Ant Colony Optimization and Local Search based on Exact and Estimated Objective Values for the Probabilistic Traveling Salesman Problem

Leonora Bianchi and Luca Maria Gambardella

Technical Report No. IDSIA-06-07
June 22, 2007

IDSIA / USI-SUPSI
Dalle Molle Institute for Artificial Intelligence
Galleria 2, 6928 Manno, Switzerland

Ant Colony Optimization and Local Search based on Exact and Estimated Objective Values for the Probabilistic Traveling Salesman Problem

Leonora Bianchi and Luca Maria Gambardella

June 22, 2007

Abstract

This paper deals with a general choice that one faces when developing an algorithm for a stochastic optimization problem: either design problem-specific algorithms that exploit the exact objective function, or to consider algorithms that only use estimated values of the objective function, which are very general and for which simple non-sophisticated versions can be quite easily designed. The Probabilistic Traveling Salesman Problem and the Ant Colony Optimization metaheuristic are used as a case study for this general issue. We consider four Ant Colony Optimization algorithms with different characteristics. Two algorithms exploit the exact objective function of the problem, and the other two use only estimated values of the objective function by Monte Carlo sampling. For each of these two groups, we consider both hybrid and non-hybrid versions (that is, with and without the application of a local search procedure). Computational experiments show that the hybrid version based on exact objective values outperforms the other variants and other state-of-the-art metaheuristics from the literature. Experimental analysis on a benchmark of instances designed on purpose let us identify in which conditions the performance of estimation-based variants can be competitive with the others.

1 Introduction

The Probabilistic Traveling Salesman Problem (PTSP) can be seen as the prototype of stochastic combinatorial optimization problems. It consists in finding a Hamiltonian a priori tour visiting a set of customers of minimal expected length, given that each customer has a known probability of actually requiring a visit, and that a customer is skipped by the traveling salesman if it is revealed that it does not require a visit.

The similarity of the PTSP with the classical, deterministic Traveling Salesman Problem (TSP), makes it a natural choice when extending an algorithm from the domain of deterministic to that of stochastic problems. Among the key differences between stochastic and deterministic problems there are the fact that the objective function in stochastic problems is usually much more computationally expensive than in deterministic ones, and in some cases the stochastic problem is so complex that no analytical closed-form expression exists for the objective function. These two features of stochastic problems makes it a common choice to integrate in optimization algorithms in general, and in heuristics and metaheuristics in particular, approximations of the objective function. In particular, three different situations may arise when computing the objective function of a stochastic problem: (1) closed-form expressions for the expected values are available, and the objective function is *computed exactly* based on these objective values; (2) as in case 1, closed-form expressions for the expected values are available, but the objective function is considered to be too time consuming to be always computed during optimization. Therefore, *ad-hoc and fast approximations* of the objective are designed and used during optimization (possibly alternating exact and approximated evaluations); (3) the problem is so complex in terms of decision variables and/or in terms of probabilistic dependences, that no closed-form expression exists for the expected values, therefore, the objective function is *estimated by simulation* or by *Monte Carlo sampling.* All the three above situations have been addressed by the PTSP literature, with a varying number of papers. Let us now review the main contributions of each type.

The main contributions making use of the exact PTSP objective function are the following. Rossi and Gavioli [29] adapt to the PTSP two well known TSP tour construction heuristics, the Nearest Neighbor and the Clarke-Wright algorithms, by explicitly including the customers probability in the evaluation and selection of new portions of the tour to construct. Rossi and Gavioli experimentally evaluate the two heuristics on homogeneous PTSP instances with up to 100 uniformly distributed customers, and compare their performance to that of the classical deterministic Nearest Neighbor and Clarke-Wright heuristics that solve a TSP. Their conclusion is that for the tested instance it is important to use the techniques specifically designed for the PTSP only for instances with more than 50 customers, and customers probability less than 0.6. In [1], Bertsimas proposes two local search procedures for homogeneous PTSP instances, the 2-p-opt and the 1-shift, and applies them to the solution produced by the TSP heuristic called the Space Filling curve. For the two local search procedures, the author gives recursive equations that efficiently compute the change of the homogeneous PTSP expected cost of two neighboring solutions. The efficiency is comparable to that of similar local search procedures for the (deterministic) TSP. Subsequently, other authors have verified that the recursive equations proposed in [1] are not correct, and derive correct expressions both for the homogeneous PTSP (Bianchi et al. [6]) and for the heterogeneous PTSP (Bianchi and Campbell [3]).

Ad hoc approximations are surrogates of the exact objective function that are faster to be computed, and that are used to speed up the optimization process. Examples of ad hoc approximations include: mathematical approximations of the exact objective function, such as truncated expressions neglecting terms estimated to be small; the use of the objective function of some deterministic combinatorial optimization problem related to the stochastic one; the use of scenarios, instead of considering the true probabilistic model. In the context of the PTSP, ad hoc approximations have been used by several authors. Bianchi et al. [4, 5] use an ACO algorithm originally developed for the TSP, ACS, to solve homogeneous PTSP instances, and show that for high customers probability, ACS performs better than using the exact PTSP objective function. Branke and Guntsch [10, 11] consider an ad hoc approximation, called 'depth-approximation', based on neglecting some terms estimated to be small. They integrate it in an ACO algorithm, and show that computation times can be significantly reduced, without worsening the solution quality. Tang and Miller-Hooks [31] propose approximated expressions for the cost of adding and inserting a customer in a PTSP solution. These approximations, combined with the 2-opt and Or-opt local search, obtain results which are competitive with those obtained with the exact objective function. Liu [27] integrates one of the local search algorithms proposed in [31] into a scatter search framework, and obtains better results. Campbell [12] focuses on PTSP instances with very small average customer probability, and develops approximations based on aggregating customers together. Aggregation is used to obtain fast solutions that are then improved by local search, obtaining better results than not using aggregation.

The idea behind the second type of approximations, that is, sampling approximations, is the following. Since in most stochastic problems the objective function (or a part of it) consists in computing an expected value of a quantity, it is possible to estimate expectations, and thus the objective function, by sampling or by Monte Carlo-type simulations. This type of approximation is interesting because, first of all, it is applicable to stochastic problems where a closed-form expression for the expected value(s) is not available (for example, due to complicated dependencies among random variables), secondly, the sampling-approximation is quite general, and is not tied to the particular analytical form of the objective function, like ad-hoc approximations are. Thus, any knowledge acquired on a given problem could be in principle exploited in any other one. The PTSP is used in the literature also as a test problem to investigate the performance of algorithms that exploit sampling-based approximations, since solutions obtained by such techniques can be eventually assessed by the exact objective function. Thus, the primal goal of applying algorithms using sampling-based approximations to the PTSP, is not to find state-of-the art solutions, but to investigate the behavior and the performance of such algorithms in a controlled environment. Papers that investigate sampling-based algorithms for the PTSP include [9], [24], and [8]. In [9], Bowler et al. analyze experimentally by a stochastic Simulated Annealing algorithm the asymptotic behavior of (sub) optimal homogeneous PTSP solutions, in the limit of pn (customers probability times number of customers) going to infinity. The PTSP objective function is estimated by sampling, and the sampling estimation error is used as a sort of temperature regulated during the annealing process. Gutjahr [24] proposes an Ant Colony Optimization metaheuristic called S-ACO, and experimentally tests it on the PTSP and on the TSP with time windows and stochastic service times. S-ACO implements the sampling approximation by considering variable numbers of samples, increasing with the iteration counter of the algorithm. The precise number of samples is decided on the base of a statistical parametric test. Finally, Birattari et al. [8] propose the ACO/F-Race algorithm, where at each iteration the selection of the new best solution is done with a procedure called F-Race, which is more sophisticated than the simple parametric test of S-ACO. As explained in [7], F-Race consists in a series of steps at each of which a new scenario ω is sampled and is used for evaluating the solutions that are still in the race (at the beginning, all solutions generated by ants in a given iteration, together with the current best solution, are in the race). At each step, a Friedman test is performed and solutions that are statistically dominated by at least another one are discarded from the race. The solution that wins the race is stored as the new current best solution. Preliminary experiments on homogeneous instances of the PTSP problem have shown that ACO/F-Race improves over the parametric procedure adopted by S-ACO.

Given the current state of research on the PTSP, there is still space for at least two interesting issues, that are the ones addressed by this paper: first of all, the design of high performance algorithms for the PTSP, exploiting the already available efficient 1-shift and 2-p-opt local search developed in [6] and [3]. Up to now, only simple heuristics exploited these efficient local searches, but their integration with metaheuristics is a promising area. In this paper, we consider the Ant Colony Optimization metaheuristic (ACO) and the 1-shift local search, with the goal to obtain a state-of-the-art algorithm for the PTSP. The second issue which is still open in the literature is the is the design of sampling-based local search algorithms, that could be used to hybridize sampling-based metaheuristics possibly enhancing their performance. In this paper, we thus consider also a sampling-based version of ACO with a sampling-based version of 1-shift.

The interest in considering exact versus estimation-based algorithms for the PTSP lies also in the fact that these two approaches have very different features: on the one hand, algorithms exploiting the exact objective function are in principle more efficient, since there is no uncertainty when comparing solutions; on the other hand, they are tailored to the specific problem, and they cannot be easily extended to others; moreover, a lot of effort might be necessary in order to design efficient algorithms such as the 1-shift and the 2-p-opt local search, which exploit rather complicated recursive expressions for the cost of a local search move. On the contrary, methods based on the sampling approximation can be very general, and implementations of varying degrees of sophistication can be designed.

The remainder of the paper is organized as follows. Section 2 formally defines the PTSP and the notation used. Section 3 explains the working principles of the ACO metaheuristic, and describes in detail the four ACO variants that we propose (hybrid and non-hybrid, and based on exact and estimated objective function values). Section 4 is devoted to the computational experiments performed for the assessment of our proposed ACO algorithms, and for comparisons with other methods. This section also describes the test instances used and the tuning of ACO parameters. Finally, section 5 summarizes the main results achieved by this paper.

2 Statement of the problem

For many delivery companies, only a subset of the customers require a pickup or delivery each day. Information may be not available far enough in advance to create optimal schedules each day for those customers that do require a visit or the cost to acquire sufficient computational power to find such solutions may be prohibitive. For these reasons, it is not unusual to design a tour containing all customers (called *a priori* tour), and each day to follow the ordering of this a priori tour to visit only the customers requiring a visit that day (see Figure 1). The tour actually traveled each day when the customers requiring a visit are revealed is called *a posteriori* tour. The

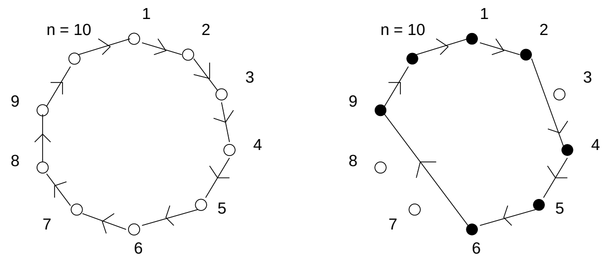

Figure 1: An a priori tour (left), and a possible a posteriori tour (right) obtained from the a priori tour by visiting only the customers requiring a visit on a given random realization of customers, and by keeping the same order of visit as in the a priori tour. In this example, a random realization where only customers number 1, 2, 4, 5, 6, 9, 10 require a visits is shown.

problem of finding an a priori tour of minimum expected cost, given a set of customers each with a given probability of requiring a visit, defines the PTSP. In the remainder of this paper, the terms a priori tour tour, and solution, will be used interchangeably.

More formally, let $N = \{i \mid i = 1, 2, \dots, n\}$ be a set of n customers. For each pair of customers $i, j \in N$, $d(i, j)$ represents the distance between i and j. Here, we assume that the distances are symmetric, that is, $d(i, j) = d(j, i)$. In the remainder of the thesis, distances will also be referred to as costs. An a priori tour $\lambda = (\lambda(1), \lambda(2), \dots, \lambda(n))$ is a permutation over N, that is, a tour visiting all customers exactly once. Given the independent probability p_i that customer i requires a visit, $q_i = 1 - p_i$ is the probability that i does not require a visit. The general case where customers probabilities p_i may be different, is referred to as *heterogeneous* PTSP, while if probabilities are all equal ($p_i = p$ for every customer i), the problem is called *homogeneous* PTSP. We will use the following convention for any customer index i:

$$i := \begin{cases} i(\mathrm{mod} n) & \text{iff } i \neq 0 \text{ and } i \neq n \\ n & \text{otherwise,} \end{cases} \tag{1}$$

where $i(\mathrm{mod}\, n)$ is the remainder of the division of i by n. The reason for defining the above convention is that we want to use as customer indices numbers from 1 to n (and not from 0 to $n-1$), and we need to make computations with the 'modulus' operator. The expected length of an a priori tour λ can be computed in $O(n^2)$ time with the following expression derived by Jaillet [25]

$$E[L(\lambda)] = \sum_{i=1}^{n} \sum_{r=1}^{n-1} d(\lambda(i), \lambda(i+r)) p_{\lambda(i)} p_{\lambda(i+r)} \prod_{i+1}^{i+r-1} q_\lambda. \tag{2}$$

We use the following notation for any $i, j \in \{1, 2, \dots, n\}$

$$\prod_{i}^{j} q_\lambda := \begin{cases} \prod_{t=i}^{j} q_{\lambda(t)} & \text{if } 0 \leq j - i < n - 1 \\ \prod_{t=i}^{n} q_{\lambda(t)} \prod_{u=1}^{j} q_{\lambda(u)} & \text{if } i - j > 1 \\ 1 & \text{otherwise.} \end{cases} \tag{3}$$

The expression for the objective function (equation (2)) has the following intuitive explanation: each term in the summation represents the distance between the i^{th} customer and the $(i+r)^{\text{th}}$ customer weighted by the probability that the two customers require a visit ($p_{\lambda(i)}p_{\lambda(i+r)}$) while the $r-1$ customers between them do not require a visit ($\prod_{i+1}^{i+r-1} q_\lambda$).

In the homogeneous PTSP, where $p_i = p$ and $q_i = q$ for every customer i, the expression for the objective function still requires $O(n^2)$ computation time, but it is a bit simpler than equation (2):

$$E[L(\lambda)] = \sum_{i=1}^{n} \sum_{r=1}^{n-1} p^2 q^{r-1} d(\lambda(i), \lambda(i+r)). \tag{4}$$

3 Description of ACO and local search algorithms

In this section we first describe the general working principles of ACO and local search algorithms, and we then concentrate on the specific algorithms studied in this paper, that are: pACS with its hybrid version pACS+1-shift which are based on exact objective function evaluations, and pACS-S with is hybrid version pACS-S+1-shift-S which are based on sampling-estimated objective function evaluations.

3.1 General working principles

The inspiring concept that links optimization with biological ants is based on the observation of their foraging behavior: when walking on routes from the nest to a source of food, ants seem to find not simply a random route, but a quite 'good' one, in terms of shortness, or, equivalently, in terms of time of travel; thus, their behavior allows them to solve an optimization problem. This kind of success of biological ants is entirely explained by their type of communication and by their way of deciding where to go: While walking, ants deposit a chemical called *pheromone* on the ground, and they tend to choose routes marked by strong pheromone concentrations. Given two initially unexplored routes, a short and a long one, between the nest and the source of food, ants choose at first randomly which one to walk. Ants that have chosen, at first by chance, the shorter route are the first to reach the food and to start their return to the nest. Therefore, pheromone starts to accumulate faster on the shorter route than on the longer one. Subsequent ants tend to follow the shorter route because it has more pheromone, thus reinforcing it more and more, and further attracting other ants on the good route.

Combinatorial problems addressed by ACO are usually encoded by a *construction graph* $G = (V, A)$, a completely connected graph whose nodes V are components of solutions, and arcs A are connections between components. Finding a solution means constructing a feasible walk in G. For example, in the TSP nodes correspond to customers, arcs correspond to streets connecting customers, and a feasible solution is a Hamiltonian path on the graph. Some other examples are described in [19].

The ACO algorithm is essentially the interplay of three procedures [15]: ConstructAntsSolutions, UpdatePheromones, and DeamonActions, as represented by Algorithm

1.

ConstructAntsSolutions is the process by which artificial ants construct walks on the construction graph incrementally and stochastically. For a given ant, the probability p_{kl} to go from a node k to a feasible successor node l is an increasing function of τ_{kl} and $\eta_{kl}(u)$, where τ_{kl} is the pheromone on arc (k, l), and $\eta_{kl}(u)$ is the *heuristic* value of arc (k, l), which should be a reasonable guess of how good arc (k, l) is (for example, in the TSP η_{kl} is the reciprocal of the distance between customer k and customer l). The heuristic value may depend on the partial walk u.

UpdatePheromones is the process by which pheromone is modified on arcs. Pheromone may be both increased and decreased. Pheromone is modified (decreased) by each ant on each arc as soon as it is added to a partial walk on the construction graph, this operation is called *local update*. Moreover, pheromone is further modified (increased) on selected good solutions to more strongly bias the search in future iterations, and this operation is called *global update*. Decreasing pheromone on selected arcs is important, in order to avoid too rapid convergence of the algorithm to suboptimal solutions. Interestingly, pheromone decreases also in the biological environment, due to evaporation.

DeamonActions are centralized operations, such as comparing solution values among ants in order to find the best solution, or running a local search procedure. In this last case, the ACO algorithm is also referred to as 'hybrid', since it exploits a procedure that is itself another, in principle sophisticated, algorithm. In its simplest form, a local search algorithm works as follows. Consider a starting current solution, evaluate its neighboring solutions (according to a given neighborhood structure), and set the best or the first found neighbor which is better than the current solution as new current solution. Iterate this process until an improving solution is found in the neighborhood of a current solution. The local search stops when the current solution is better than all its neighbors, that is, when the current solution is a *local optimum*.

Algorithm 1 Ant Colony Optimization (ACO)

```
while termination condition not met do
  ScheduleActivities
    ConstructAntsSolutions
    UpdatePheromone
    DeamonActions
  end ScheduleActivities
end while
```

The first algorithms based on the ant colony analogy appeared at the beginning of the nineties in a paper by Dorigo et al. [17] later published as [18]. ACO is now a widely studied metaheuristic for combinatorial optimization problems, as the recent book by Dorigo and Stützle [19] testifies. Several convergence proofs have been provided for the ACO metaheuristic. Convergence in value, which, informally, means that the algorithm will find at least once the optimal solution, has been given by Gutjahr [21] and by Stützle and Dorigo [30]. Convergence in solution, which, informally, means that the algorithm will generate over and over the optimal solution, has been given by Gutjahr [22]. For details and for a comprehensive discussion, see Dorigo and Stützle [19].

3.2 Algorithms based on exact objective function evaluations

3.2.1 The pACS and pACS+1-shift algorithms

Because of the structural similarity between the PTSP and the TSP (the solution structure is the same, only the cost of a solution is different), as a first implementation of the ACO algorithm for the PTSP, we consider an adaptation to the PTSP of the ACS algorithm by Dorigo and Gambardella [16] which was successfully applied to the TSP. We call this algorithm probabilistic ACS, or pACS. We call pACS+1-shift the hybridized version of pACS obtained by inserting in pACS the 1-shift local search (this local search will be described in detail in Section 3.2.2). Preliminary experiments that compare pACS with ACS on homogeneous PTSP instances are reported in [4, 5]. The skeleton of pACS (and of pACS+1-shift) is shown by Algorithm 2. The procedures

Algorithm 2 pACS [and pACS+1-shift]

```
1: Initialization
2: for iteration $k = 1, 2, \dots$ do
3:   Initialize best ant solution $\lambda_{BA}$
4:   for ant $a = 1, 2, \dots, m$ do
5:     ConstructAntSolution [each ant constructs its solution $\lambda_a$]
6:     if $E[L(\lambda_a)] < E[L(\lambda_{BA})]$ then
7:       set $\lambda_{BA} = \lambda_a$
8:     end if
9:   end for
10:  if $E[L(\lambda_{BA})] < E[L(\lambda_{BSF})]$ then
11:    set $\lambda_{BSF} = \lambda_{BA}$
12:  end if
13:  [Apply 1-shift($\lambda_{BA}, \lambda_{BSF}$) (Algorithm 4)] % only in pACS+1-shift
14:  GlobalPheromoneUpdate
15: end for
```

Initialization, ConstructAntSolution, and GlobalPheromoneUpdate work as follows.

Initialization consists of four operations: the positioning of m ants on their starting customers, the initialization of the best-so-far-solution λ_{BSF}, the computation and initialization of the heuristic information η, and the initialization of pheromone τ. Note that η and τ are bidimensional matrices of information, where η_{ij} and τ_{ij} are the values of the heuristic information, respectively pheromone, on the arc (i, j) that goes from customer i to customer j. Initialization is done as follows in pACS: the starting customer of each ant is chosen randomly; heuristic information is so that $\eta_{ij} = 1/d_{ij}$, where d_{ij} is the distance between i and j; pheromone values are set all equal to τ_0, which is computed according to

$$\tau_0 = \frac{1}{n \cdot E[L(\lambda_{FI})]}, \tag{5}$$

where $E[L(\lambda_{FI})]$ is the expected length of the tour constructed by the Farthest Insertion heuristic [26]. As noted in [16], in the denominator of equation (5) any very rough approximation of the optimal solution value would suffice, therefore, the expected cost of other heuristics could be used for the initialization of τ_0. Observe that, in this simple pACS version, η is a static information that is computed just once during the initialization phase.

ConstructAntSolution is the procedure by which each ant probabilistically builds a tour by choosing the next customer to move to on the basis of the two types of information: the pheromone τ and the heuristic information η. When an ant a is on city i, the next city is chosen as follows.

- With probability q_0 (a parameter), a city j that maximizes $\tau_{ij} \cdot \eta_{ij}^{\beta}$ is chosen in the set $J_a(i)$ of the cities not yet visited by ant a. Here, β is a parameter which determines the relative influence of the heuristic information.

- With probability $1 - q_0$, a city j is chosen randomly with a probability given by

$$p_a(i,j) = \begin{cases} \frac{\tau_{ij} \cdot \eta_{ij}^{\beta}}{\sum_{r \in J_a(i)} \tau_{ir} \cdot \eta_{ir}^{\beta}}, & \text{if } j \in J_a(i) \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Hence, with probability q_0 the ant chooses the best city according to the pheromone trail and to the distance between cities, while with probability $1 - q_0$ it explores the search space in a biased way.

The procedure ConstructAntSolution also takes care of local pheromone updates, where each ant, after it has chosen the next city to move to, applies the following local update rule:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \rho \cdot \tau_0, \tag{7}$$

where τ_0 is the initial pheromone parameter, and ρ, $0 < \rho \leq 1$, is another parameter. The effect of the local updating rule is to make less desirable an arc which has already been chosen by an ant, so that the exploration of different tours is favored during one iteration of the algorithm.

After each ant has built its solution, the best ant solution λ_{BA} is updated and stored (steps 6 to 8 of Algorithm 2) for future comparison with the best-so-far-solution λ_{BSF} (steps 10 to 12 of Algorithm 2). After λ_{BSF} has also been updated, the GlobalPheromoneUpdate function is applied to modify pheromone on arcs belonging to λ_{BSF} with the following global updating rule

$$\tau_{ij} \leftarrow (1-\alpha) \cdot \tau_{ij} + \alpha \cdot \Delta\tau_{ij}, \tag{8}$$

where

$$\Delta\tau_{ij} = \frac{1}{E[L(\lambda_{BSF})]} \tag{9}$$

with $0 < \alpha \leq 1$ being the pheromone decay parameter, and $E[L(\lambda_{BSF})]$ is the expected cost of the best-so-far solution.

As we have already pointed out, pACS is a straightforward adaptation of the ACS algorithm originally designed for the TSP [16]. Basically, if in the above description of pACS (in equations (5) and (9), and in steps 6 to 10 of Algorithm 2) we substitute the expected length of a solution with the length of it, we obtain ACS. Thus, from the point of view of code, pACS is very similar to ACS. Note, however, that from the complexity point of view there are more important differences, since the expected length of a solution needs $O(n^2)$ time to be computed, while the length of a solution only requires $O(n)$ time. It is useful here to consider in more detail what is the asymptotic time complexity of each iteration of pACS, as a function of the number of customers n and of the number of ants m:

$$\begin{aligned} t(\text{one iteration of pACS or ACS}) = {} & m \cdot t(\mathsf{ConstructAntSolution}) \\ & + m \cdot t(\text{computation of the objective value}) \\ & + t(\mathsf{GlobalPheromoneUpdate}). \end{aligned} \tag{10}$$

Procedures ConstructAntSolution and GlobalPheromoneUpdate require respectively $O(n^2)$ and $O(n)$ time both in pACS and ACS. Thus, we have

$$t(\text{one iteration of pACS}) = 2m \cdot O(n^2) + O(n) = O(n^2), \tag{11}$$
$$t(\text{one iteration of ACS}) = m \cdot O(n^2) + (m+1)O(n) = O(n^2). \tag{12}$$

The asymptotic time complexity is the same in the two algorithms ($O(n^2)$), but the constant involved are different. In practice, given the same computation time is allowed, pACS will be able to perform far fewer iterations than ACS.

The hybrid version of pACS, pACS+1-shift, is obtained very simply by applying the 1-shift local search to the best ant solution after each iteration, as shown in step 13 of Algorithm 2.

3.2.2 The 1-shift local search

The 1-shift neighborhood of an a priori tour is the set of tours obtained by moving a node which is at position i to position j of the tour, with the intervening nodes being shifted backwards one space accordingly, as in Figure 2. The number of neighbors generated by 1-shift moves applied to one a priori tour is $O(n^2)$. When the order in which the 1-shift neighborhood is explored is lexicographic, the cost of a 1-shift move can be evaluated exactly and recursively in constant time. For the derivation of these expression in the homogeneous and heterogeneous PTSP, see, respectively, [6] and [3]. Here, we just illustrate the final recursive expressions in case of homogeneous and heterogeneous PTSP instances.

Consider, without loss of generality, a tour $\zeta = (1, 2, \ldots, i, i+1, \ldots, j, j+1, \ldots, n)$, and denote by $\zeta_{i,j}$ a tour obtained from ζ by moving node i to position j and shifting backwards one space the nodes $i+1, ..., j$, where $i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, n\}$, and $i \neq j$ (see Figure 2). Let $\Delta E_{i,j}$ denote the change in the expected length $E[L(\zeta_{i,j})] - E[L(\zeta)]$. Let $j = i + k$, with $1 \leq k \leq n - 1$ (we are using the notation expressed by equation (1)).

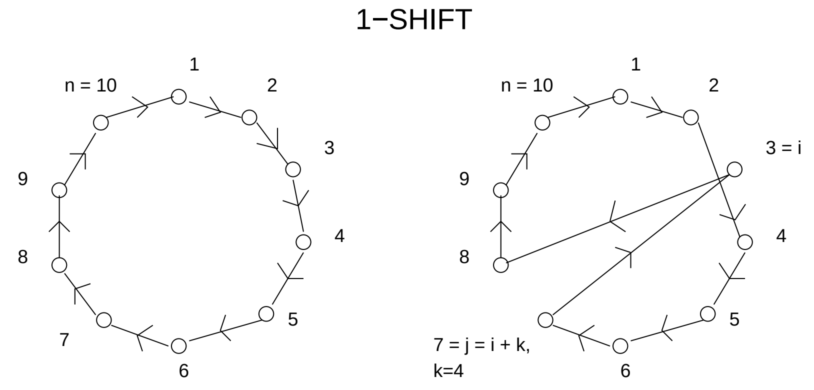

Figure 2: A tour $\zeta = (1, 2, ..., i, i+1, ..., j, j+1, ..., n)$ (left) and a tour $\zeta_{i,j}$ belonging to its 1-shift neighborhood (right) obtained from ζ by moving node i to position j and shifting backwards the nodes $(i+1, ..., j)$, with $n = 10$, $i = 3$, $j = 7$.

Move cost for homogeneous PTSP instances Given the two-dimensional matrices of partial results A and B

$$A_{i,k} = \sum_{r=k}^{n-1} q^{r-1} d(i, i+r) \quad \text{and} \quad B_{i,k} = \sum_{r=k}^{n-1} q^{r-1} d(i-r, i), \tag{13}$$

for $k = 1$, the move cost is

$$\Delta E_{i,i+1} = p^3[q^{-1}A_{i,2} - (B_{i,1} - B_{i,n-1}) - (A_{i+1,1} - A_{i+1,n-1}) + q^{-1}B_{i+1,2}], \tag{14}$$

and for $k \geq 2$, the move cost is

$$\begin{aligned} \Delta E_{i,j} = \Delta E_{i,j-1} + p^2[&(q^{n-k} - q^{-k})(qA_{i,k} - A_{i,k+1}) \\ &+(q^{k-n} - q^{k-1})(qB_{i,n-k} - B_{i,n-k+1}) \\ &+(1 - q^{-1})(q^k B_{i,1} - B_{j,k+1}) \\ &+(q^{-1} - 1)(q^{n-k} A_{i,1} - A_{j,n-k+1}) \\ &+(q - 1)(A_{j,1} - A_{j,n-k}) \\ &+(1 - q)(B_{j,1} - B_{j,k})]. \end{aligned} \tag{15}$$

Move cost for heterogeneous PTSP instances Let us first define the following products

$$Q_{i,j} = \prod_{i}^{j} q, \quad \overline{Q}_{i,j} = \prod_{j+1}^{i+n-1} q, \tag{16}$$

then, we re-define the matrices A and B in terms of Q and $\overline{Q}$:

$$A_{i,k} = \sum_{r=k}^{n-1} d(i, i+r) p_i p_{i+r} Q_{i+1,i+r-1} \tag{17}$$

$$B_{i,k} = \sum_{r=k}^{n-1} d(i-r, i) p_{i-r} p_i Q_{i-r+1,i-1}. \tag{18}$$

Now, for heterogeneous PTSP instances with $k = 1$, the move cost is

$$\begin{aligned} \Delta E_{i,i+1} = \left(\frac{1}{q_{i+1}} - 1\right) A_{i,2} + (q_{i+1} - 1)(B_{i,1} - B_{i,n-1}) \\ + (q_i - 1)\left(A_{i+1,1} - A_{i+1,n-1}\right) + \left(\frac{1}{q_i} - 1\right) B_{i+1,2}, \end{aligned} \tag{19}$$

and for $k \geq 2$, the move cost is

$$\begin{aligned} \Delta E_{i,j} = \Delta E_{i,j-1} &+ (\overline{Q}_{i,j} - \frac{1}{Q_{i,j}})(q_j A_{i,k} - A_{i,k+1}) \\ &+ (\frac{1}{\overline{Q}_{i,j}} - Q_{i,j})(B_{i,n-k} - \frac{1}{q_j} B_{i,n-k+1}) \\ &+ (1 - \frac{1}{q_j}) Q'_{i,j} B_{i,1} + (\frac{1}{q_i} - 1) B_{j,k+1} \\ &+ (1 - q_j)\overline{Q}_{i,j} A_{i,1} + (1 - \frac{1}{q_i}) A_{j,n-k+1} \\ &+ (q_i - 1)(A_{j,1} - A_{j,n-k}) \\ &+ (1 - q_i)(B_{j,1} - B_{j,k}). \end{aligned} \tag{20}$$

Pseudocode of the 1-shift local search The 1-shift local search proceeds in two phases. The first phase consists of only exploring the moves that swap two consecutive nodes $\lambda(i)$ and $\lambda(i+1)$ of the current tour λ, as represented in pseudocode by Algorithm 3, named Swapping_local_search. The costs $\Delta E_{i,i+1}$ are computed for every value of i (by means of equation (14) or (19)), and each time a negative $\Delta E_{i,i+1}$ is encountered, one immediately swaps the two nodes involved. Note that since the computation of $\Delta E_{i,i+1}$ only involves two rows of A and B, one can proceed to the next pair of nodes without recomputing each entire matrix. The local search has, as second argument, a tour λ_{BSF} which is the best-so-far tour known by the algorithm calling the Swapping_local_search function. At the end of the first phase of the local search, an a priori tour is reached for which every $\Delta E_{i,i+1}$ is positive, and the matrices A and B are complete and correct for that tour.

The second phase of the local search consists of computing $\Delta E_{i,j}$ with $j = i + k$ and $k \geq 2$ recursively by means of equation 15 or 20. The 1-shift local search is represented in pseudocode by Algorithm 4. It is implemented as a 'best-improvement' local search, that is, the whole neighborhood is explored and the current solution is updated with the best (improving) neighbor solution. When there are not improving solutions in the neighborhood, or when the time is over, the search stops.

Algorithm 3 Swapping_local_search(λ, λ_{BSF})

```
for ($i = 1, 2, \dots, n$) do
  compute rows $i$ and $i + 1$ of matrices $A$ and $B$ relative to the current solution $\lambda$
  compute $\Delta E_{i,i+1}$ according to equation (19)
  if ($\Delta E_{i,i+1} < 0$) then
    $\lambda :=$ the tour obtained from $\lambda$ by switching node $\lambda(i)$ with node $\lambda(i + 1)$
    if $E[L(\lambda)] < E[L(\lambda_{BSF})]$ then
      $\lambda_{BSF} := \lambda$
    end if
  end if
end for
if (the starting solution has changed) then
  re-compute the full matrices $A$ and $B$
end if
```

Algorithm 4 1-shift(λ, λ_{BSF})

```
(1) Swapping_local_search($\lambda, \lambda_{BSF}$)
while (locally optimal tour not found and time is not over) do
  for ($i = 1, 2, \dots, n$) do
    for ($k = 2, \dots, n - 2$) do
      compute $\Delta E_{i,i+k}$ according to equation (15)
    end for
  end for
  if ($\arg\min_{i,k} \Delta E_{i,i+k} < 0$) then
    $\lambda :=$ tour obtained from $\lambda$ by inserting $\lambda(i)$ after $\lambda(i + k)$
    if $E[L(\lambda)] < E[L(\lambda_{BSF})]$ then
      $\lambda_{BSF} := \lambda$
    end if
    go to (1)
  else
    return locally optimal tour $\lambda$
  end if
end while
```

3.3 Algorithms based on sampling-estimated objective function evaluations

Here, we describe an extension of pACS and of 1-shift where the algorithms do not use the exact objective function, but instead evaluate solutions and local search moves by the 'sampling approximation' of the objective function.

The sampling approximation is based on the observation that the PTSP objective function computes an expected value of a random quantity, and therefore it may be estimated by a sample average. More precisely, given an a priori tour λ, the objective function may be written as

$$E[L(\lambda)] = \sum_{\omega \subseteq V} p(\omega) L(\lambda_{|_\omega}). \tag{21}$$

In the above expression, ω is a subset of the set of customers V, $L(\lambda_{|_\omega})$ is the length of the tour λ, pruned in such a way as to only visit the customers in ω, skipping the others, and $p(\omega)$ is the probability for the subset of customers ω to require a visit:

$$p(\omega) = \prod_{i \in \omega} p_i \prod_{i \in V \setminus \omega} q_i. \tag{22}$$

The objective function, as expressed by equation (21), computes the expected length of the tour λ, over all possible random subsets of customers.

The idea, in sampling approximation, is to *estimate* the exact expected cost (21) through sampling in the following way. The length $L(\lambda)$ is a discrete random variable, taking the value $L(\lambda_{|_\omega})$ with probability $p(\omega)$. Let ω_i, $i \in 1, 3, ..., N$ be subsets of the original set V of n customers sampled independently with probability $p(\omega_i)$. The sampling approximation to $E[L(\lambda)]$ is the following

$$E_{S_N}[L(\lambda)] = \frac{1}{N} \sum_{i=1}^{N} L(\lambda_{|_{\omega_i}}). \tag{23}$$

The time complexity of sampling approximation is $O(Nn)$, therefore, the smaller the sample size N, the bigger the time gain of sampling approximation with respect to the $O(n^2)$ exact objective function. Nevertheless, the smaller N, the bigger the difference, or error, between approximated and exact computation. Thanks to the strong law of large number, we can say that the error of sampling approximation is $O(1/\sqrt{N})$.

3.3.1 The pACS-S and pACS-S+1-shift-S algorithm

We call our ACO algorithm based on the sampling approximation pACS-S, and pACS-S+1-shift-S the hybridized version of pACS-S obtained by inserting in pACS-S the 1-shift-S local search (this local search will be described in detail in Section 3.3.2). The pseudocode is shown in Algorithm 5. pACS-S uses a variable number of samples N, linear in the iteration counter k, and quadratic in the number of customers n, similarly to what has been done by Gutjahr in [24]. Using a variable instead of a

Algorithm 5 pACS-S [and pACS-S+1-shift-S]

```
 1: Initialization [like in pACS]
 2: for iteration $k = 1, 2, \ldots$ do
 3:    Initialize best ant solution $\lambda_{BA}$
 4:    Set N = NumberOfSamples($k$) [apply Equation (24)]
 5:    for ant $a = 1, 2, \ldots, m$ do
 6:       ConstructAntSolution [each ant constructs its solution $\lambda_a$]
 7:       GenerateSamples($N$)
 8:       Compute $E_{S_N}[L(\lambda_a)]$ and re-compute $E_{S_N}[L(\lambda_{BA})]$ using the last generated
          samples
 9:       if $E_{S_N}[L(\lambda_a)] < E_{S_N}[L(\lambda_{BA})]$ then
10:          set $\lambda_{BA} = \lambda_a$
11:       end if
12:    end for
13:    GenerateSamples($N$)
14:    Re-compute $E_{S_N}[L(\lambda_{BA})]$ and $E_{S_N}[L(\lambda_{BSF})]$ using the last generated samples
15:    if $E_{S_N}[L(\lambda_{BA})] < E_{S_N}[L(\lambda_{BSF})]$ then
16:       set $\lambda_{BSF} = \lambda_{BA}$
17:    end if
18:    [Apply 1-shift-S($\lambda_{BA}, \lambda_{BSF}$) (Algorithm ??)] % only in pACS-S+1-shift-S
19:    GlobalPheromoneUpdate [using $\Delta\tau_{ij} = (E_{S_N}[L(\lambda_{BSF})])^{-1}$]
20: end for
21: Compute $E[L(\lambda_{BSF})]$
```

fixed number of samples seems appropriate also from a theoretical point of view, as shown in [23]. Here, N is computed according to the following rule

$$N = c + \lfloor b \cdot n^2 \cdot k \rfloor, \tag{24}$$

where c and b are two parameters.

In pACS-S, after each solution has been constructed, it is evaluated by the sampling approximation with a freshly generated set of samples (step 8 of Algorithm 5). The same set of samples is used to re-evaluate the best-ant solution λ_{BA}. After the ants construction phase is over, at each iteration a new set of samples is generated, in order to update the best-so-far solution λ_{BSF} (step 13 to 17 of Algorithm 5). Only at the end of the algorithm the final best-so-far solution is evaluated once with the exact objective function (step 21 of Algorithm 5).

Similarly to the hybridization of pACS, the hybrid version of pACS-S, pACS-S+1-shift-S, is obtained very simply by applying the 1-shift-S local search to the (estimated) best ant solution after each iteration, as shown in step 18 of Algorithm 2.

3.3.2 The 1-shift-S local search

Sampling approximation of the move cost The 1-shift-S local search is based on the sampling based approximation $\Delta_{S_N} E_{i,j}$ of the exact move cost $\Delta E_{i,j}$, as defined in section 3.2.2. More precisely, given a set of N samples of customers $\omega_1, \ldots, \omega_k, \ldots, \omega_N$, sampled independently according to probability $p(\omega_k)$ (see Equation (22)), the Sampling approximation for the 1-shift is defined as follows

$$\Delta_{S_N} E_{i,j} = E_{S_N}[L(\zeta_{i,j})] - E_{S_N}[L(\zeta)], \tag{25}$$

where

$$E_{S_N}[L(\zeta_{i,j})] = \frac{1}{N}\sum_{k=1}^{N} L(\zeta_{i,j|_{\omega_k}}) \text{ and } E_{S_N}[L(\zeta)] = \frac{1}{N}\sum_{k=1}^{N} L(\zeta_{|_{\omega_k}}). \tag{26}$$

Equation (25) can be rewritten as a sample average over length differences, each one corresponding to a sample ω_k of customers. For this purpose, let us introduce the following definitions

Definition 1 *Given a subset of customers $\omega \subseteq V$, and a node $i \in V$,*

$$pred_\omega(i) = \begin{cases} i, & \textit{if } i \in \omega, \\ \textit{the first-met node } r \in \omega \textit{ going backward from } i \textit{ along } \zeta, & \textit{otherwise.} \end{cases} \tag{27}$$

Definition 2 *Given a subset of customers $\omega \subseteq V$, and a node $i \in V$,*

$$succ_\omega(i) = \begin{cases} i, & \textit{if } i \in \omega, \\ \textit{the first-met node } r \in \omega \textit{ going onward from } i \textit{ along } \zeta, & \textit{otherwise.} \end{cases} \tag{28}$$

Equipped with the above definitions, the length difference of the two a posteriori tours induced by ζ and $\zeta_{i,j}$ on a given subset of customers ω_k is the following

$$\Delta_{\omega_k} L_{i,j} = \begin{cases} 0, & \text{if } i \notin \omega_k, \\ \quad d(pred_{\omega_k}(i-1), succ_{\omega_k}(i+1)) & \\ +d(pred_{\omega_k}(j), i) + d(i, succ_{\omega_k}(j+1)) & \text{otherwise,} \\ -d(pred_{\omega_k}(j), succ_{\omega_k}(j+1)) & \\ -d(pred_{\omega_k}(i-1), i) - d(i, succ_{\omega_k}(i+1)), & \end{cases} \tag{29}$$

and the sampling approximation of 1-shift can be computed with the following expression

$$\Delta_{S_N} E_{i,j} = \frac{1}{N}\sum_{k=1}^{N} \Delta_{\omega_k} L_{i,j}. \tag{30}$$

Since the time required to compute a predecessor or a successor node is $O(n)$ in the worst case, the time complexity for computing $\Delta_{S_N} E_{i,j}$ is $O(Nn)$, which is much higher than the constant time complexity of 1-shift. Note, however, that there is the possibility to decrease at least by a constant factor the complexity of $\Delta_{S_N} E_{i,j}$, if the neighborhood is explored lexicographically, and if the same set of samples is kept fixed during the neighborhood exploration. Let us see in more detail how this is done.

Suppose that the 1-shift neighborhood is explored lexicographically, such that for each $i = 1, 2, \ldots, n$ and for each $j = 1, 2, \ldots, n$, the value of $\Delta_{S_N} E_{i,j}$ is computed in this order by means of Equation (30). Suppose also that before starting the double cycle over indexes i and j, N random samples of the subsets of customers are generated independently, and this set is kept fixed until all $\Delta_{S_N} E_{i,j}$ are evaluated. It is not difficult to see that in such a situation it is possible to compute some of the successor and predecessor nodes needed for the computation of $\Delta_{\omega_k} L_{i,j}$ (see Equation (29)) recursively. In fact, for any node $i \in V$ and any subset of customers ω, one can compute $succ_\omega(i)$ and $pred_\omega(i)$ recursively, and thus faster, in the following ways

$$succ_\omega(i) = \begin{cases} succ_\omega(i) & \text{(by Definition 2 in } O(n)), & \text{if } i-1 \in \omega \\ succ_\omega(i-1) & \text{(recursively in } O(1)), & \text{otherwise.} \end{cases} \tag{31}$$

$$pred_\omega(i) = \begin{cases} pred_\omega(i-1) & \text{(recursively in } O(1)), & \text{if } i \notin \omega \\ i & \text{(in } O(1)), & \text{otherwise.} \end{cases} \tag{32}$$

Pseudocode of the 1-shift-S local search The move cost expression $\Delta_{S_N} E_{i,j}$ in principle can be used inside a local search algorithm without any restriction on the order in which the 1-shift neighbors are explored, and both in first-improvement or best-improvement mode. However, since our goal is to compare the effectiveness of this approximated move cost with the exact one, we have developed 1-shift-S by keeping the same exploration strategy (best-improvement with lexicographic neighborhood exploration) as the 1-shift algorithm based on the exact recursive move costs (by keeping the lexicographic order, we are also able to speed up the computation of the move cost, as previously explained). The pseudocode of 1-shift-S is very similar to the one of the exact recursive 1-shift, except for two differences: first, the exact move cost $\Delta E_{i,j}$ is substituted by $\Delta_{S_N} E_{i,j}$; second, there is no distinction between a first phase (where only single swap moves were checked) and a second phase. The pseudocode of 1-shift-S is shown in Algorithm 6.

4 Computational experiments

This section focuses on the computational experiments done to evaluate the performance of the four ACO algorithms proposed (that is, pACS, pACS+1-shift, pACS-S, and pACS-S+1-shift-S). Section 4.1 describes two sets of test instances used in the computational experiments: one generated by the authors (PTSPLIB benchmark), and one taken from the literature (TMH benchmark). Section 4.2 describes the parameter tuning of the ACO algorithms. Finally, sections 4.3 and 4.4 analyze the computational results obtained, respectively, on the PTSPLIB and on the TMH benchmark. Experiments on the TMH benchmark include comparisons with other metaheuristics from the literature.

All experiments have been run on a machine with two processors Intel Xeon with 1.70GHz CPU and 904MB RAM, running the GNU/Linux Debian 2.4.27 operating

Algorithm 6 1-shift-S(λ, λ_{BSF}) with number of samples N

```
while (locally optimal tour not found and time is not over) do
  GenerateSamples($N$)
  for ($i = 1, 2, \ldots, n$) do
    for ($k = 1, \ldots, n-2$) do
      compute $\Delta_{S_N} E_{i,i+k}$ using equation (30)
    end for
  end for
  if ($\min_{i,k} \Delta_{S_N} E_{i,i+k} < 0$) then
    $(i, k) := \arg\min_{i,k} \Delta_{S_N} E_{i,i+k}$
    $\lambda :=$ tour obtained from $\lambda$ by inserting $\lambda(i)$ after $\lambda(i+k)$
    GenerateSamples($N$)
    Compute $E_{S_N}[L(\lambda)]$ and (re-)compute $E_{S_N}[L(\lambda_{BSF})]$ using the last generated
    samples
    if $E_{S_N}[L(\lambda)] < E_{S_N}[L(\lambda_{BSF})]$ then
      $\lambda_{BSF} := \lambda$
    end if
  else
    return locally optimal tour $\lambda$
  end if
end while
```

system, and each ACO algorithm has been run once for a CPU time equal to $n^2/100$.

4.1 Test instances

In this paper we have considered two benchmarks of instances, the PTSPLIB and the TMH benchmark. These two benchmarks have very different characteristics, as we are going to explain.

The PTSPLIB benchmark This benchmark has been designed by us with the main goal of analyzing the behavior of optimization algorithms for different levels of stochasticity (average customer probability, and variance of customer probability). In general, a PTSP instance is characterized by two types of information: distances between each couple of customers, and probability for each customer of requiring a visit. Since customer distances are what characterize instances of the TSP, we have used instances from the well known TSPLIB benchmark [32] for this type of information (hence, the name 'PTSPLIB' we gave to our benchmark). For each TSP instance, we have then considered different customer probability configurations. Homogeneous PTSP instances have been generated by simply associating a TSP instance one single probability value, corresponding to all customers probabilities. For heterogeneous PTSP instances different sets of customer probability values have been randomly generated according to a probability distribution.

We have considered 4 TSP instances, with n ranging from 100 to 198, as reported in Table 1. For these instances we have computed Euclidean distances between cus-

TSPLIB name [32]	n
kroA100	100
eil101	101
ch150	150
d198	198

Table 1: TSPLIB instances providing customers coordinates, from which Euclidean distances have been computed. Each of these TSP instances has been combined with 54 different customer probability configurations characterized by average probability and variance as reported in Table 2.

tomers, on the base of the customers coordinates provided by the TSPLIB. Customers probabilities have been generated as follows. For homogeneous PTSP instances the customer probability has been varied from 0.1 to 0.9 with an increment of 0.1. For heterogeneous PTSP instances, each customer probability p_i, $i = 1, 2, \ldots, n$ has been generated randomly, according to the 'beta probability distribution'. We have chosen this probability distribution because it is defined on the finite interval $(0, 1)$, and it can easily model different average and variance of customers probability values by choosing the appropriate parameters. The 'beta probability distribution' $\beta_{a,b}(p_i)$ is defined as follows

$$\beta_{a,b}(p_i) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} p_i^{a-1}(1-p_i)^{b-1}, \tag{33}$$

where $p_i \in (0, 1)$, and $a, b > 0$ are parameters. The a and b parameters of the 'beta probability distribution' determine the average customer probability p and the variance σ^2 around the average value:

$$p = \frac{a}{a+b} \tag{34}$$

$$\sigma^2 = \frac{ab}{(a+b)^2(a+b+1)}. \tag{35}$$

In our benchmark we have generated different sets of customer probabilities by varying the average customer probability p and the variance σ^2. In order to have direct control on these parameters, rather than on a and b, we have used the inverse of Equation (34) and (35):

$$a = \frac{p\left[p(1-p) - \sigma^2\right]}{\sigma^2}, \tag{36}$$

$$b = \frac{(1-p)\left[p(1-p) - \sigma^2\right]}{\sigma^2}. \tag{37}$$

Note that the variance is limited by the relation $\sigma^2 < p(1-p) \leq 0.25$. Similarly to the homogeneous case, we have considered customer probability sets by varying p between 0.1 and 0.9, with and increment of 0.1. For every value of p, we have considered five different values of variance, corresponding respectively to 1/6, 2/6, 3/6, 4/6, and 5/6 of the maximum variance which is $p(1-p)$. More precisely, we

p	σ^2					
0.1	0	0.015	0.030	0.045	0.060	0.075
0.2	0	0.027	0.053	0.080	0.107	0.133
0.3	0	0.035	0.070	0.105	0.140	0.175
0.4	0	0.040	0.080	0.120	0.160	0.200
0.5	0	0.042	0.083	0.125	0.167	0.208
0.6	0	0.040	0.080	0.120	0.160	0.200
0.7	0	0.035	0.070	0.105	0.140	0.175
0.8	0	0.027	0.053	0.080	0.107	0.133
0.9	0	0.015	0.030	0.045	0.060	0.075

Table 2: Average customer probability p and variance σ^2 characterize the 54 customer probability configurations generated for our PTSPLIB benchmark. Homogeneous PTSP instances correspond to the column with $\sigma^2 = 0$. Heterogeneous probability configurations have been obtained for each TSP instance of Table 1 by using the above values of p and $\sigma^2 (\neq 0)$ for computing a and b parameters (Equation (36)-(37)), and by generating sets of random customer probabilities according the 'beta probability distribution' (Equation (33)).

have set $\sigma^2 \in \{p(1-p)/6, 2p(1-p)/6, 3p(1-p)/6, 4p(1-p)/6, 5p(1-p)/6\}$. For each probability, the different σ^2 values are also referred to as the 16%, 33%, 50%, 66%, and 83% of the maximum variance. In the plots that will be presented in the remainder of this thesis, the variance values will be indicated by percentage. The shape of the corresponding 'beta probability distributions' are shown in Figure 3.

Summarizing, for each of the 4 TSPLIB instances, we have considered 54 customer probability configurations, which in total means 216 PTSP instances. Table 2 reports all the considered values of p and σ^2, including the ones corresponding to homogeneous instances (for which $\sigma^2 = 0$).

The TMH benchmark This benchmark has been generated by Tang and Miller-Hooks [31] and subsequently used by Liu [27]. We chose to run our algorithms also on this benchmark, in order to compare our results with the heuristics and metaheuristics of the two above cited papers. In illustrating the TMH benchmark, we follow the description given by [31]. The benchmark consists of 90 randomly generated PTSP instances with size n' equal to 50, 75, 100. For each problem, $n'+1$ vertex positions (1 depot plus n' customer vertices) (x_i, y_i), $i \in V$ were generated on the basis of a uniform distribution from $[0, 100]^2$. Note that, according to how we have formalized the PTSP in section 2, $n'+1 = n$. The travel distance matrix (d_{ij}) was constructed by setting $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. For each problem size, three groups of problem instances were created. These tree groups of problem instances were categorized on the basis of the different intervals from which their vertex presence probabilities were generated. Presence probabilities of customer vertices were randomly generated from a uniform distribution on the intervals $(0, 0.2]$, $(0, 0.5]$, and $(0, 1]$, one for each problem group. The depot (vertex 0) was always assigned a presence probability of 1.

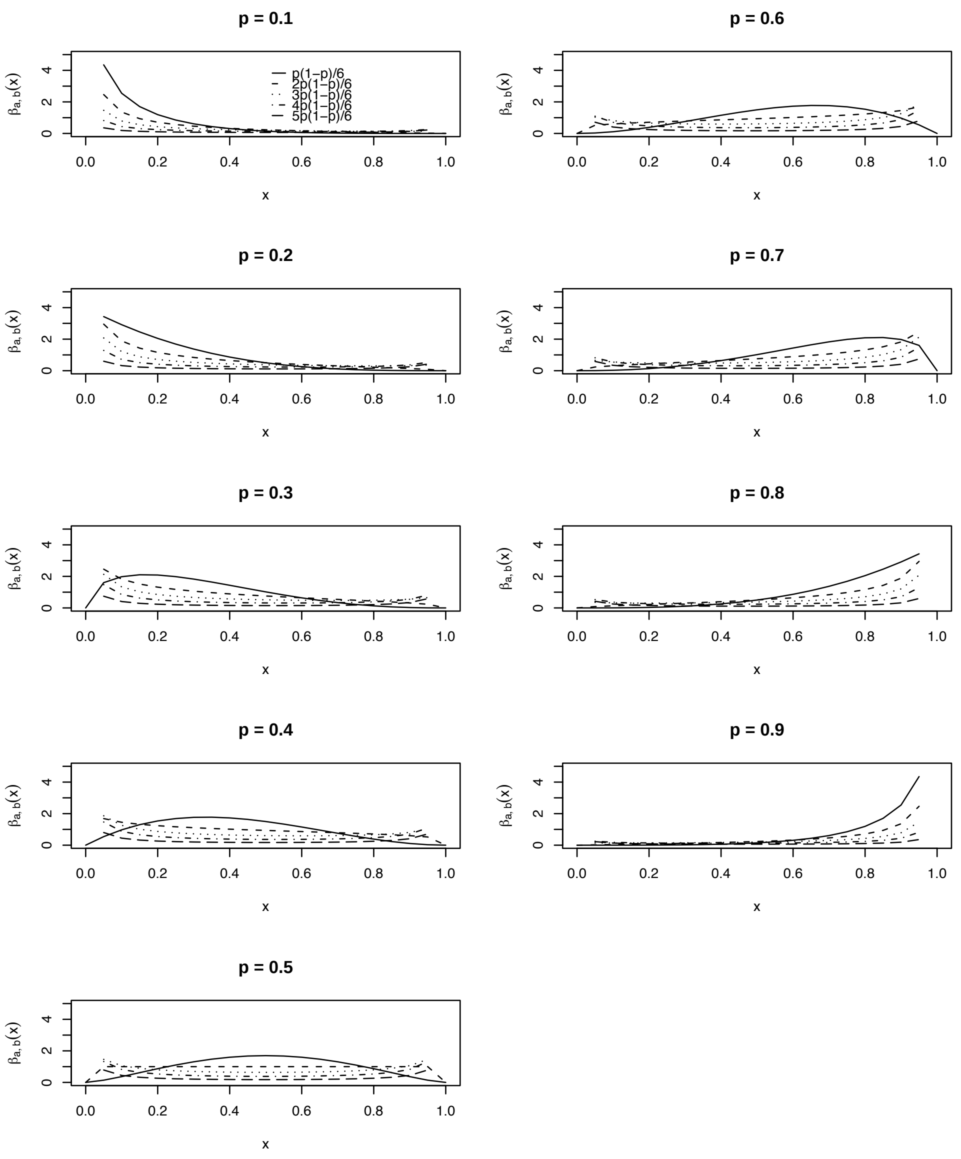

Figure 3: Shape of the ‘beta probability distributions’ used for generating customers probabilities in heterogeneous instances of our PTSP benchmark. The legend in the first plot specifies the type of line used for different values of σ^2, and applies to all plots of the figure.

4.2 Tuning

The four ACO algorithms under study (pACS, pACS+1-shift, pACS-S, and pACS-S+1-shift-S), have been tuned separately, so to spend the same tuning effort for each of them. The instances for tuning have been generated as follows. Customers coordinates correspond to the eil51.tsp instance from the TSPLIB [32], which is a TSP instance with 51 customers where distances are computed according to the Euclidean metric. For this TSP instance, 6 different customers probability configurations have been generated, by considering three values of (average) customer probability $p \in \{0.2, 0.5, 0.8\}$, and two values of variance of the customer probability $\sigma^2 \in \{0, 50\%\}$. Instances with $\sigma^2 = 0$ are homogeneous PTSP instances with fixed probability, while instances with $\sigma^2 = 50\%$ are heterogeneous PTSP instances with randomly generated customers probabilities according to the 'beta probability distribution' as described in section 4.1. Each algorithm was run once on each PTSP instance for 26 CPU seconds, a time corresponding to $n^2/100$, with $n = 51$. The values of the tuning parameters for each ACO algorithm are shown in Table 3. Boldface parameters are those that led to the best average results on the tuning instances, and that have been chosen for running the experiments on the PTSPLIB and TMH benchmarks. We have verified empirically that the algorithms are not very sensitive to parameters ρ, α, and ψ, when their value is equal to 0.1. Thus, we have chosen this value a priori for ρ, α, and ψ, without performing any additional tuning.

Table 3: Parameter sets used for tuning the ACO algorithms under study. Bold entries correspond to the final parameter choice after tuning.

Parameter	Algorithm			
	pACS	pACS+1-shift	pACS-S	pACS-S+1-shift-S
$\rho = \alpha = \psi$	$\mathbf{0.1}$	$\mathbf{0.1}$	$\mathbf{0.1}$	$\mathbf{0.1}$
q_0	$\{0.95, \mathbf{0.98}, 1\}$	$\{\mathbf{0.95}, 0.98, 1\}$	$\{\mathbf{0.95}, 0.98, 1\}$	$\{\mathbf{0.95}, 0.98, 1\}$
β	$\{\mathbf{1}, 2, 3, 4\}$	$\{\mathbf{1}, 2, 3, 4\}$	$\{1, 2, \mathbf{3}, 4\}$	$\{1, 2, \mathbf{3}, 4\}$
m	$\{5, 10, \mathbf{15}, 20\}$	$\{\mathbf{5}, 10, 15, 20\}$	$\{5, \mathbf{10}, 15, 20\}$	$\{\mathbf{5}, 10, 15, 20\}$
N (fixed)	–	–	$\{10, 50, 200\}$	$\{10, 50, 200\}$
c (N variable)	–	–	$\{\mathbf{1}, 10, 50\}$	$\{\mathbf{1}, 10, 50\}$
b (N variable)	–	–	$\{10^{-3}, 10^{-4}, \mathbf{10^{-5}}\}$	$\{10^{-3}, 10^{-4}, \mathbf{10^{-5}}\}$

4.3 Experimental analysis on the PTSPLIB benchmark

This set of experiments has been designed with the goal of analyzing how the algorithms performance varies with different 'stochasticity levels' in the PTSP instances. Loosely speaking, 'stochasticity levels' are those features of a PTSP instance that makes it different from a deterministic TSP instance. In our PTSPLIB benchmark, these features correspond to the average customers probability p, and to its variance σ^2. In fact, when $p = 1$ and $\sigma^2 = 0$, a PTSP instance is an instance of a deterministic TSP. Intuitively, the smaller p and the bigger σ^2, the higher the 'stochasticity' of the PTSP instance.

The main motivation for this set of experiments is the following. It is reasonable to expect that under some level of stochasticity, the optimal TSP solution evaluated with the PTSP objective function is a very good approximation of the optimal PTSP solution. Since the TSP is one of the most well known problems in the operations research literature, there are freely available algorithms, such as CONCORDE [13], that can solve efficiently to optimality even big instances of the problem. For this reason, it is possible that for PTSP instances with a small level of stochasticity, algorithms such as CONCORDE outperform PTSP-specific heuristics or metaheuristics. Therefore, when designing algorithms for the PTSP, it is important to identify the range of stochaticity levels for which PTSP-specific algorithms significantly outperform the already available TSP solvers. This type of information would be useful also in the real world, since an analyst could decide on the base of the stochasticity level of its real data, wether it is more convenient to employ already available software, or to invest in more sophisticated, problem-specific algorithms that explicitly address data uncertainty.

Another motivation for this set of experiments is to see how stochasticity influences exact versus sampling-based, and non-hybrid versus hybrid ACO variants. The interest in this question lies in the very different characteristics (in terms of generality and design effort) of the different variants. Summarizing what we have already pointed out, we can say that: (1) pACS and pACS+1-shift are very problem specific, since they rely on the analytical expression of the objective function; (2) pACS+1-shift required also a big analytical effort in deriving efficient recursive equations for the move cost expression of the 1-shift local search; (3) pACS-S and pACS+1-shift-S are general, since the sampling-estimation of the objective function can in principle be performed on any stochastic combinatorial optimization problem (4) pACS-S and pACS-S+1-shift-S can be implemented at different levels of sophistication, but a simple version like the one considered in this paper does not requires a lot of effort. Knowing which variant performs better in which situation may be important in addressing other stochastic optimization problems or in the real world, when one could decide in advance if it is more convenient to spend effort in designing problem-specific, or more general sampling-based ACO algorithms, and hybrid, or non-hybrid ones.

Let us now describe our experimental results. Comparison among ACO algorithms and the optimal TSP solution is shown in Figure 4. The optimal TSP solution for each instance has been found by the CONCORDE algorithm [13], and then evaluated with the PTSP objective function. Figure 4 reveals that the performance of ACO algorithms is very different for different average customers probability. In particular, there is a *critical probability* that goes from 0.4 (for pACS) to 0.6 (for pACS+1-shift), under which it is really worth solving PTSP instances by ACO, while above the critical probability the problem can be better treated like a TSP. Note that results of pACS+1-shift are reported only up to $p = 0.6$. The reason is that, for bigger probability values, some PTSP instances produced overflow in the 1-shift computation of terms such as $1/Q_{i,j}$, and $1/\overline{Q}_{i,j}$ (see equation (20)).

We will restrict the analysis of subsequent experimental results on the PTSPLIB benchmark to instances with $p \leq 0.5$, which, as we have just seen, are the most significant ones. Figure 4 shows the average gain of each ACO algorithm with respect to the TSP optimal solution, but it does not allow a clear comparison among the

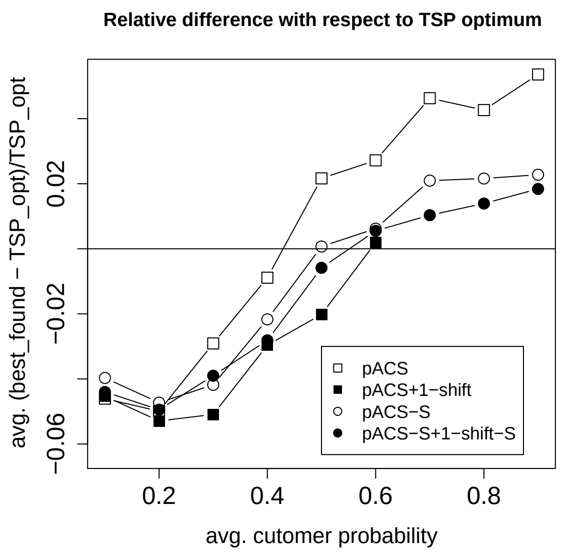

Figure 4: Average relative difference of the best solution found by our proposed ACO algorithms over optimal TSP solutions, versus average customers probability. The horizontal line represents the PTSP value of the optimal TSP solution. Being under the horizontal line means performing better than the optimal TSP solution.

performance of different ACO variants. This type of comparison is more evident in Figure 5, where a boxplot of the ranks of our algorithms, including the optimal TSP solution, is shown. The meaning of the vertical line joining pACS and pACS-S on the left of the plot is the following: according to the Friedman two-way analysis of variance by ranks [14], the difference between the two algorithms is not statistically significant at a confidence level of 95%. Thus, if two algorithms are *not* linked by the vertical line, their difference is statistically significant.

Figure 5 shows that the best performing algorithm is pACS+1-shift, followed by pACS-S+1-shift-S, and that pACS-S and pACS are significantly worse. This means, first of all, that hybrid versions (that is, ACO algorithms using local search) perform better than non-hybrid ones. Moreover, the fact that pACS-S+1-shift-S performs worse than pACS+1-shift means that the analytical effort spent in developing the efficient recursive 1-shift local search has been worth indeed. On the other hand, the difference between the two non-hybrid versions pACS and pACS-S is not statistically significant, and this is an indication that the more general sampling-based algorithms may in principle achieve good quality levels. Given that our pACS-S is a very simple implementation of a sampling-based algorithm, it is possible that more sophisticated versions could outperform the exact-objective-based pACS algorithm, and it would be interesting more research in this direction.

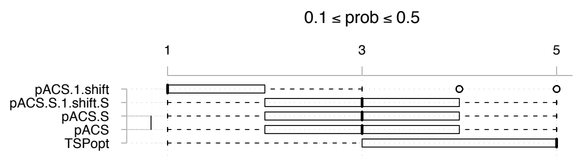

Figure 5: Boxplot of the distribution of ranks of our proposed ACO algorithms, and TSPopt (the optimal solution of the TSP evaluated with the PTSP objective function). The vertical line on the left of the plot which joins pACS-S and pACS means that these two algorithms are not significantly different at a confidence level of 95%, according to the Friedman two-way analysis of variance by ranks [14]. The interpretation of a box is the following: the solid line inside the box is the median rank, the limits of the box are the lower and upper quartile, the 'whiskers' are the smallest and largest values (outliers excepted), and the circles are the outliers.

We can have a deeper understanding of the difference among ACO variants when we look at how their ranks varies according to the variance of the customers probability in PTSP instances (this is the second factor characterizing the 'stochasticity level' of a PTSP instance, besides the average customers probability). This information is shown in Figure 6. It is very interesting to observe that the higher the variance, the smaller the difference among ACO variants. In particular, the performance of pACS+1-shift seems degrade as the variance increases. This means that the supremacy of pACS+1-shift is not absolute, but for instances with very high variance of the customers probability, other, simpler to implement algorithms such as non-hybrid ACO or hybrid ACO based on estimated objective values, may be a better alternative.

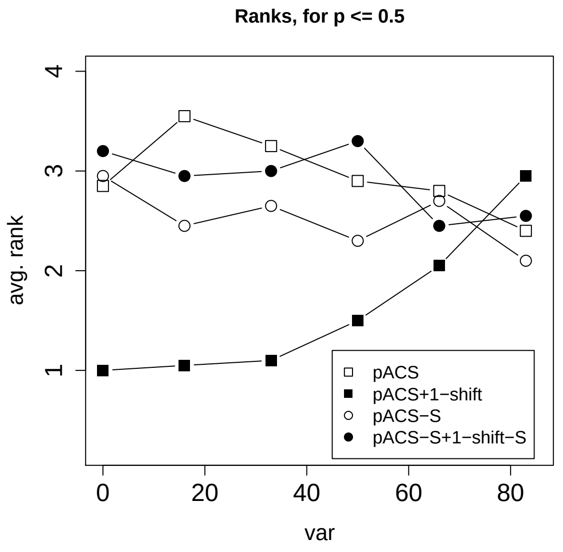

Figure 6: Average ranks of our proposed ACO algorithms, with respect to the variance of the customers probability in PTSP instances of the PTSPLIB benchmark, with average probability $p \leq 0.5$.

Another important type of comparison that we performed is the runtime analysis of the four ACO variants, as shown in Figure 7. The figure reports the normalized value of the best-so-far solution of each algorithm versus the normalized computation time in logarithmic scale. The first observation to do is that non-hybrid versions outperform hybrid ones at the beginning of the run. A second and interesting result is that the sampling-based algorithm pACS-S is faster on average than pACS. Given that at the end of the run these two algorithms are not statistically significant different (see Figure 5), the fact that pACS-S is faster than pACS is an indication that pACS-S is the algorithm to be preferred, in case one does not have the possibility to implement or use a local search. Different considerations must be done for hybrid ACO versions: in fact, in this case the sampling-based algorithm pACS-S+1-shift-S outperforms pACS+1-shift only in a small time window situated circa in the middle of the run, and at the end of the run the latter algorithm is statistically significant better than the former (see Figure 5). This means that the sampling-based local search, in order to be competitive with the recursive exact one, needs to be improved by using more sophisticated search strategies (such as, for instance, non-lexicographic neighborhood exploration, candidate lists, and so on).

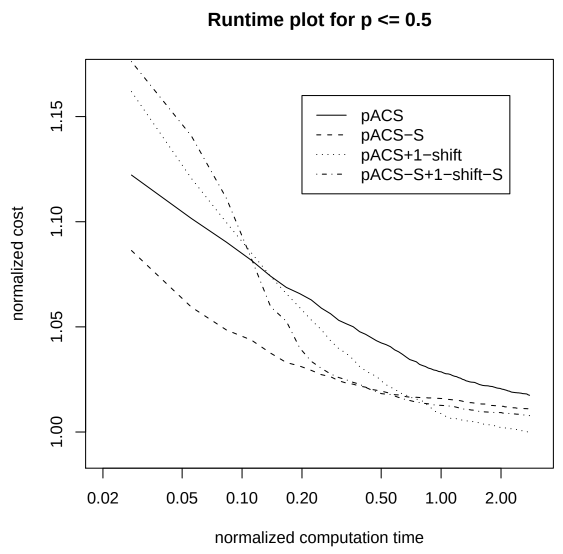

Figure 7: Runtime plot of our proposed ACO algorithms in PTSP instances of the PTSPLIB benchmark, with average probability $p \leq 0.5$.

4.4 Comparison with other methods on the TMH benchmark

In this section we compare the performance of our proposed ACO algorithms with that of other heuristics and metaheuristics from the PTSP literature, namely, TMH-EX proposed by Tang and Miller-Hooks in [31], HSS0, and HSS1 proposed by Liu in [27]. A brief description of these three algorithms follows.

By TMH-EX we denote a simple algorithm [31], which consists in applying the Or-opt local search [28] to a randomly generated starting solution. In [31], this algorithm has been compared to others, particularly to local search algorithms based on different types of ad-hoc approximations of the objective function. The simple Or-opt local search is the one that obtains on average the best results (even if in a bigger computation time than approximation-based algorithms), and this is the reason why we select it for comparison with our ACO algorithms. We use the name 'TMH-EX' for consistency with Liu in [27]. HSS0 and HSS1 [27] are two variants of the scatter search metaheuristic [20], which is a method inspired by evolutionary computation.

The performance comparison on the whole TMH benchmark is summarized by Figure 8, where a boxplot of the average rank of each algorithm is reported, with bars joining algorithms that are not statistically significant different according two the Friedman wo-way analysis of variance by ranks (as previously done in Figure 5 for comparisons on the PTSPLIB benchmark). It is confirmed here that the best performing algorithm is pACS+1-shift, immediately followed by pACS. On this benchmark though, differently from the PTSPLIB one, the sampling-based ACO algorithms are worse, and they are not statistically significant different from the scatter search algorithms.

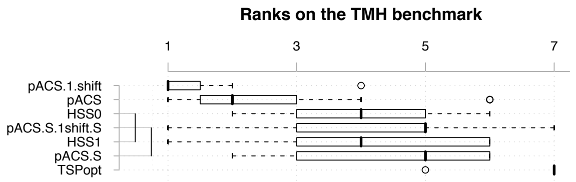

Figure 8: Boxplot of the distribution of ranks of our proposed ACO algorithms, two scatter search algorithms (HSS0 and HSS1), a simple local search (TMH-EX), and TSPopt (the optimal solution of the TSP evaluated with the PTSP objective function). The vertical line on the left of the plot which joins some algorithms means that these algorithms are not significantly different at a confidence level of 95%, according to the Friedman two-way analysis of variance by ranks [14]. The interpretation of a box is the following: the solid line inside the box is the median rank, the limits of the box are the lower and upper quartile, the 'whiskers' are the smallest and largest values (outliers excepted), and the circles are the outliers.

Numerical results are reported in Table 4. The table confirms that pACS+1-shift is on average the best performing algorithm. Only in the group of instances with maximum probability equal to 0.5 and with 76 customers, the best performance has been not achieved by this algorithm, but by pACS-S instead. TMH-EX, HSS0, and HSS1, achieved the same performance of pACS+1-shift only in the group of instances with smallest probability and smallest customers number (that is, maximum probability equal to 0.2, and customers number equal to 51). Unfortunately, it is difficult to draw any conclusion on the comparison of computation times, since experiments have been run on different machines.

5 Conclusions

In this paper we have considered four ACO algorithms with different characteristics. Two algorithms exploit the exact objective function of the problem, and the other two use only estimated values of the objective function by Monte Carlo sampling with a variable number of samples. For each of these two groups, we have considered both hybrid and non-hybrid versions (that is, with and without a local search procedure).

The local search based on the exact objective value is based on a fast and recursive computation of the move cost that has been derived in previous studies, and to our knowledge this is the first time it is combined with a metaheuristic. The local search based on estimated objective values is a simple implementation, and to our knowledge

Table 4: Comparison of our proposed ACO algorithms (that is, pACS, pACS+1-shift, pACS-S, and pACS-S+1-shift-S) with other metaheuristics from the literature on the TMH benchmark.

n'	$(p_{min}, p_{max}]$	TMH-EX		HSS0		HSS1		pACS		pACS+1-shift		pACS-S		pACS-S+1-shift-S	
		E	t^a	E	t^b	E	t^b	E	t^c	E	t^c	E	t^c	E	t^c
50	(0,0.2]	**224.8**	81	**224.8**	*54.1*	224.8	*38.9*	**224.8**	*28.8*	**224.8**	*1.3*	226.8	*63.9*	226.1	*63.9*
	(0,0.5]	341.3	*72.4*	341.6	*55*	341.5	*38.5*	341.1	*29.4*	**340.8**	*2.7*	342.5	*64.0*	342.9	*64.0*
	(0,1]	476.4	*64.8*	452.7	*54.5*	453.2	*37.2*	447.3	*15.1*	**445.6**	*11.0*	448.6	*63.9*	448.6	*63.9*
75	(0,0.2]	266.2	*844.2*	**265.9**	*240.6*	**265.9**	*152.5*	**265.9**	*187.3*	**265.9**	*14.8*	267.5	*239.8*	267.0	*239.8*
	(0,0.5]	404.9	*721.1*	404.0	*243.7*	404.0	*155.2*	408.6	*107.8*	408.9	*18.9*	**401.6**	*239.9*	402.1	*239.9*
	(0,1]	549.1	*640.8*	532.5	*245.4*	533.8	*156.2*	520.7	*120.3*	**514.6**	*49.0*	516.8	*239.9*	519.1	*239.9*
100	(0,0.2]	301.8	*4484.7*	300.9	*689.9*	300.9	*448.6*	300.9	*624.2*	**300.8**	*26.9*	302.9	*757.3*	302.1	*798.2*
	(0,0.5]	480.2	*4153.2*	463.0	*690.3*	463.4	*459.2*	459.4	*476.0*	**455.5**	*21.6*	460.3	*799.8*	459.0	*799.9*
	(0,1]	649	*3752.8*	632.1	*790.1*	632.4	*512.4*	612.1	*456.9*	**606.5**	*126.7*	613.9	*797.9*	612.1	*798.5*

[a] time on a DEC AlphaServer 1200/533 computer with 1GB RAM
[b] time on an Intel Pentium IV personal computer with 2.8GHz CPU and 512MB RAM
[c] time on an Intel Xeon personal computer with 1.7GHz CPU and 904MB RAM

this is also the first time this type of local search is used in combination with an estimation based ACO.

Overall, the best performing ACO variant is the one based on exact objective values and hybridized with the exact and fast recursive local search. This is shown to significantly outperform also other heuristics and metaheuristics from the literature based on the scatter search and on different types of local search algorithms. Such result suggests that, even if developing an exact and efficient local search in a stochastic problem can be quite a difficult task, it is worth the effort, especially when this local search can be combined with a metaheuristic such as ACO.

One of the main goal of this paper was to analyze the performance of the proposed ACO variants by varying the degree of stochasticity of problem instances, measured in terms of average customers probability, and variance of the customers probability. We have observed the following: there is a *critical probability* that goes from 0.4 (for the worst performing ACO) to 0.6 (for the best performing ACO), under which it is really worth solving the PTSP by ACO, while above the critical probability the problem can be better treated like the classical, deterministic, TSP. Moreover, the higher the variance of the customers probability, the smaller the performance difference among ACO variants. In particular, the performance of the best algorithm (which is the hybrid one based on exact objective values) seems to degrade as the variance increases. This means that for instances with very high variance of the customers probability, other, simpler to implement algorithms such as non-hybrid ACO or hybrid ACO based on estimated objective values, may be a better alternative.

The runtime analysis revealed that non-hybrid versions outperform hybrid ones at the beginning of the run, but later they stagnate on worse local optima than hybrid versions. This suggests that, in applications where one needs very fast results, it is better to consider ACO algorithms without local search. Among the non-hybrid versions, the one based on estimated objective values is the fastest. This is an interesting indication that the estimation based approximation might be chosen also in other stochastic problems even if the exact objective function is known, just with the purpose of speeding up the computation and having a faster algorithm.

Acknowledgments

The authors are indebted to Elise Miller-Hooks and Hao Tang for sharing the test instances of the TMH benchmark. The authors also thank Yu-Hsin Liu for discussing some technical aspects of his paper [27] that have helped us to make comparisons with the algorithms he had developed. Leonora Bianchi acknowledges the support of the FNS grant Nr. 200021-108007.

References

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

[2] L. Bianchi and A. M. Campbell. Extension of the 2-p-opt and 1-shift algorithms to the heterogeneous probabilistic traveling salesman problem. Technical Report IDSIA-01-04, IDSIA - Dalle Molle Institute for Artificial Intelligence, Manno, Switzerland, 2004.

[3] L. Bianchi and A. M. Campbell. Extension of the 2-p-opt and 1-shift algorithms to the heterogeneous probabilistic traveling salesman problem. *European Journal of Operational Research*, 176(1):131–144, 2007.

[4] L. Bianchi, L. M. Gambardella, and M. Dorigo. An ant colony optimization approach to the probabilistic traveling salesman problem. In J. J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacañas, and H.-P. Schwefel, editors, *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN VII)*, volume 2439 of *Lecture Notes in Computer Science*, pages 883–892. Springer, London, UK, 2002.

[5] L. Bianchi, L. M. Gambardella, and M. Dorigo. Solving the homogeneous probabilistic traveling salesman problem by the ACO metaheuristic. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Proceedings of the 3rd International Workshop on Ant Algorithms (ANTS 2002)*, volume 2463 of *Lecture Notes in Computer Science*, pages 176–187. Springer, London, UK, 2002.

[6] L. Bianchi, J. Knowles, and N. Bowler. Local search for the probabilistic traveling salesman problem: correction to the 2-p-opt and 1-shift algorithms. *European Journal of Operational Research*, 162(1):206–219, 2005.

[7] M. Birattari, P. Balaprakash, and M. Dorigo. ACO/F-Race: ant colony optimization and racing techniques for combinatorial optimization under uncertainty. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. J. Gutjahr, R. F. Hartl, and M. Reimann, editors, *Proceedings of the 6th Metaheuristics International Conference (MIC 2005)*, pages 107–112, 2005.

[8] M. Birattari, P. Balaprakash, and M. Dorigo. The ACO/F-RACE algorithm for combinatorial optimization under uncertainty. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. J. Gutjahr, R. F. Hartl, and M. Reimann, editors, *Metaheuristics - Progress in Complex Systems Optimization.* Operations Research/Computer Science Interfaces Series. Springer, Berlin, Germany, 2006.

[9] N. E. Bowler, T. M. A. Fink, and R. C. Ball. Characterization of the probabilistic traveling salesman problem. *Physical Review E*, 68(036703), 2003.

[10] J. Branke and M. Guntsch. New ideas for applying ant colony optimization to the probabilistic TSP. In *Proceedings of the 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2003)*, volume 2611 of *Lecture Notes in Computer Science*, pages 165–175. Springer, Berlin, Germany, 2003.

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

[12] A. M. Campbell. Aggregation for the probabilistic traveling salesman problem. *Computers & Operations Research*, 33:2703–2724, 2006.

[13] Concorde TSP solver. `http://www.tsp.gatech.edu/concorde.html`.

[14] W. J. Conover. *Practical Nonparametric Statistics.* John Wiley & Sons, New York, NY, USA, 1999.

[15] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[16] M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1:53–66, 1997.

[17] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: an autocatalytic optimization process. Technical Report 91-016, Department of Electronics, Politecnico di Milano, Milan, Italy, 1991.

[18] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 26(1):29–41, 1996.

[19] M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press, Cambridge, MA, USA, 2004.

[20] F. Glover. A template for scatter search and path relinking. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenaurer, and D. Snyers, editors, *Artificial Evolution*, volume 1363 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 1998.

[21] W. J. Gutjahr. A graph-based Ant System and its convergence. *Future Generation Computer Systems*, 16(8):873–888, 2000.

[22] W. J. Gutjahr. ACO algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters*, 82(3):145–153, 2002.

[23] W. J. Gutjahr. A converging ACO algorithm for stochastic combinatorial optimization. In *Proceedings of the 2nd Symposium on Stochastic Algorithms, Foundations and Applicaions (SAGA 2003)*, volume 2827 of *Lecture Notes in Computer Science*, pages 10–25. Springer, Berlin, Germany, 2003.

[24] W. J. Gutjahr. S-ACO: An ant-based approach to combinatorial optimization under uncertainty. In *Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2004)*, volume 3172 of *Lecture Notes in Computer Science*, pages 238–249. Springer, Berlin, Germany, 2004.

[25] P. Jaillet. *Probabilistic Traveling Salesman Problems.* PhD thesis, MIT, Cambridge, MA, USA, 1985.

[26] D. S. Johnson and L. A. McGeoch. Experimental analysis of heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[27] Y.-H. Liu. A hybrid scatter search for the probabilistic traveling salesman problem. *Computers & Operations Research*, In Press.

[28] I. Or. *Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking.* PhD thesis, Department of Industrial Engineering and Management Sciences, Nortwestern University, Evanston, IL, USA, 1976.

[29] F. A. Rossi and I. Gavioli. Aspects of heuristic methods in the probabilistic traveling salesman problem. In *Advanced School on Statistics in Combinatorial Optimization*, pages 214–227. World Scientific, Singapore, 1987.

[30] T. Stützle and M. Dorigo. A short convergence proof for a class of ACO algorithms. *IEEE Transactions on Evolutionary Computation*, 6(4):358–365, 2002.

[31] H. Tang and E. Miller-Hooks. Approximate procedures for probabilistic traveling salesperson problem. *Transportation Research Record: Journal of the Transportation Research Board*, 1882:27–36, 2004.

[32] TSPLIB. `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.