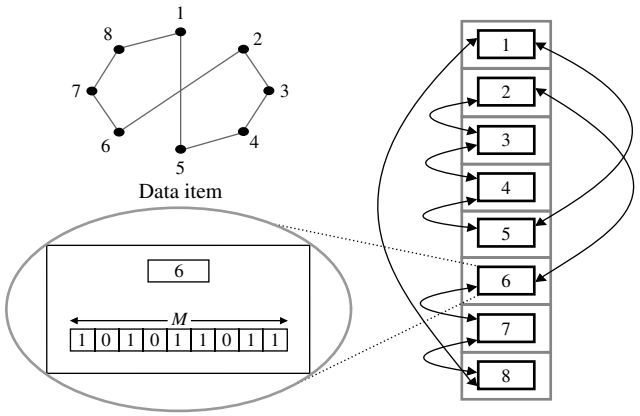

Figure 7 Assume that We Have an A Priori Solution in Which the Nodes Are Visited in the Order 1, 5, 4, 3, 2, 6, 7, 8, 1

Notes. 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.

the complexity the probability p that a node requires being visited, but the inclusion of this term is not completely justified and is not thoroughly discussed in her work. 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 $O(n^2LM)$. It should be noted that in large TSP instances, the value of L is typically chosen independent of instance size and is usually set between 20 and 40 (Johnson and McGeoch 1997). We expect that these conclusions hold also for the PTSP. Furthermore, it should be observed that because 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

We base our analysis on *homogeneous* PTSP instances that we obtained from TSP instances generated with the DIMACS instance generator (Johnson et al. 2001). We carried out experiments with two classes of instances. In the first class, nodes are *uniformly distributed* in a square; in the second, nodes are arranged in *clusters*. For each instance class, we generated 100 TSP instances of 100, 200, 300, and 1,000 nodes. Note that 300 is the largest instance size that has been used by Bianchi (2006). From each TSP instance, we obtained nine 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 of 300 and 1,000 nodes for certain probability levels. The general trends of the experimental results obtained on the other instance class 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™244 1.75 GHz 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): for each node i, a coordinate system is defined with origin in node i. The candidate list of node i then contains for each quadrant the 10 cities that are connected to i by the 10 edges of least cost. If fewer than 10 nodes are available in a quadrant, the list is filled with nodes from other quadrants. Each iterative improvement algorithm is run until it reaches a local optimum. The number of realizations in `2.5-opt-EEs` is set to 100. To highlight this fact, we denote the algorithm `2.5-opt-EEs-100`.

For the homogeneous PTSP with probability p and size n, given an a priori solution x, the exact cost $F(x)$ of x can be computed using a closed-form expression: $F(x) = \sum_{u=1}^{n} \sum_{v=1}^{n-1} p^2(1-p)^{v-1} c_{(x(u),\, x(u+v))}$, where $x(u)$ and $x(v)$ are the nodes of index u and v in x, respectively (Jaillet 1985). We use this formula for the post evaluation of the best-so-far solutions found by each algorithm according to its evaluation procedure.

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

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

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

The increased speed of 2.5-opt-ACs also shows that the amount of computational time saved due to the use of neighborhood reduction techniques is much higher than the time that is lost in computing the cost difference from scratch. Regardless 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 comparable with—1-shift and

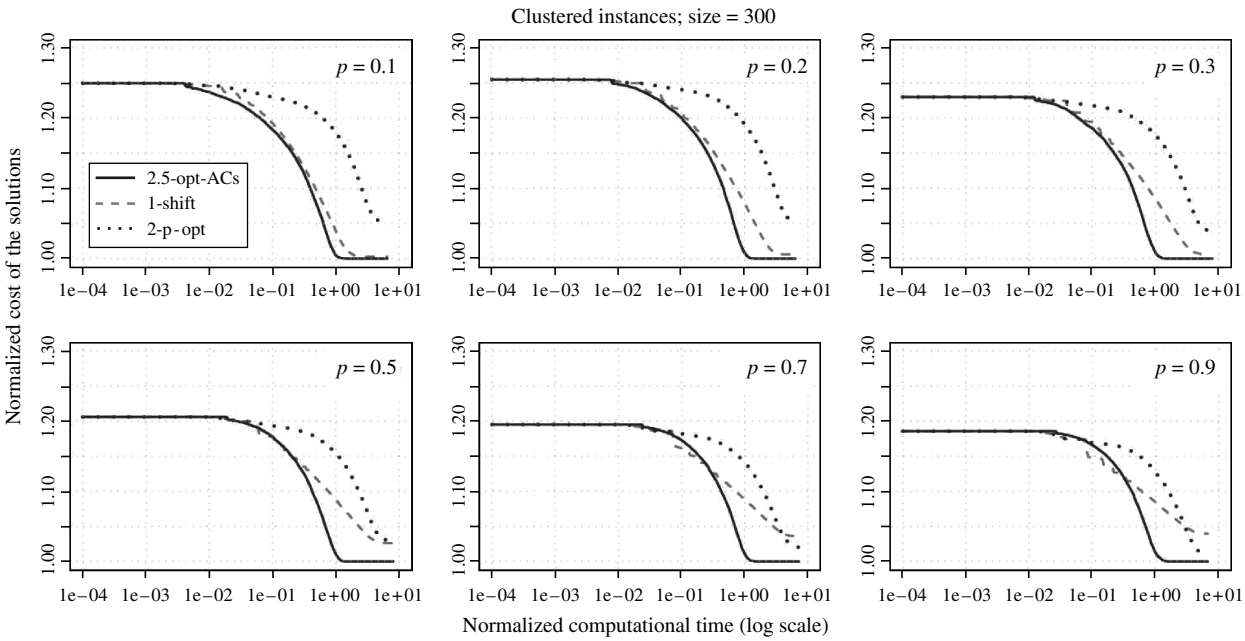

Figure 8 Experimental Results on Clustered Homogeneous PTSP Instances of Size 300

Notes. 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.

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**

	Solution cost		Computation time	
Algorithm	Mean	s.d.	Mean	s.d.
$p = 0.1$				
2.5-opt-EEs-100	2,776,865	456,487	0.120	0.014
2.5-opt-ACs	2,730,221	454,321	6.453	1.067
1-shift	2,738,026	450,970	11.771	2.404
2-p-opt	2,870,013	462,383	22.440	5.831
$p = 0.2$				
2.5-opt-EEs-100	3,595,283	467,721	0.086	0.011
2.5-opt-ACs	3,585,254	471,967	3.413	0.540
1-shift	3,606,878	467,069	10.103	1.773
2-p-opt	3,775,106	474,269	13.848	3.254
$p = 0.3$				
2.5-opt-EEs-100	4,239,788	499,001	0.064	0.008
2.5-opt-ACs	4,259,032	501,810	2.214	0.399
1-shift	4,286,461	481,061	8.478	1.856
2-p-opt	4,429,328	497,857	9.842	2.462
$p = 0.5$				
2.5-opt-EEs-100	5,190,835	537,186	0.046	0.005
2.5-opt-ACs	5,201,221	557,895	1.421	0.230
1-shift	5,336,979	544,328	5.933	1.355
2-p-opt	5,352,456	553,028	5.978	1.616
$p = 0.7$				
2.5-opt-EEs-100	5,874,044	579,475	0.038	0.004
2.5-opt-ACs	5,875,100	579,015	1.127	0.209
1-shift	6,087,249	597,356	4.851	1.056
2-p-opt	5,993,481	589,339	4.776	1.146
$p = 0.9$				
2.5-opt-EEs-100	6,412,335	602,907	0.036	0.004
2.5-opt-ACs	6,428,845	602,878	1.020	0.220
1-shift	6,683,735	628,833	4.112	0.937
2-p-opt	6,491,451	604,388	4.093	0.954

Notes. 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.

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-Based Approach

In this section, we compare 2.5-opt-EEs-100 with 2.5-opt-ACs. The two algorithms differ only in the delta evaluation procedure they adopt: empirical estimation versus analytical computation. The experimental results are illustrated using runtime development plots and are shown in Figure 9.

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

In Table 3, we report the observed relative difference between the cost of the local optima obtained by the two algorithms and a 95% confidence interval of the relative difference (obtained from interval estimation through a t-test). For the sake of completeness, we also present these data for what concerns the comparison of 2.5-opt-EEs-100 with 1-shift and 2-p-opt.

Table 3 confirms that, concerning the average cost of the local optima found, 2.5-opt-EEs-100 is either slightly worse (for $p = 0.1$) or essentially equivalent to 2.5-opt-ACs (for $p > 0.1$). To be more precise, for $p = 0.1$, 2.5-opt-EEs-100 has obtained solutions, the average cost of which is higher than the one of those obtained by 2.5-opt-ACs. The difference is significant in a statistical sense, but it is nonetheless relatively small: With a confidence of 95%, we can state that the expectation of the costs of the solutions obtained by 2.5-opt-EEs-100, on the class of instances under analysis, is at most 1.98% higher than the one of 2.5-opt-ACs.

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

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

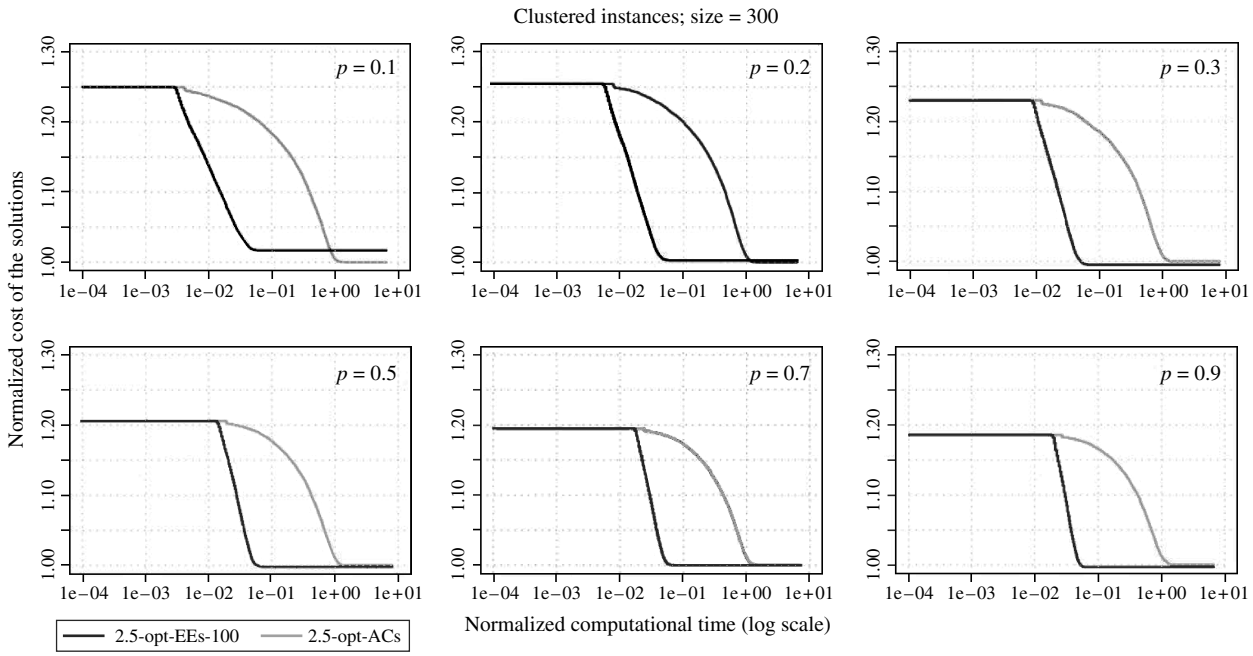

Figure 9 Experimental Results on Clustered Homogeneous PTSP Instances of Size 300

Notes. 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.

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

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

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

Table 3 Comparison of the Average Cost Obtained by `2.5-opt-EEs-100` and by `2.5-opt-ACs`, `1-shift`, and `2-p-opt`, on Clustered Instances of Size 300

	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	95% CI	Difference	95% CI	Difference	95% CI
0.1	+**1.71**	[+1.438, +1.98]	+**1.42**	[+1.00, +1.84]	−**3.25**	[−3.60, −2.90]
0.2	+0.28	[−0.054, +0.61]	−0.32	[−0.84, +0.19]	−**4.76**	[−5.22, −4.31]
0.3	−0.45	[−0.938, +0.03]	−**1.09**	[−1.65, −0.53]	−**4.28**	[−4.71, −3.86]
0.5	−0.20	[−0.672, +0.28]	−**2.74**	[−3.25, −2.22]	−**3.02**	[−3.50, −2.52]
0.7	−0.02	[−0.367, +0.31]	−**3.50**	[−4.05, −2.98]	−**1.99**	[−2.43, −1.58]
0.9	−0.26	[−0.660, +0.05]	−**4.06**	[−4.58, −3.63]	−**1.22**	[−1.65, −0.88]

Notes. 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 interval (CI) obtained through the *t*-test on the relative difference. 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 the *t*-test, at a confidence of 95%.

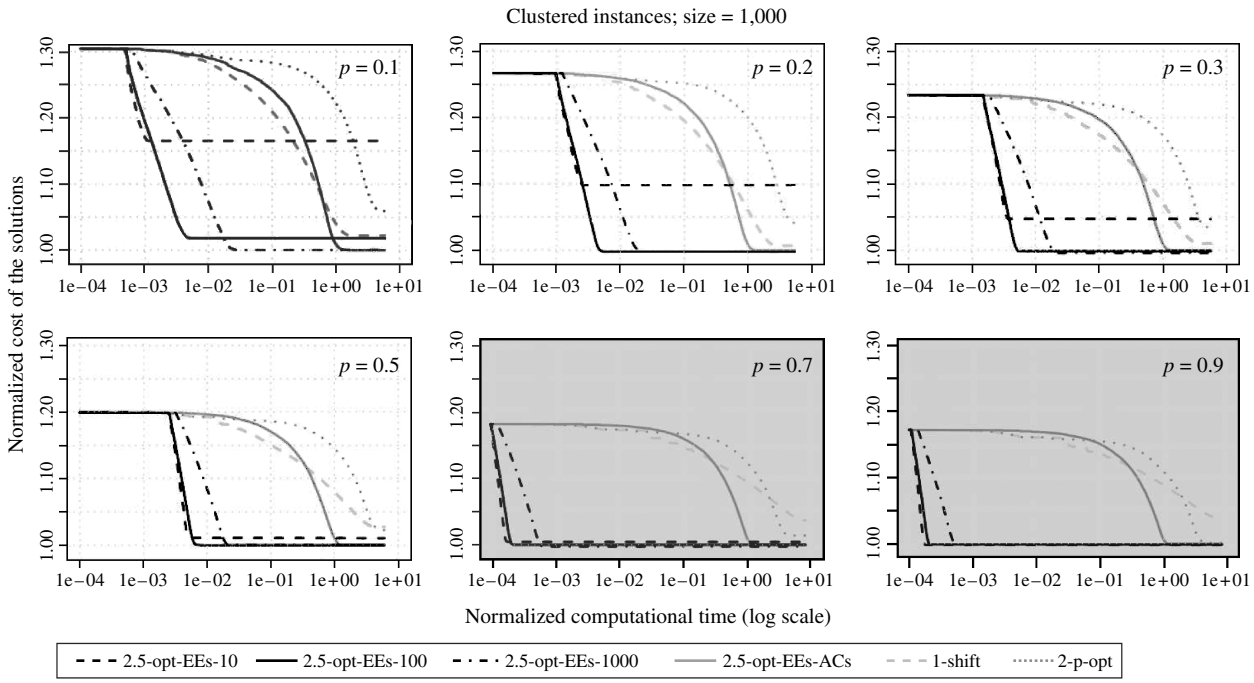

Figure 10 Experimental Results on Clustered Homogeneous PTSP Instances of Size 1,000

Notes. 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 for $p > 0.5$ the algorithms based on the analytical computation techniques use a library for arbitrary precision arithmetics (denoted by shaded graphs).

5.3. Experiments on Large Instances

In this section, we study the performance of `2.5-opt-EEs-100` when applied to large instances. For this purpose, we considered PTSP instances with 1,000 nodes. In these experiments, for $p > 0.5$, `2.5-opt-ACs`, `1-shift`, and `2-p-opt` use MPFR (Fousse et al. 2007), a state-of-the-art library for arbitrary precision arithmetics to handle the numerical issue discussed in §3.3.

In the very same setting, we also study the impact on the performance of `2.5-opt-EEs` of the number of realizations considered. For this purpose, we consider samples of size 10, 100, and 1,000 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 10 and Table 4.

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

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

Concerning the impact of the sample size on the performance of `2.5-opt-EEs`, we can observe that the use of a large number of realizations, in our case $M = 1{,}000$, is indeed very effective with respect to the cost of the local optima for low probability values. Even though this improvement is achieved at the expense of computation time, the total search time is relatively short when compared to analytical computation algorithms. On the other hand, the use of few realizations, in our case $M = 10$, is less effective and does not significantly reduce the computation time. Concerning the average computation time, `2.5-opt-EEs-10` is faster than `2.5-opt-EEs-100` by a factor of approximately two, whereas `2.5-opt-EEs-1000` is slower than `2.5-opt-EEs-100` by a factor of four. Nonetheless, an important observation is that for $p \leq 0.5$, `2.5-opt-EEs-1000` is approximately 1.5 orders

Table 4 **Experimental Results for** 2.5-opt-EEs-10, 2.5-opt-EEs-100, 2.5-opt-EEs-1000, 2.5-opt-ACs, 2-p-opt, **and** 1-shift **on Clustered Instances of Size 1,000**

	Solution cost		Computation time	
Algorithm	Mean	s.d.	Mean	s.d.
$p = 0.1$				
2.5-opt-EEs-10	5,909,938	461,029	0.462	0.030
2.5-opt-EEs-100	5,158,215	448,385	1.839	0.190
2.5-opt-EEs-1000	5,069,560	424,448	9.487	1.170
2.5-opt-ACs	5,068,223	450,709	443.952	70.934
1-shift	5,178,144	469,977	635.757	84.010
2-p-opt	5,365,486	449,318	1,464.535	341.993
$p = 0.2$				
2.5-opt-EEs-10	7,364,518	513,937	0.524	0.032
2.5-opt-EEs-100	6,692,459	486,598	1.024	0.092
2.5-opt-EEs-1000	6,681,179	475,423	3.981	0.447
2.5-opt-ACs	6,697,814	480,609	229.288	33.165
1-shift	6,744,906	494,658	547.263	71.878
2-p-opt	6,978,843	477,590	859.276	159.102
$p = 0.3$				
2.5-opt-EEs-10	8,263,425	554,699	0.507	0.030
2.5-opt-EEs-100	7,894,854	547,385	0.722	0.051
2.5-opt-EEs-1000	7,875,735	511,413	2.658	0.306
2.5-opt-ACs	7,901,717	524,412	149.881	22.702
1-shift	7,982,498	531,787	451.773	58.575
2-p-opt	8,175,022	547,812	552.554	95.447
$p = 0.5$				
2.5-opt-EEs-10	9,693,061	630,069	0.422	0.020
2.5-opt-EEs-100	9,592,605	623,310	0.526	0.038
2.5-opt-EEs-1000	9,591,076	635,788	1.689	0.149
2.5-opt-ACs	9,597,432	599,270	89.272	14.155
1-shift	9,856,073	579,796	316.049	44.883
2-p-opt	9,799,426	594,452	338.203	63.679
$p = 0.7$				
2.5-opt-EEs-10	10,852,736	686,839	0.371	0.017
2.5-opt-EEs-100	10,803,761	623,940	0.448	0.025
2.5-opt-EEs-1000	10,776,397	677,248	1.281	0.112
2.5-opt-ACs	**10,805,384**	**669,183**	**2,377.355**	**296.572**
1-shift	**11,203,988**	**666,372**	**14,909.760**	**1,911.958**
2-p-opt	**10,955,608**	**634,087**	**8,012.715**	**1,339.187**
$p = 0.9$				
2.5-opt-EEs-10	11,752,749	701,937	0.347	0.012
2.5-opt-EEs-100	11,764,968	707,582	0.407	0.018
2.5-opt-EEs-1000	11,763,689	716,438	1.015	0.073
2.5-opt-ACs	**11,777,782**	**716,614**	**2,062.499**	**192.265**
1-shift	**12,241,504**	**701,184**	**12,351.744**	**1,610.433**
2-p-opt	**11,792,577**	**684,387**	**7,019.027**	**993.119**

Notes. Each algorithm is allowed to run until it reaches a local optimum. The table gives mean and standard deviation (s.d.) of final solution cost and computation time in seconds. The results are given for 100 instances at each probability level. Note that for $p > 0.5$, the algorithms based on the analytical computation techniques use a library for arbitrary precision arithmetics (denoted by boldface).

of magnitude faster than 2.5-opt-ACs. For $p > 0.5$, the adoption of the arbitrary precision arithmetics entails a major computational overhead: the former is approximately three orders of magnitude faster than the latter. Concerning the average cost of local optima, 2.5-opt-EEs-10 is worse than the algorithms that use 100 and 1,000 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.

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 of using the same set of M realizations for all steps of the iterative improvement algorithm: (i) a set of M realizations is sampled anew each time an improved solution is found, and (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 variability with respect to the mean of the cost difference estimator. To avoid this problem, we implemented a mechanism that for $p = 0.1$ memorizes moves to reject them in successive search steps. The results on clustered instances with 300 nodes are given in Figure 11.

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

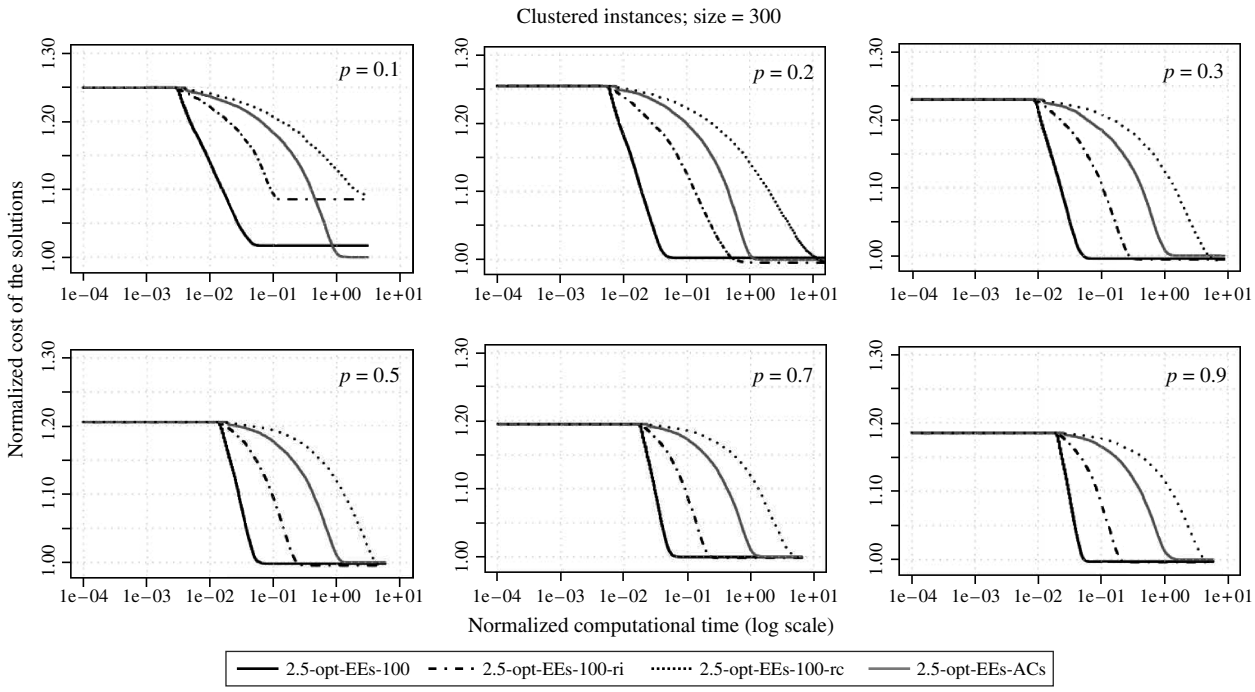

Figure 11 Experimental Results on Clustered Homogeneous PTSP Instances of Size 300

Notes. 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.

optima. Concerning the perturbation, which generates new starting solutions for the subsequent local search by introducing some changes in the incumbent local optima, we adopted a random double-bridge move (Martin et al. 1991, Johnson and McGeoch 1997).

We denote the two ILS algorithms `ILS-2.5-opt-EEs` and `ILS-2.5-opt-ACs`, respectively. For `ILS-2.5-opt-EEs`, we use 100 and 1,000 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 criterion 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 1,000 nodes are given in Figure 12 and Table 5.

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 $p \leq 0.2$. This is because 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`. Because the cost difference estimator is accurate enough, the use of 100 realizations instead of 1,000 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 t-test, with a confidence of 95%.

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. 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 the proposed approach is the use of 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 a closed form. Moreover, we have shown that TSP-specific neighborhood reduction techniques are also effective for the PTSP. Furthermore,

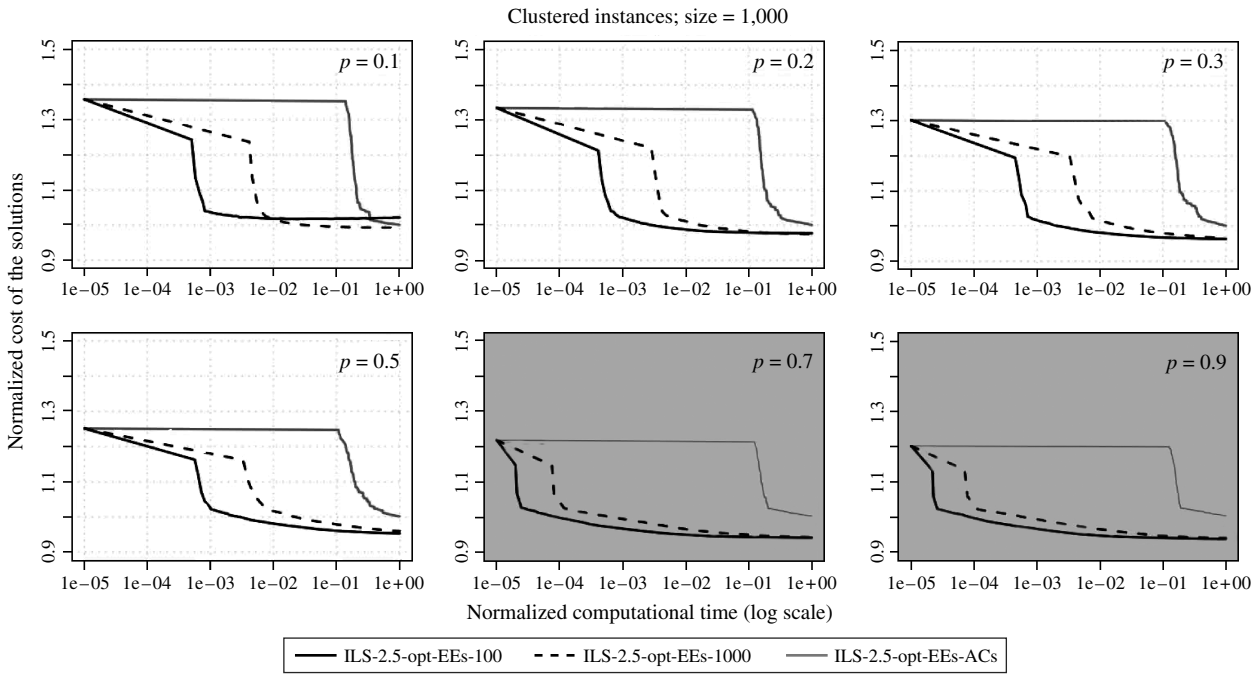

Figure 12 Experimental Results on Clustered Homogeneous PTSP Instances of Size 1,000

Notes. 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 for $p > 0.5$, `ILS-2.5-opt-ACs` uses a library for arbitrary precision arithmetics (denoted by shaded graphs).

Table 5 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 1,000

	Solution cost		Computation time	
Algorithm	Mean	s.d.	Mean	s.d.
$p = 0.1$				
`ILS-2.5-opt-ACs`	4,902,016	422,874	3,242.608	356.583
`ILS-2.5-opt-EEs-100`	5,008,657	422,307		
`ILS-2.5-opt-EEs-1000`	4,867,455	421,312		
$p = 0.2$				
`ILS-2.5-opt-ACs`	6,396,843	495,856	1,938.307	185.242
`ILS-2.5-opt-EEs-100`	6,251,947	463,578		
`ILS-2.5-opt-EEs-1000`	6,231,327	467,713		
$p = 0.3$				
`ILS-2.5-opt-ACs`	7,526,267	528,724	1,294.857	155.803
`ILS-2.5-opt-EEs-100`	7,252,326	499,856		
`ILS-2.5-opt-EEs-1000`	7,270,660	500,364		
$p = 0.5$				
`ILS-2.5-opt-ACs`	9,216,158	602,968	844.968	110.294
`ILS-2.5-opt-EEs-100`	8,785,082	559,811		
`ILS-2.5-opt-EEs-1000`	8,841,611	555,746		
$p = 0.7$				
`ILS-2.5-opt-ACs`	**10,494,415**	**665,735**	**15,783.765**	**3,814.845**
`ILS-2.5-opt-EEs-100`	9,881,134	600,153		
`ILS-2.5-opt-EEs-1000`	9,897,012	599,627		
$p = 0.9$				
`ILS-2.5-opt-ACs`	**11,480,461**	**738,214**	**13,191.050**	**4,044.378**
`ILS-2.5-opt-EEs-100`	10,763,395	643,185		
`ILS-2.5-opt-EEs-1000`	10,788,126	646,192		

Notes. The table gives mean and standard deviation (s.d.) of final solution cost and computation time in seconds. The results are given for 100 instances at each probability level. Note that for $p > 0.5$, `ILS-2.5-opt-ACs` uses a library for arbitrary precision arithmetics (denoted by boldface).

we identified a practical issue in applying the current state-of-the-art iterative improvement algorithms. This issue has never been reported in the literature and it has to be addressed explicitly by resorting to arbitrary precision arthimetics. This, as shown in §5.3, has a major impact on computation time.

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, 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 §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. Moreover, other stochastic combinatorial optimizations techniques such as the stochastic ruler method, the nested partitions method, and stochastic branch and bound will be considered for an empirical comparison with the presented approach.

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 Training Site funded by the Commission of the European Community, and by the ANTS project, an *Action de Recherche Concertée* funded by the French Community of Belgium. The authors acknowledge support from the fund for scientific research F.R.S.-FNRS of the French Community of Belgium.

References

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

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

Bertsimas, D., P. Jaillet, A. Odoni. 1990. A priori optimization. *Oper. Res.* **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., A. Campbell. 2007. Extension of the 2-p-opt and 1-shift algorithms to the heterogeneous probabilistic traveling salesman problem. *Eur. J. Oper. Res.* **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. *Eur. J. Oper. Res.* **162** 206–219.

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. *J. Math. Model. Algorithms* **5** 91–110.

Birattari, M., P. Balaprakash, T. Stützle, M. Dorigo. 2007. Extended empirical analysis of estimation-based local search for stochastic combinatorial optimization. IRIDIA Supplementary page, 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.

Fousse, L., G. Hanrot, V. Lefèvre, P. Pélissier, P. Zimmermann. 2007. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Software* **33** 1–15.

Fu, M. C. 1994. Optimization via simulation: A review. *Ann. Oper. Res.* **53** 199–248.

Fu, M. C. 2002. Optimization for simulation: Theory vs. practice. *INFORMS J. Comput.* **14** 192–215.

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

Gutjahr, W. J. 2003. A converging ACO algorithm for stochastic combinatorial optimization. A. Albrecht, K. Steinhofl, eds. *Stochastic Algorithms: Foundations and Applications, Lecture Notes in Computer Science*, Vol. 2827. Springer-Verlag, Berlin, 10–25.

Gutjahr, W. J. 2004. S-ACO: An ant based approach to combinatorial optimization under uncertainty. 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, Lecture Notes in Computer Science*, Vol. 3172. Springer-Verlag, Berlin, 238–249.

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

IEEE. 1985. IEEE standard for binary floating-point arithmetic. Institute of Electrical and Electronics Engineers, New York.

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

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, UK, 215–310.

Johnson, D. S., L. A. McGeoch, C. Rego, F. Glover. 2001. 8th DIMACS implementation challenge. Retrieved January 2007, http://www.research.att.com/~dsj/chtsp/.

Kleywegt, A. J., A. Shapiro, T. Homem de Mello. 2002. The sample average approximation method for stochastic discrete optimization. *SIAM J. Optim.* **12** 479–502.

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. Kluwer 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 J. Comput.* **6** 68–81.

Pichitlamken, J., B. L. Nelson. 2003. A combined procedure for optimization via simulation. *ACM Trans. Model. Comput. Simulation* **13** 155–179.

Verweij, B., S. Ahmed, A. J. Kleywegt, G. Nemhauser, A. Shapiro. 2003. The sample average approximation method applied to stochastic routing problems: A computational study. *Comput. Optim. Appl.* **24** 289–333.