4

Engineering Stochastic Local Search Algorithms: A Case Study in Estimation-Based Local Search for the Probabilistic Travelling Salesman Problem

Prasanna Balaprakash, Mauro Birattari, and Thomas Stützle

IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
{pbalapra,mbiro,stuetzle}@ulb.ac.be

Summary. In this article, we describe the steps that have been followed in the development of a high performing stochastic local search algorithm for the probabilistic travelling salesman problem, a paradigmatic combinatorial stochastic optimization problem. In fact, we have followed a bottom-up algorithm engineering process that starts from basic algorithms (here, iterative improvement) and adds complexity step-by-step. An extensive experimental campaign has given insight into the advantages and disadvantages of the prototype algorithms obtained at the various steps and directed the further algorithm development. The final stochastic local search algorithm was shown to substantially outperform the previous best algorithms known for this problem. Besides the systematic engineering process for the development of stochastic local search algorithms followed here, the main reason for the high performance of our final algorithm is the innovative adoption of techniques for the estimation of the cost of neighboring solutions using delta evaluation.

Keywords: Stochastic Local Search, Algorithm Engineering, Stochastic Optimization Problems, Probabilistic Travelling Salesman Problem, Estimation-Based Local Search.

4.1 Introduction

Stochastic local search (SLS) algorithms are powerful tools for tackling computationally hard decision and optimization problems that arise in many application areas. The field of SLS is rather vast and there exists a large variety of algorithmic techniques and strategies. They range from simple constructive and iterative improvement algorithms to general purpose SLS methods such as iterated local search, tabu search, and ant colony optimization.

A frequently used approach for the development of SLS algorithms appears to be that the algorithm designer takes his favorite general-purpose SLS method. Then, this method is adapted in a sometimes more, often less organized way to the problem under consideration, guided by the researcher's intuitions and previous experiences.

There is an increasing awareness that the development of SLS algorithms should be done in a more structured way, ideally in an engineering style

C. Cotta and J. van Hemert (Eds.): Recent Advances in Evol. Comp., SCI 153, pp. 53–66, 2008.
springerlink.com

following a well-defined process and applying a set of well established procedures. One possibility is to follow a bottom-up engineering process that starts from basic algorithms and adds complexity step-by-step. Such a process could be organized as follows. First, start by analyzing the problem under concern and gaining insights into its structure and possibly previous work on it. Second, develop basic constructive and iterative improvement algorithms and analyze them experimentally. Third, integrate these basic heuristics into simple SLS methods to improve upon their performance. Fourth, if deemed necessary, add more advanced concepts to extend the SLS algorithm by, for example, consider populations. Clearly, such an approach is to be considered an iterative one where insights gained through experiments at one level may lead to further refinements at the same or at previous levels.

In this chapter, we illustrate the steps that we have followed in the development of new state-of-the-art algorithms for the PROBABILISTIC TRAVELLING SALESMAN PROBLEM (PTSP). In fact, we were following a bottom-up approach, which in this case was mainly focused on the implementation and refinement of a very effective iterative improvement algorithm. As it will be shown later, this process was supported by a comprehensive experimental analysis, the usage of the automatic tuning of some algorithm parameters, an efficient implementation of supporting data structures, and an integration of the iterative improvement algorithm into an iterated local search.

The high-level steps of the adopted bottom-up approach materialize in the following main elements. The main underlying ideas contributing to the success of this research were (i) the inclusion of speed-up techniques known from the deterministic TSP to the PTSP local search algorithms, and (ii) the use of empirical estimation in the evaluation of local search moves. These ideas have been implemented into a new estimation-based iterative improvement algorithm for the PTSP. Experimental results with the final iterated local search algorithm show that we actually have obtained a new state-of-the-art algorithm that outperforms the previous best algorithm for this problem.

4.2 The Probabilistic Travelling Salesman Problem

The PTSP [70] is a paradigmatic example of routing problems under uncertainty. It is similar to the TSP with the only difference that each node has a probability of requiring being visited. The *a priori* optimization approach [71] for the PTSP consists in finding an *a priori* solution that visits all the nodes and that minimizes the expected cost of *a posteriori* solutions. The *a priori* solution must be found prior to knowing which nodes are to be visited. The associated *a posteriori* solution is computed *after* knowing which nodes need to be visited. It is obtained by skipping the nodes that do not require being visited and visiting the others in the order in which they appear in the *a priori* solution. An illustration is given in Figure 4.1. Formally, a PTSP instance 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

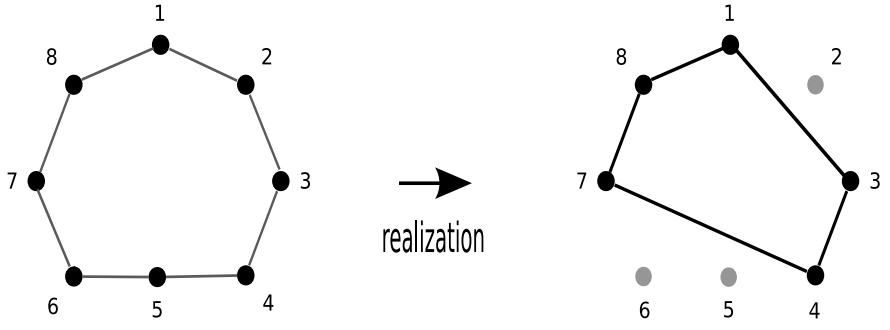


Fig. 4.1. An *a priori* solution for a PTSP instance with 8 nodes. The nodes in the *a priori* solution are visited in the order 1, 2, 3, 4, 5, 6, 7, 8, and 1. Assume that a realization of $\boldsymbol{\omega}$ prescribes that nodes 1, 3, 4, 7, and 8 are to be visited. The *a posteriori* solution visits then nodes in the order in which they appear in the *a priori* solution and skips nodes 2, 5, and 6 since they do not require being visited.

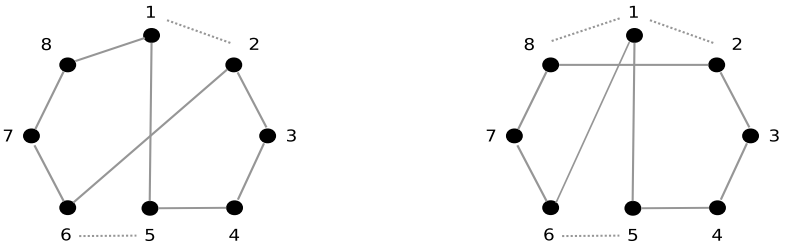
nodes, $C = \{c_{ij} : \langle i, j\rangle \in A\}$ is a set of costs associated with edges, and $P = \{p_i : i \in V\}$ is a set of probabilities that for each node i specifies its probability p_i of requiring being visited. Hence, for the PTSP the stochastic element of the problem is defined by a random variable $\boldsymbol{\omega}$ that is distributed according to 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 being visited and a '0' indicates that it does not. We assume that the costs are symmetric. 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 the given n-variate Bernoulli distribution.

4.3 Iterative Improvement Algorithms for the PTSP

Iterative improvement algorithms start from some initial solution and repeatedly try to move from a current solution x to a lower cost neighboring one. An iterative improvement search terminates in a local optimum, that is, a solution that does not have any improving neighbor. In the PTSP literature, mainly *2-exchange* and *node-insertion* neighbourhood structures were considered (see Figure 4.2 for examples).

Crucial to the performance of many iterative improvement algorithms is the possibility of performing *delta evaluation*, that is, of computing the cost of a neighbor by only considering the cost contribution of the solution components in which the two solutions differ. For example, in the case of a *2-exchange* move that deletes edges $\langle a, b\rangle$ and $\langle c, d\rangle$ and adds edges $\langle a, c\rangle$ and $\langle b, d\rangle$, the cost difference is given by $c_{a,c} + c_{b,d} - c_{a,b} - c_{c,d}$.

`2-p-opt` and `1-shift`, the current state-of-the-art iterative improvement algorithms for the PTSP, use analytical computations, that is, closed-form expressions based on heavy mathematical derivations, for correctly taking into account



(a) (b)

Fig. 4.2. 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.

the random variable $\boldsymbol{\omega}$ in such a *delta evaluation* [72, 73, 74]. Unfortunately, to allow the re-use of already computed expressions and, thus, to make the *delta evaluation* more efficient, `2-p-opt` and `1-shift` require to scan the neighbourhood in a fixed lexicographic order.

4.4 Engineering an Iterative Improvement Algorithm for the PTSP

We now present the main steps that we followed for engineering an iterative improvement algorithm for the PTSP. It is based on two main hypotheses.

Hypothesis 1: Exploiting known TSP speed-up techniques can increase computation speed.

Hypothesis 2: We can use estimations of the costs for the *delta evaluation* to make the local search faster.

4.4.1 Introducing TSP Speed-Up Techniques

Iterative improvement algorithms for the TSP strongly exploit neighbourhood reduction techniques such as fixed radius search, candidate lists, and don't look bits [75,76]. These techniques allow the local search to focus on the most relevant part for obtaining improvements by pruning a large part of the neighbourhood. While reducing very strongly the number of neighbors, the exploitation of these techniques does not allow to scan the neighbourhood in the lexicographic order, which is required to use the speed-ups of the *delta evaluation* as it is used in `2-p-opt` and `1-shift`.

We implemented an iterative improvement algorithm using the three above mentioned TSP speed-up techniques. For doing so, we have to compute the cost difference between two neighboring solutions from scratch. This is done by using

the closed-form expressions proposed for the *2-exchange* and the *node-insertion* neighbourhood structures [74]. In fact, we implemented an iterative improvement algorithm based on the *2.5-exchange neighbourhood* [77]. The *2.5-exchange neighbourhood* is well-known in TSP solving and it combines the *node-insertion neighbourhood* and the *2-exchange neighbourhood* structure into a hybrid one. We call the resulting algorithm `2.5-opt-ACs`, where `AC` and `s` stand for *analytical computation* and *speedup*, respectively.

`2.5-opt-ACs` was experimentally compared to `2-p-opt` and `1-shift`. Our analysis is based on PTSP instances that we obtained from TSP instances generated with the DIMACS instance generator [78]. They are homogeneous PTSP instances, where all nodes of an instance have a same probability p of appearing in a realization. We carried out experiments on clustered instances of 300 nodes, where the nodes are arranged in a number of clusters, in a $10^6 \times 10^6$ square. We considered the following probability levels: $[0.1, 0.9]$ with a step size of 0.1. All algorithms were implemented in `C` and the source codes were compiled with `gcc`, version 3.3. Experiments were carried out on AMD Opteron$^{\text{TM}}$244 1.75 GHz processors with 1 MB L2-Cache and 2 GB RAM, running under the Rocks Cluster GNU/Linux. Each iterative improvement algorithm is run until it reaches a local optimum. (In the following we present only some of the most important results; more details are given in [79].)

The results given in Figure 4.3, which illustrate the development of the average cost obtained, show that `2.5-opt-ACs` *dominates* `2-p-opt` and `1-shift`. Irrespective of the probability value, `2.5-opt-ACs` reaches local optima about four times faster than `2-p-opt`. Compared to `1-shift`, the same holds when $p \geq 0.5$; for small p, however, the speed difference 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`. The same trend is true when compared to `1-shift`, except for $p \leq 0.3$, where the difference between `2.5-opt-ACs` and `1-shift` is small. All the observed cost differences are statistically significant, as shown by the t-test; an exception is for $p \leq 0.2$, where the difference between `2.5-opt-ACs` and `1-shift` is not significant.

4.4.2 Estimation-Based Local Search

Empirical estimation is a technique for estimating the expectation through Monte Carlo simulation. Empirical estimation can also be applied to estimate the cost for the *delta evaluation*. This has the advantage that one can use any neighbourhood structure without requiring complex mathematical derivations. In particular, the *2.5-exchange neighbourhood* can easily be used. Clearly, an estimation-based *delta evaluation* procedure also does not impose constraints on the order of exploring the neighbourhood. Thus, it is easy to integrate the TSP neighbourhood reduction techniques. Given these advantages, our hypothesis is that the estimation-based *delta evaluation* procedure can lead to a very fast and highly effective iterative improvement algorithms.

In empirical estimation, an unbiased estimator of $F(x)$ for a solution x can be computed using M independent realizations of the random variable $\boldsymbol{\omega}$ and

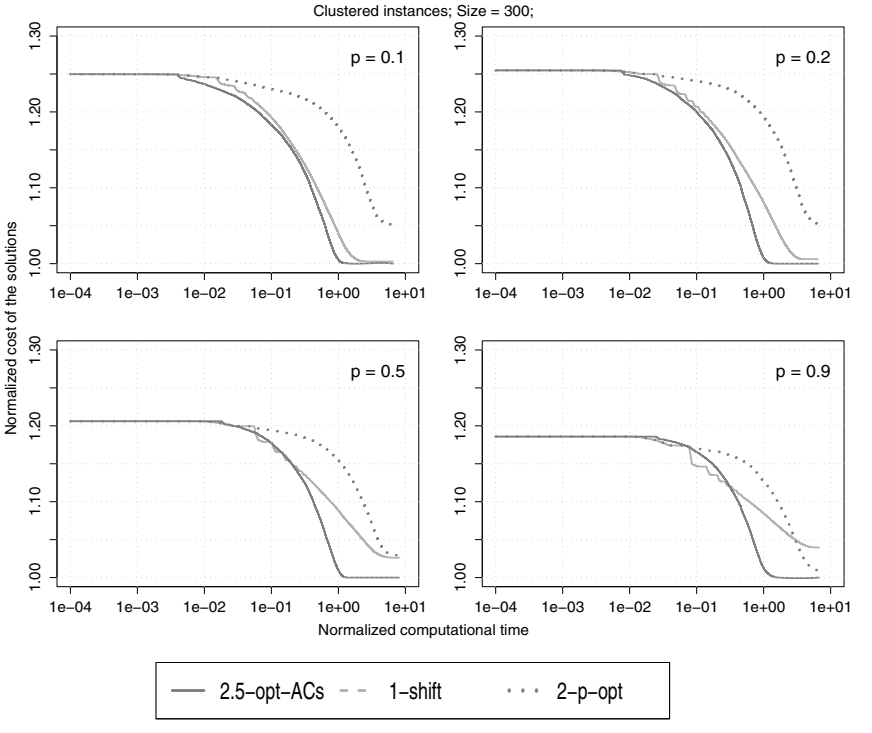


Fig. 4.3. Experimental results on clustered homogeneous PTSP instances of size 300. The plots represent the development of the average cost of the solutions obtained by `2-p-opt` and `1-shift` normalized by the one obtained by `2.5-opt-ACs`. Each algorithm is stopped when it reaches a local optimum.

the resulting costs of these *a posteriori* solutions [80, 79]. The cost difference between a solution x and a neighboring solution x' can then be estimated by $\hat{F}_M(x') - \hat{F}_M(x)$, which is given by:

$$\hat{F}_M(x') - \hat{F}_M(x) = \frac{1}{M} \sum_{r=1}^{M} \Big(f(x', \omega_r) - f(x, \omega_r) \Big). \tag{4.1}$$

We use the same M realizations for all steps of the iterative improvement algorithms. Alternatively, one may sample for each comparison M new realizations; however, this was proven to be not effective in our experiments (for details see [79]).

For estimating the cost differences between two neighboring *a priori* solutions by a realization ω, one needs to identify the edges that are not common in the two *a posteriori* solutions. This is done as follows. For every edge $\langle i, j \rangle$ that is deleted from x, one needs to find the corresponding edge $\langle i^*, j^* \rangle$ that is deleted in the *a posteriori* solution of x. This so-called *a posteriori edge* is obtained

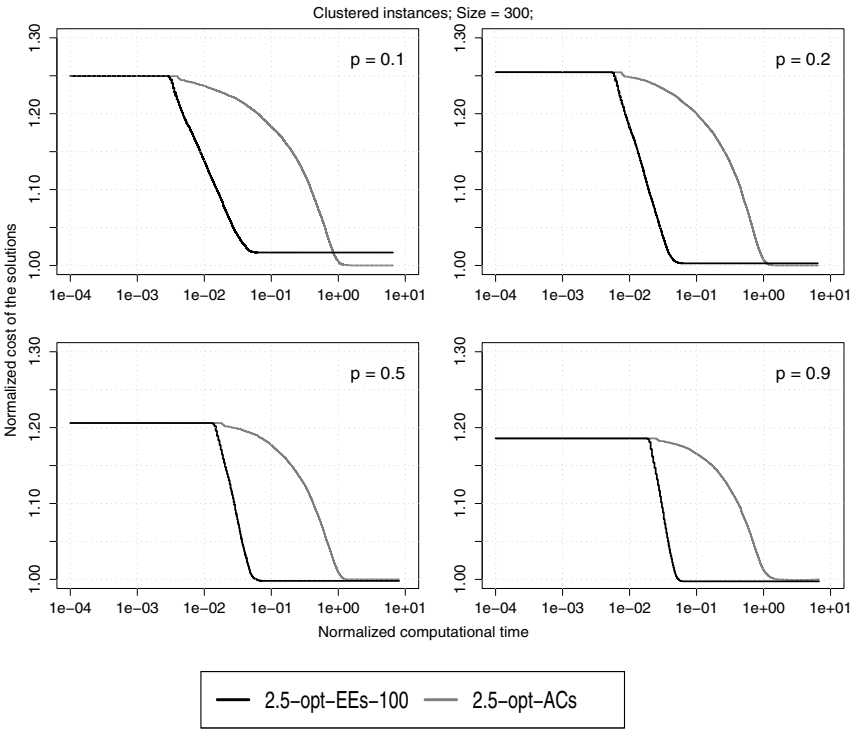


Fig. 4.4. Experimental results on clustered homogeneous PTSP instances of size 300. The plots represent the average solution cost obtained by `2.5-opt-EEs-100` normalized by the one obtained by `2.5-opt-ACs`. Each algorithm is stopped upon reaching a local optimum.

as follows. If node i requires being visited, we have $i^* = i$; otherwise, i^* is the first predecessor of i in x such that $\omega[i^*] = 1$, that is, the first predecessor that requires being visited. If node j requires being visited, then $j^* = j$; otherwise, j^* is the first successor of j such that $\omega[j^*] = 1$. In a *2-exchange* move that deletes the edges $\langle a, b\rangle$ and $\langle c, d\rangle$ from x and replaces them by $\langle a, c\rangle$ and $\langle b, d\rangle$, hence, first the corresponding *a posteriori edges* $\langle a^*, b^*\rangle$ and $\langle c^*, d^*\rangle$ for a given realization ω are to be identified. The cost difference between the two *a posteriori* solutions is then given by $c_{a^*,c^*} + c_{b^*,d^*} - c_{a^*,b^*} - c_{c^*,d^*}$. and Eq. 4.1 then simply sums the cost differences for each of the M realizations. This procedure can be directly extended to *node-insertion* moves. Furthermore, the algorithm adopts the neighbourhood reduction techniques *fixed-radius search*, *candidate lists*, and *don't look bits* [81, 77, 75]. We call the resulting first-improvement algorithm `2.5-opt-EEs`. (See [79] for more details).

As a default, we use 100 realizations in `2.5-opt-EEs`, which is indicated by denoting this version as `2.5-opt-EEs-100`. Next, we compare `2.5-opt-EEs-100` with `2.5-opt-ACs`; these two algorithms differ only in the *delta evaluation*

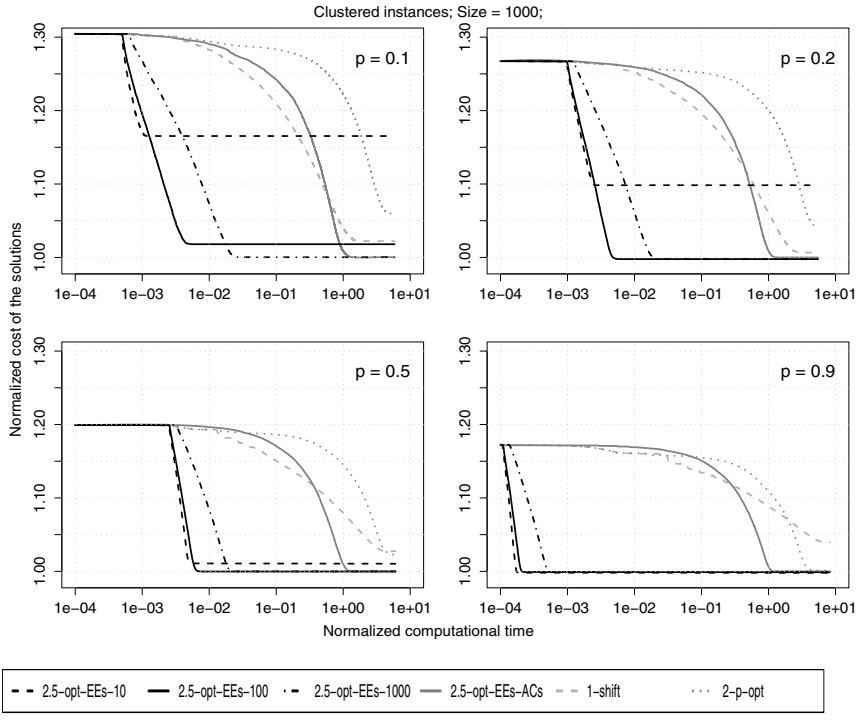


Fig. 4.5. Experimental results on clustered homogeneous PTSP instances of size 1000. The plots give the solution cost obtained by `2.5-opt-EEs-10`, `2.5-opt-EEs--100`, `2.5-opt-EEs-1000`, `1-shift`, and `2-p-opt` normalized by the one obtained by `2.5-opt-ACs`. Each algorithm stops upon reaching a local optimum.

procedure: *empirical estimation* versus *analytical computation*. Figure 4.4 gives the experimental results. Both algorithms reach similar average costs with the only exception of $p = 0.1$, where `2.5-opt-EEs-100` returns local optima with an average cost that is approximately 2% higher than that of `2.5-opt-ACs`. However, `2.5-opt-EEs-100` is much faster than `2.5-opt-ACs`; it reaches local optima, irrespective of the probability value, approximately 1.5 orders of magnitude faster.

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. Since the variance of the cost difference estimator is very high at low probability levels, the adoption of 100 realizations is not sufficient to obtain a good estimate of the cost difference. We therefore added experiments to examine 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`. We considered PTSP instances with 1000 nodes, which are generated in the same way as described before, to study the performance of the

algorithms when applied to large instances. (For a PTSP instance size of 1000, `2.5-opt-ACs`, `2-p-opt`, and `1-shift` suffer from numerical problems and they need to resort to an arbitrary precision arithmetic for $p > 0.5$ [79], which makes them even slower.) The results are given in Figure 4.5.

As conjectured, the use of a large number of realizations, in our case $M = 1000$, is effective with respect to the cost of the local optima for low probability values. Even though larger sample sizes incur more computation time, the total search time is very short compared to the *analytical computation* algorithms. On the other hand, the use of few realizations, in our case $M = 10$, does not lead to a further very significant reduction of 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 solution cost, `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 comparable with the one of `2.5-opt-ACs`. `2.5-opt-EEs-10` reaches clearly much worse costs than the other estimation-based algorithms; only for high probability values it appears to be competitive.

With respect to the instance size, the trends concerning the relative performance of `2.5-opt-EEs-100` and `2.5-opt-ACs` are similar as for instances of size 300. However, the differences between the computation times of the two algorithms are larger and `2.5-opt-EEs-100` reaches, irrespective of the value of p, local optima approximately 2.3 orders of magnitude faster than `2.5-opt-ACs`. Similarly, for the comparison between `2.5-opt-ACs` and `1-shift` and `2-p-opt`, respectively, the results for instance size 1000 are analogous to those for instance size 300.

4.4.3 Improving the Estimation-Based Local Search

The results of the previous section clearly show that the performance of `2.5-opt-EEs` depends on the number of realizations and the probabilities associated with the nodes. In particular, for low probability values, `2.5-opt-EEs` is less effective with few realizations. This insight led to our third hypothesis:

Hypothesis 3: `2.5-opt-EEs` can be improved by choosing an appropriate sample size and a special treatment of the low probability cases.

Two main ideas were developed to improve `2.5-opt-EEs`. The first is to use an *adaptive sampling procedure* that selects the appropriate number of realizations with respect to the variance of the cost estimator. In fact, as shown in [82], the variance of the cost of the *a posteriori* solutions depends strongly on the probabilities associated with the nodes: the smaller the probability values, the higher the variance. The increased variance could be handled by increasing the sample size since averaging over a large number of realizations reduces the variance of

the estimator. However, for high probability values using a large number of realizations is a waste of time. For addressing this issue, we adopted an adaptive sampling procedure that makes use of *Student's t-test* in the following way: Given two neighboring *a priori* solutions, the cost difference between their corresponding *a posteriori* solutions is sequentially computed on a number of realizations. Once the *t-test* rejects the null hypothesis of zero cost difference, the computation is stopped; if the null hypothesis cannot be rejected, the computation is stopped after a maximum of M realizations, where M is a parameter. The sign of the estimator determines the solution of lower cost. The *estimation-based* iterative improvement algorithm that adds adaptive sampling to `2.5-opt-EEs` will be called `2.5-opt-EEas`.

The second main idea is to adopt the *importance sampling* technique for reducing the variance of the estimator when dealing with highly stochastic problem instances: given two neighboring *a priori* solutions, this technique considers, instead of realizations from the given distribution $\boldsymbol{\omega}$, realizations from another distribution $\boldsymbol{\omega}^*$—the so-called biased distribution. $\boldsymbol{\omega}^*$ forces the nodes involved in the cost difference computation to occur more frequently. The resulting cost difference between two *a posteriori* solutions for each realization is corrected for the adoption of the biased distribution and the correction is given by the likelihood ratio of the original distribution with respect to the biased distribution. We denote `2.5-opt-EEais` the algorithm that adds to `2.5-opt-EEas` the above described importance sampling procedure. Note that the adoption of the importance sampling technique in `2.5-opt-EEas` requires additional parameters for defining the biased distribution. These parameters have been tuned by `Iterative F-Race` [83]. We refer the reader to [82] for a more detailed description of `2.5-opt-EEais` and its tuning.

Next, we compared the performance of `2.5-opt-EEas` and `2.5-opt-EEais` to `2.5-opt-EEs`, which does not use an adaptive sample size and importance sampling. In the case of `2.5-opt-EEs`, we again consider samples of size 10, 100, and 1000. The clustered instances of 1000 nodes are used for the experiments with the probability levels $[0.050, 0.200]$ with a step size of 0.025 and $[0.3, 0.9]$ with a step size of 0.1. Results of the comparison are given in Figure 4.6, where `2.5-opt-EEs-1000` is taken as a reference. (We only highlight the most important results; more details are given in [82].)

The computational results show that, especially for low probabilities, `2.5-opt-EEais` is more effective than the other algorithms. For what concerns the comparison of `2.5-opt-EEais` and `2.5-opt-EEas`, the results show that the adoption of importance sampling allows the former to achieve high quality solutions for very low probabilities, that is, for $p < 0.2$—the average cost of the local optima obtained by `2.5-opt-EEais` is between 1% and 3% less than that of `2.5-opt-EEas`. The observed differences are significant according to the paired t-test. For $p \geq 0.3$, the average solution cost and the computation time of `2.5-opt-EEais` are comparable to the ones of `2.5-opt-EEas`.

Concerning the comparison to `2.5-opt-EEs`, the following results are most noteworthy. `2.5-opt-EEais` reaches an average solution similar to that of

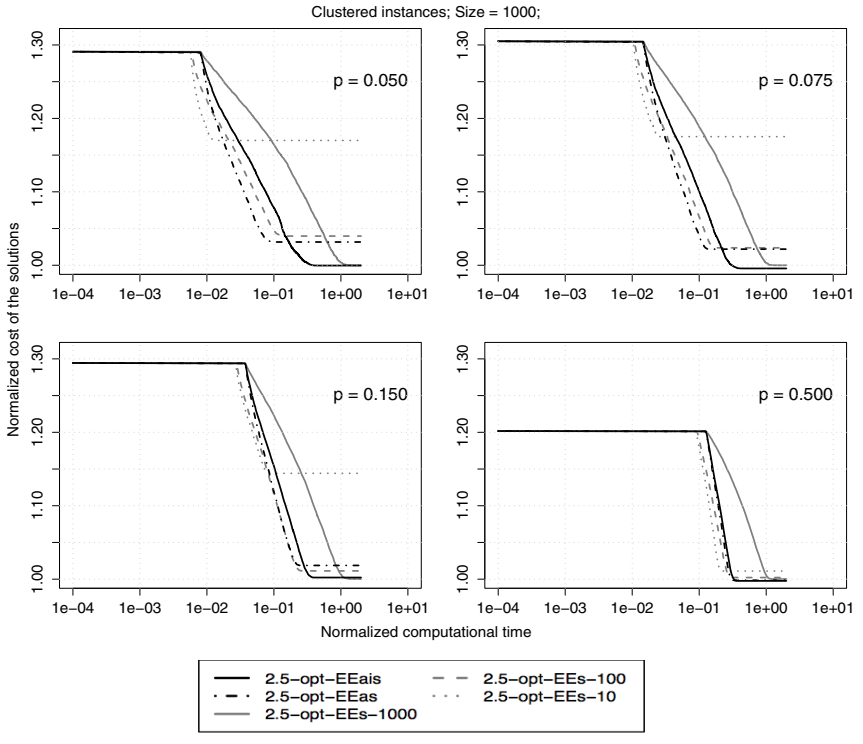


Fig. 4.6. Experimental results on clustered homogeneous PTSP instances of size 1000. The plots represent the cost of the solutions obtained by `2.5-opt-EEas`, `2.5-opt--EEais`, `2.5-opt-EEs-10`, and `2.5-opt-EEs-100` normalized by the one obtained by `2.5-opt-EEs-1000`. Each algorithm is stopped when it reaches a local optimum.

`2.5-opt-EEs-1000`, but it does so approximately four times faster. Compared to `2.5-opt-EEs-100`, `2.5-opt-EEais` reaches for low probability values, $p \leq 0.2$, an average cost that is between 1% and 3% lower (the differences are statistically significant according to the paired t-test), while for $p \geq 0.3$ the two algorithms are comparable. Taking into account both the computation time and the cost of the solutions obtained, we can see that `2.5-opt-EEais` emerges as a clear winner among the considered *estimation-based* algorithms.

Finally, we compared `2.5-opt-EEais` with `2.5-opt-ACs`. In order to avoid over-tuning [84], we generated 100 new instances for each probability level. The results are shown in Figure 4.7. The computational results show that `2.5-opt--EEais` is very competitive. Regarding the time required to reach local optima, irrespective of the probability levels, `2.5-opt-EEais` is approximately 2 orders of magnitude faster than `2.5-opt-ACs`. The average cost of local optima obtained by `2.5-opt-EEais` is comparable to the one of `2.5-opt-ACs`.

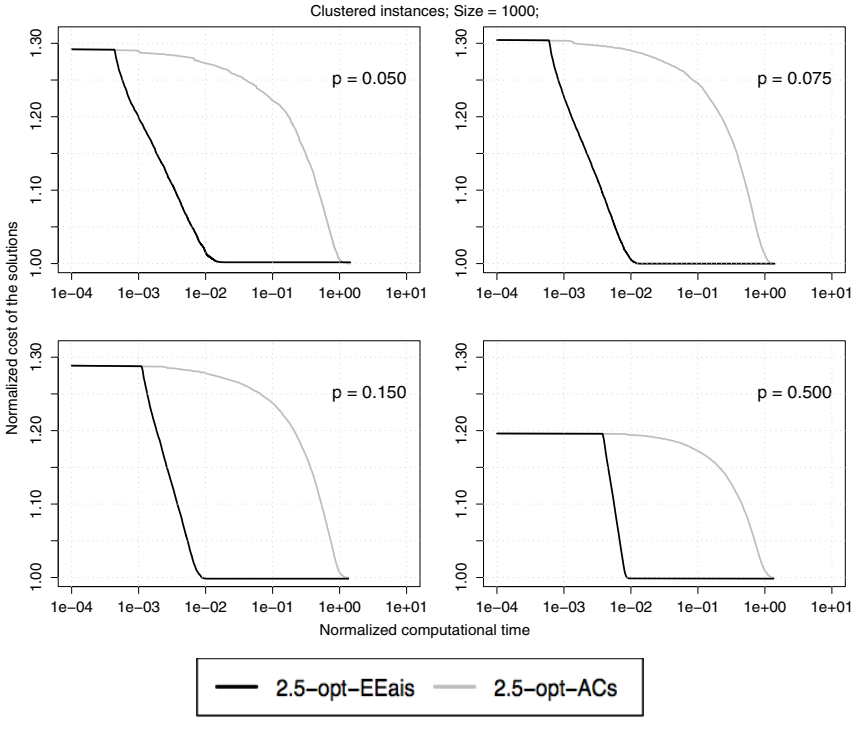


Fig. 4.7. Experimental results on clustered homogeneous PTSP instances of size 1000. The plots represent the solution cost obtained by `2.5-opt-EEais` normalized by the one obtained by `2.5-opt-ACs`. Each algorithm stops upon reaching a local optimum.

4.5 Estimation Based Iterated Local Search

As a final step, we tried to derive a new state-of-the-art SLS algorithm for the PTSP. Clearly, the significant performance gains obtained by `2.5-opt-EEais` over the previous state-of-the-art iterative improvement algorithms should make this feasible. Given the very high performance of iterated local search (ILS) [85] algorithms for the TSP, we decided to adopt this general-purpose SLS method also for the PTSP. That is, we implement our fourth hypothesis:

Hypothesis 4: `2.5-opt-EEais` is a good candidate procedure to derive a new state-of-the-art SLS algorithm for the PTSP.

Our ILS algorithm is a straightforward adaptation of ILS algorithms for the TSP. It starts from a nearest neighbor tour and uses `2.5-opt-EEais` as the subsidiary local search algorithm. The acceptance criterion compares two local optima by using the *t-test* with up to a maximum of n realizations. If no statistically significant difference is detected, the solution with lower cost estimate is accepted. The perturbation consists of a number of simple moves. In particular,

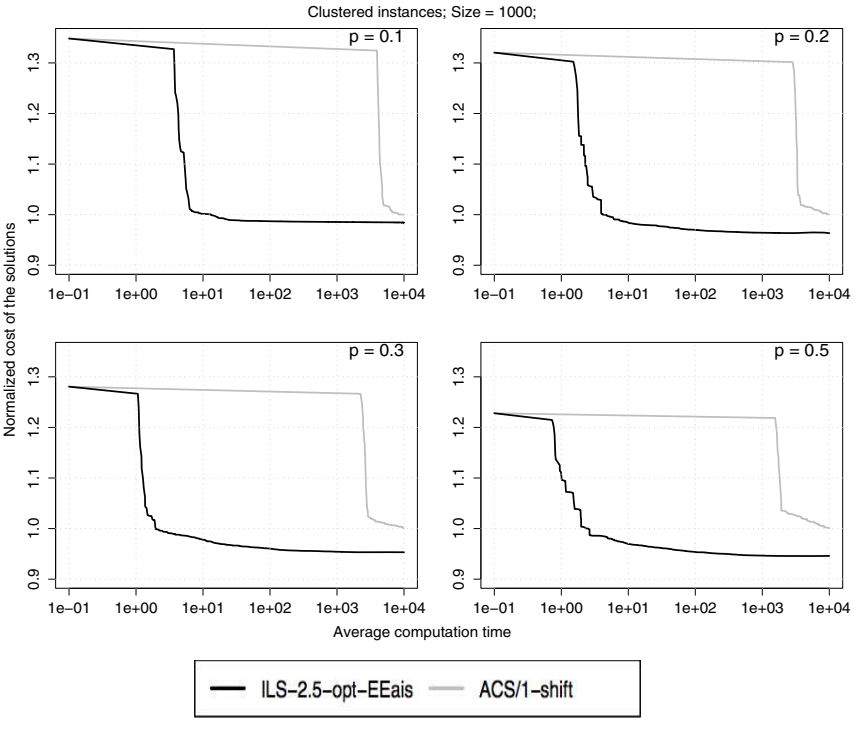


Fig. 4.8. Experimental results on clustered homogeneous PTSP instances of size 1000. Each algorithm is allowed to run for 10 000 seconds. The plots give the solution cost obtained by `ILS-2.5-opt-EEais` normalized by the one obtained by `ACS/1-shift`.

we perform two random double-bridge moves and perturbe the position of $ps\%$ of the nodes, where ps is a parameter. This change of the position is done by picking uniformly at random $ps\%$ of nodes, removing them from the tour and then re-inserting them again according to the farthest insertion heuristic. This composite perturbation is motivated by the change of the usefulness of edge exchange moves and insertion moves, as indicated by the crossover in the relative performance of `2-p-opt` and `1-shift`. We denote the final algorithm by `ILS-2.5-opt-EEais`.

We first tuned the parameter ps of `ILS-2.5-opt-EEais` using `Iterative F-Race`, resulting in a value of $ps = 6$. Next, we compared its performance to `ACS/1-shift` [86], an ant colony optimization algorithm [87] that uses `1-shift` as the underlying local search and was shown to be a state-of-the-art stochastic local search algorithm for the PTSP. For this task, we adopted the parameter settings of `ACS/1-shift` that were fine-tuned in [86]. We compared the two algorithms on clustered instances with 1000 nodes using 10 000 seconds as a stopping criterion.

The experimental results in Figure 4.8 show that `ILS-2.5-opt-EEais` outperforms `ACS/1-shift` both in terms of final solution quality and computation time to reach a specific bound on the solution quality. In fact, the final average solution cost of `ILS-2.5-opt-EEais` is between 1% and 5% lower that that of `ACS/1-shift`; all observed differences are statistically significant according to a paired t-test with $\alpha = 0.05$.

4.6 Conclusions

In this chapter, we presented a case study in engineering an effective SLS algorithm for the PTSP. Our approach has used a bottom-up process. It had the particularity that it focused very strongly on the development and refinement of the underlying iterative improvement algorithm. We do not claim that this strong focus on this element of the process is always necessary. In fact, this process required, in this particular case, a significant number of new ideas. However, we strongly believe that this bottom-up approach is a potentially very successful way of deriving very high performing algorithms.

SLS engineering is relatively a new area of research in SLS algorithms and it is receiving considerable attention in recent years. Therefore, it has a number of avenues open for further contributions. One of the main focus of research in SLS engineering is to develop a framework of principled procedures for SLS design, implementation, analysis, and in-depth experimental studies. Moreover, SLS engineering needs a tight integration with tools that support the algorithm development process such as software frameworks, statistical tools, experimental design, automated tuning, search space analysis, and efficient data structures. Given researchers' and practitioners' quest for high performing algorithms, we strongly believe that SLS engineering is going to play a major role in designing SLS algorithms.

Acknowledgements

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.