New Approximation-Based Local Search Algorithms for the Probabilistic Traveling Salesman Problem

Dennis Weyland, Leonora Bianchi, and Luca Maria Gambardella

IDSIA, Dalle Molle Institute for Artificial Intelligence
{dennis,leonora,luca}@idsia.ch
www.idsia.ch

Abstract. In this paper we present new local search algorithms for the Probabilistic Traveling Salesman Problem (PTSP) using sampling and ad-hoc approximation. These algorithms improve both runtime and solution quality of state-of-the-art local search algorithms for the PTSP.

1 Introduction

The field of combinatorial optimization under uncertainty has received increasing attention within the last years. Combinatorial optimization problems containing uncertain and dynamic information can be used for more realistic models of real world problems. One common way of representing the uncertainty is to express input parameters by probability distributions instead of single values. Combinatorial optimization problems using this kind of stochastic information are called Stochastic Combinatorial Optimization Problems (SCOPs). Among the SCOPs the Probabilistic Traveling Salesman Problem (PTSP) is currently one of the most significant problems. A broad overview about recent developments in this field is given in [4].

Since most of the Stochastic Combinatorial Optimization Problems are generalizations of combinatorial optimization problems it is not surprising that they are in practice usually harder to solve than the underlying (deterministic) combinatorial optimization problem. For problem sizes of practical relevance exact approaches are computationally too expensive and therefore heuristics are mostly used to tackle these kind of problems.

Currently local search algorithms play an important role in this field and a lot of algorithms for the PTSP are either itself local search algorithms or use local search algorithms as subroutines. In this paper we introduce new local search algorithms for the Probabilistic Traveling Salesman Problem and show that these new algorithms outperform the state-of-the-art local search algorithms for the PTSP.

2 Probabilistic Traveling Salesman Problem

The Probabilistic Traveling Salesman Problem (PTSP) is a generalization of the well known Traveling Salesman Problem (TSP). In contrast to the TSP each city

R. Moreno-Díaz et al. (Eds.): EUROCAST 2009, LNCS 5717, pp. 681–688, 2009.

in the PTSP has to be visited only with a certain probability, thus allowing more realistic models and scenarios. The goal here is to find a so called a-priori tour that visits all cities exactly once, minimizing the expected cost over all possible a-posteriori tours, where cities which do not require a visit are just skipped without chaning the order of the a-priori tour.

As a generalization of the TSP the PTSP is NP-hard and therefore algorithms computing near optimal solutions in a reasonable amount of time are of great interest. Especially local search algorithms play an important role in this field.

Formally we can define the PTSP over a complete undirected edge- and node-weighted graph $G = \{V, c, p\}$. $V = \{1, 2, \ldots, n\}$ is the set of nodes which represent the customers, $p : V \longrightarrow [0, 1]$ is the probability function that assigns to each node the probability that the node requires a visit and $c : V \times V \longrightarrow \mathbb{R}^+$ is the symmetric cost function that represents the non-negative travel costs between any two nodes. For real world problems c usually obeys the triangle inequality or even represents distances in an Euclidean space. The goal is to find a permutation $\tau : V \rightarrow V$ (the a-priori solution) which minimizes the expected cost over all a-posteriori solutions. A closed-form expression for this cost due to [8] is given in section 3.2.

3 Local Search Algorithms

We propose five new local search algorithms for the PTSP. Starting point for our work is a state-of-the-art local search algorithm for the PTSP which uses the 2.5-opt neighbourhood and a sampling-based approximation for the difference between the expected costs of two neighbouring solutions. It is called *2.5-opt-EEs* and a detailed description can be found in [7] with some extensions in [3].

A description of the 2.5-opt neighbourhood operator, which is a combination of the famous 2-opt and 1-shift neighbourhood operators, can be found in [9]. A lot of common optimization techniques for local search algorithms that are used in our new algorithms are also discussed in that work in the context of the TSP.

After introducing the general local search framework we show how the exact cost of a solution can be computed. Then we give a detailled presentation of our new algorithms.

3.1 Local Search Framework

Algorithm 1 describes the general local search framework for the PTSP. At the beginning an initial solution is created ignoring the additional customer probabilities and using the well known nearest neighbour heuristic for the PTSP. As long as our current solution is not a local optimum, we replace the current solution with an improving one. The way in which we explore the neighbourhood and in which the improving solution is chosen differs among our algorithms and is described in the corresponding chapters.

Algorithm 1. *Local Search Framework for the PTSP*

1. *Create an initial solution S using the nearest neighbour heuristic*
2. *While S is not a local optimum regarding the chosen neighbourhood:*
   Replace S by an improving solution
3. *Return S*

For a better runtime behaviour we use neighbourhood lists and don't look bits which are both common optimization techniques for the PTSP and which are also described in [9]. Additionaly we use delta-evaluation for the comparison of two solutions. Instead of comparing the solution costs directly only the difference of the solution costs is calculated. For the neighbourhood operators used in this work delta-evaluation leads to another major runtime improvement.

3.2 Exact Evaluation

Here we show two ways of computing the exact cost of a solution. The first approach calculates the sum of the a-posteriori costs over all possible combinations of realizations of the random variables each multiplied with the according probability that such realizations occur. Since there are two different possibilities for each customer we have in total 2^n different combinations. This naive computation requires exponential runtime and is therefore not useful for any practical implementations, but can be used as a starting point for an approximation with Monte-Carlo-Sampling (cf. 3.3).

Another way to calculate the exact solution cost analytically is to sum over all edges and multiply their costs with the probability that they occur in the a-posteriori tour. The probability that a certain edge occurs in the a-posteriori tour is the product of the probabilities, that both of its vertices require a visit and that all the vertices that are between them in the a-priori tour do not require a visit. Let $\tau : V \longrightarrow V$ be the permutation that represents our current solution and let $\tau_i = \tau(i) \ \ \forall i \in V$. Then the expected cost of the a-posteriori tour can be written as

$$E(\tau) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} c(\tau_i, \tau_j)\, p(\tau_i)\, p(\tau_j) \prod_{k=i+1}^{j-1} (1 - p(\tau_k))$$
$$+ \sum_{i=1}^{n} \sum_{j=1}^{i-1} c(\tau_i, \tau_j)\, p(\tau_i)\, p(\tau_j) \prod_{k=i+1}^{n} (1 - p(\tau_k)) \prod_{k=1}^{j-1} (1 - p(\tau_k))$$

Using this formulation the expected cost of a solution can be calculated in runtime $\mathcal{O}(n^2)$ by adding the summands in a certain order. Although this is a lot better than the exponential runtime of the first approach, it is still too slow for input instances of reasonable sizes. This is why there is a huge need for fast and accurate approximations of the expected cost of a solution. We introduce such approximations in the next chapters in the context of the algorithms where they are used.

3.3 2.5-Opt-Optimized

This algorithm is based on a state-of-the-art algorithm in [7]. It uses the 2.5-opt neighbourhood within our local search framework. The solutions are explored in a random order in each iteration and the first improving solution is used to replace the current solution.

For the comparison of two solutions we use Monte-Carlo-Sampling with the naive exact computation from 3.2, but instead of summing over all possible scenarios, we sample s scenarios at the beginning of the algorithm using the know probabilities and take the average over the a-posteriori costs regarding the sampled scenarios as an approximation for the expected costs. Using the same samples during the whole run of the algorithm is a well known variance reduction technique. With delta-evaluation it is even possible to calculate the difference of two solutions in the 2.5-opt neighbourhood regarding a particular sample in constant time. This leads to a total runtime of $\mathcal{O}(s)$ for the comparison of two solutions using s samples.

For this algorithm it is necessary to calculate for a customer i and a sample s the first customer in the a-priori tour prior to customer i that requires a visit regarding sample s as well as the first customer after customer i that requires a visit regarding sample s. The runtime of the algorithm can be improved by computing these values at the beginning of the algorithm for the initial solution and by updating them after each improvement step. We call this algorithm *2.5-opt-sampling*.

3.4 1-Shift-Delta

The special structure of the 1-shift neighbourhood makes it possible to push this precalculation process to an extreme. Here we compute for the initial solution the delta-values (differences between the solutions) for all possible 1-shift moves. After each improvement step these values are then updated. For the 1-shift neighbourhood the number of values that really change and require an update stays small in relation to the number of all possible moves. Unfortunately this does not hold for the 2-opt neighbourhood, since large segments of the a-priori tour are reversed in some cases and this makes it impossible to benefit from the precalculations for this neighbourhood operator.

The precalculation of the delta-values makes it possible to explore the whole neighbourhood in each step and to replace the current solution with the best solution in its neighbourhood. We call this algorithm *1-shift-delta*.

3.5 2.5-Opt-Depth

Like *2.5-opt-sampling* this algorithm uses the 2.5-opt neighbourhood within our local search framework. The solutions are also explored in a random order in

each iteration and the first improving solution is used to replace the current solution.

The calculation of the solution costs is based directly on the formula given in the section about the exact evaluation of the solution costs. Instead of summing over all edges, here we sum only over those edges whose vertices have a distance of at most d in the a-priori tour, where d is called the depth of the approximation. The formula changes to

$$E_{depth}(\tau) = \sum_{i=1}^{n} \sum_{j=i+1}^{\min\{i+d,n\}} c(\tau_i, \tau_j)\, p(\tau_i)\, p(\tau_j) \prod_{k=i+1}^{j-1} (1 - p(\tau_k)) + \sum_{i=1}^{n} \sum_{j=1}^{i+d-n} c(\tau_i, \tau_j)\, p(\tau_i)\, p(\tau_j) \prod_{k=i+1}^{n} (1 - p(\tau_k)) \prod_{k=1}^{j-1} (1 - p(\tau_k))$$

The computational time required for this approach is $\mathcal{O}(dn)$. Usually d is a constant between 10 and 50, which leads to a good tradeoff between runtime and approximation accuracy. Another important speedup can be achieved by computing the difference of the costs of two solutions in the 2.5-opt neighbourhood. In this case only the $\mathcal{O}(d^2)$ edges that are at a distance of at most d to at least one of the removed/added edges have to be considered, leading to a runtime of $\mathcal{O}(d^2)$. The resulting local search algorithm is called *2.5-opt-depth*.

3.6 2.5-Opt-Threshold

This algorithm is similar to *2.5-opt-depth* and also uses the 2.5-opt neighbourhood. The solutions are explored in a random order in each iteration and the first improving solution is used to replace the current solution. But instead of summing over all edges whose vertices have a distance of at most d in the a-priori tour, here we sum over all edges that occur in the a-posteriori solution with a probability of at least t, where t, with $0 < t < 1$, is called the threshold.

The computational time required for this approach can be bounded by $\mathcal{O}(un)$ where u depends on t and the problem instance and is usually a lot smaller than n for reasonable values of t. By calculating the difference of the costs of two solutions in the 2.5-opt neighbourhood an important speedup can be achieved. The computational time in this case decreases to $\mathcal{O}(u^2)$. The resulting local search algorithm is called *2.5-opt-threshold*.

3.7 2.5-Opt-Combined

Since the local search algorithms *2.5-opt-depth*, *2.5-opt-threshold* and *2.5-opt-sampling* use different perturbed variants of the exact solution cost function, we tried to combine these algorithms. Especially the following combination of *2.5-opt-threshold* and *2.5-opt-sampling* turned out to be useful.

Algorithm 2. *Combined Local Search Algorithm for the PTSP*

1. *Create an initial solution S using the nearest neighbour heuristic*
2. *Repeat i times:*
   Use 2.5-opt-threshold *with S as the initial solution, store the result in S*
   Use 2.5-opt-sampling *with S as the initial solution, store the result in S*
3. *Return S*

We will refer to this combined local search approach with *2.5-opt-combined*.

4 Experiments and Results

We tested different parameterizations of the algorithms on instances from the TSPLIB benchmark [10] supplemented with the probabilities for the customers (tsplib instances), on Euclidean instances in which customers are distributed uniformly at random in a square (uniform instances) and Euclidean instances in which customers are distributed around a certain number of centers which themselves are distributed uniformly at random in a square (clustered instances). We either have selected the same probability for each city with typical values of $0.05, 0.1, 0.2, \ldots, 0.5$ or we have selected the probabilities uniformly from a fixed interval. For each algorithm and each parameterization of the algorithm we performed 50 independent runs and calculated the average runtime and the average solution quality. Complete numerical results of our experiments will be soon available at [1].

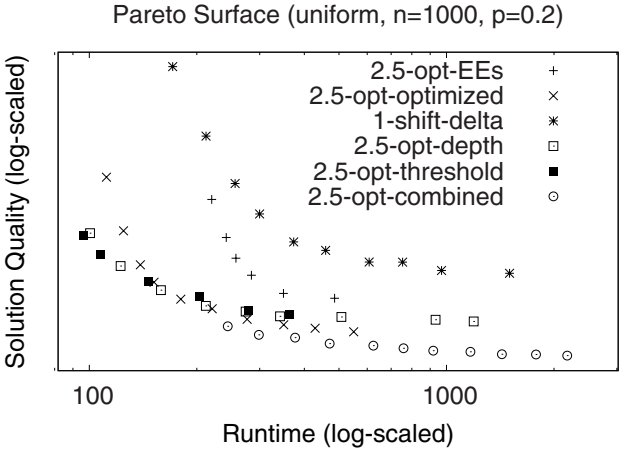

Fig. 1. Pareto Surface for different parameterizations of the algorithms *2.5-opt-EEs*, *2.5-opt-optimized*, *1-shift-delta*, *2.5-opt-depth*, *2.5-opt-threshold* and *2.5-opt-combined* on uniform instances with $n = 1000$ and homogeneous probabilities of $p = 0.2$. Since the PTSP is a minimization problem, better solutions are plotted with a lower solution quality.

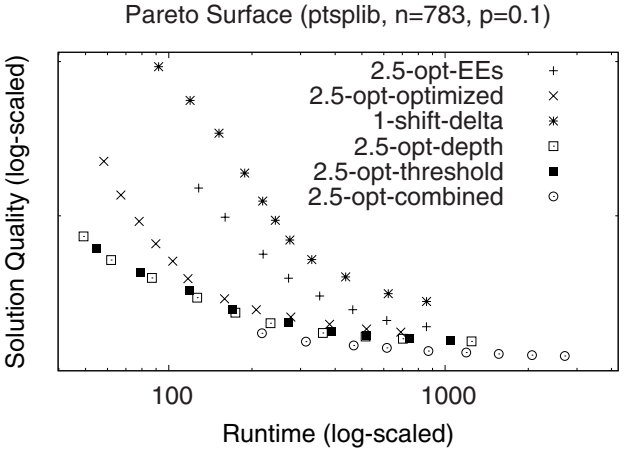

Fig. 2. Pareto Surface for different parameterizations of the algorithms *2.5-opt-EEs*, *2.5-opt-optimized*, *1-shift-delta*, *2.5-opt-depth*, *2.5-opt-threshold* and *2.5-opt-combined* on a 783-city TSPLIB instance with homogeneous probabilities of $p = 0.1$. Since the PTSP is a minimization problem, better solutions are plotted with a lower solution quality.

Since heuristics for the PTSP should both be fast and produce good solutions, the development of these heuristics can be seen as a multi-objective optimization problem. That means the goal here is to find the so called Pareto front of non-dominated algorithms. The practitioner then can pick the proper algorithm concerning the required solution quality or the available runtime. Therefore we used runtime/solution quality graphs for the visualization of the results.

In our experiments we found out that the algorithms *2.5-opt-optimized*, *2.5-opt-depth*, *2.5-opt-threshold* and *2.5-opt-combined* always dominate *2.5-opt-EEs* which itself dominates *1-shift-delta* for most of the problem instances. In all our tests *2.5-opt-combined* is part of the Pareto front on its whole runtime range, whereas for lower runtimes the situation is not clearly decided between *2.5-opt-optimized*, *2.5-opt-depth* and *2.5-opt-threshold*.

Figure 1 illustrates the results for a uniform instance with $n = 1000$ customers and homogeneous probabilities of $p = 0.2$ and figure 2 illustrates the results for a 783-city TSPLIB instance with homogeneous probabilities of $p = 0.1$. These results are typical for our experiments and representative for the results mentioned above.

5 Conclusions and Outlook

In this work we have presented new sampling- and ad-hoc approximation-based local search algorithms for the PTSP with significant improvements over a state-of-the-art local search algorithm. These improvements could be achieved on the

basis of the following three ideas: Precomputation combined with more elaborate data structures for the sampling-based algorithms, the use of ad-hoc approximation and the alternation of local search with sampling and ad-hoc approximation.

The new algorithms now build the Pareto front for small runtimes (i.e. at most some seconds) and in principle they could also be used in iterated local search algorithms or hybrid heuristics to create new state of the art algorithms in higher runtime regions. Ant Colony Optimization combined with a local search algorithm is a widely used hybrid heuristic for the PTSP, eg. in [5], [6] and [2]. It seems very promising to use the new local search algorithms within an Ant Colony Optimization algorithm.

Acknowledgements. Dennis Weyland admits support from the Swiss National Science Foundation, grant 36RBVRPSAM.

References

1. Complete Numerical Results, `http://www.idsia.ch/~weyland`
2. Balaprakash, P., Birattari, M., Stützle, T., Dorigo, M.: Adaptive sample size and importance sampling in estimation-based local search for stochastic combinatorial optimization: A complete analysis. Technical Report TR/IRIDIA/2007-015, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (September 2007)
3. Balaprakash, P., Birattari, M., Stützle, T., Yuan, Z., Dorigo, M.: Estimation-based ant colony optimization and local search for the probabilistic traveling salesman problem. Technical report, Brussels, Belgium (September 2008)
4. Bianchi, L., Dorigo, M., Gambardella, L.M., Gutjahr, W.J.: A survey on metaheuristics for stochastic combinatorial optimization problems. Accepted for publication at Natural Computing (2008), doi. 10.1007/s11047-008-9098-4
5. Bianchi, L., Gambardella, L.M., Dorigo, M.: An ant colony optimization approach to the probabilistic traveling salesman problem. In: PPSN VII: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature, pp. 883–892. Springer, Heidelberg (2002)
6. Bianchi, L., Gambardella, L.M., Dorigo, M.: Solving the homogeneous probabilistic traveling salesman problem by the aco metaheuristic. In: ANTS 2002, pp. 176–187. Springer, Heidelberg (2002)
7. Birattari, M., Balaprakash, P., Stützle, T., Dorigo, M.: Estimation-based local search for stochastic combinatorial optimization. Technical Report TR/IRIDIA/2007-003, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (February 2007)
8. Jaillet, P.: Probabilistic Traveling Salesman Problems. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA (1985)
9. Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: A case study. In: Aarts, E.H.L., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization, pp. 215–310. Wiley, Chichester (1997)
10. TSPLIB, `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`