Efficient Path Planning Involving Equivalent Places

Sebastian Feld, Martin Werner, Florian Dorfmeister
Institute for Computer Science
Ludwig-Maximilians-University Munich
Munich, Germany
{forename.surname}@ifi.lmu.de

***Abstract*— This paper investigates a complication of the classical Traveling Salesman Problem, which arises due to multiple points with functional equivalence. While the classical Traveling Salesman Problem tries to find a shortest tour visiting each point in a set exactly once, we consider the challenge of finding the shortest tour visiting each point of a given set exactly once and exactly one additional point out of a different set of equivalent places. The insertion of a gas station to a traveling salesman problem provides an example. With this paper, we investigate and analyze different algorithms to solve this complex problem by reducing it to the solution of a set of classic Traveling Salesman Problems. Furthermore, we provide an efficient way to trade off between the size of the set of traditional problems still to be solved and the expected error of the algorithm.**

I. INTRODUCTION

The Traveling Salesman Problem (TSP) might be the best studied problem in combinatorial optimization. The main problem consists of a traveling salesman finding the shortest sequence of ways to reach each city of a given set of cities exactly once. Many applications lead to Euclidian instances, where the vertices are a subset of $\mathbb{R}^n$ and the length of the edges is given by the norm of $\mathbb{R}^n$. Interest in the Traveling Salesman Problem arises from several areas of research: First of all, it is an NP-hard problem [9]. Hence, finding a polynomial time solution to this problem is of great general interest showing $P = NP$, even though few actually believe in the possibility of such a solution. A well-known approximation for the Euclidian TSP (actually a metric TSP would suffice for this algorithm) is Christofides' Algorithm with $\mathcal{O}(n^3)$ running time and an error bound of $3/2$ [4].

Application areas where Traveling Salesman Problems arise are vast. For example, efficient routing of a robot arm drilling a set of holes is a commonly cited instance, task planning and navigation are other domains.

Additionally, several constrained or otherwise more complex variants to the basic problem arise from applications. For example, a partition of a set of vertices into several convex neighborhoods results in the problem of finding a shortest tour visiting one arbitrary vertex from each neighborhood exactly once [7]. Some further algorithms to this problem can be found in [6], [5].

The variant we consider in this paper arises from a large-scale indoor navigation system at Munich airport [10], where the set of points of interest is created from a video conference between a user and a callcenter, and many categories of equivalent places exist (toilets, shops, restaurants, ...). This problem was solved by a genetic algorithm considering only the complexities arising from the indoor setting [11]. The question of how to select the right place to insert into a tour from a set of equivalent places still remained open.

Additional to the given scenario, where our main interest into the problem originated from, several other scenarios of application are possible:

Nowadays, for electro mobility applications the charging stations are still rare and the range of electric cars is small. Hence, it can be of great interest to find the best refilling possibiltiy for a given tour visiting several places. This scenario might also add constraints to the original problem such as that the refilling takes a given amount of time and has to fit to the overall plan (for example rests).

Another application arises from mobile robotics. Assume a set of mobile robots, which have a specific sensing task given by a set of places, where the sensor readings have to be taken. Assume furthermore, that the mobile robots have the possibility to reload their batteries. This might be at specific fixed equivalent places (charging station, wind, solar energy). In addition, these robots might be able to interact with each other giving a new (dynamic) set of equivalent places of possible interaction. This might be useful, when a distributed consensus algorithm is used needing a specific amount and type of interaction.

With this paper we construct an algorithm, which is able to find exactly one equivalent point to insert into a tour connecting a set of vertices $V \subseteq \mathbb{R}^2$.

The rest of this paper is organized as follows. In the following Section we formally introduce the problem and some notation. In Section III we discuss the problem and give some baseline algorithms for solving it. Section III-D describes the main algorithm of this paper. Section IV evaluates the algorithm on different types of datasets. Section V concludes the paper and gives hints on future work.

II. PROBLEM STATEMENT

Let G be a complete, Euclidian, undirected graph with vertices $V \subset \mathbb{R}^2$. A subset $P \subseteq V$ defines the points of interest and another disjoint subset $Q \subseteq V$ defines the equivalent places. A tour is given as an ordered sequence of vertices from V containing exactly one vertex from Q and all vertices from P exactly once. The length of such a tour

$T = (v_i)_{i=0\dots k}$ is given by

$$\operatorname{len}(T) = \operatorname{len}\left((v_i)_{i=0\dots k}\right) = \sum_{i=0}^{k} ||v_{i+1} - v_i||,$$

where $v_{k+1} = v_0$ for the last summand representing the way back. The problem is now to find the tour with shortest length.

III. Algorithmic Ideas

A. Exhaustive Search

The most basic idea to solve the given problem is the following: For each $q \in Q$ we construct $G_q = P \cup \{q\}$ and solve this TSP to the optimal solution $\operatorname{Opt}(G_q)$. From the set of solutions $\{\operatorname{Opt}(G_q)_{q\in Q}\}$, we select the best one $\min_{q\in Q}(\operatorname{len}(\operatorname{Opt}(G_q)))$. This algorithm is simple and correct. It always finds the best solution. The main drawback of this algorithm is the exhaustive running time in cases where many equivalent places are available. Assuming the solution of a TSP with a fixed number $k+1$ of vertices (k vertices from P and one vertex from Q) to take a constant time t_k, we result with a time consumption $|Q|t_k$ increasing linearly with the size of the set of equivalent places Q.

From these considerations the question whether there is some area containing equivalent places which can be left out of calculation arises naturally. Considering only very basic geometric calculations, there are not too many choices. As a first step, we should look for a geometric shape C defined by the set P for which we believe, that many solutions to TSPs lie in. Then we can use this shape to split $\mathbb{R}^2 = C \coprod (\mathbb{R}^2 \setminus C)$ and first try to solve the problem inside the limited region $C \cap Q$. This splitting will of course be used only for the set of equivalent places, as there is no chance to remove vertices from the set P as all points in P have to be part of the solution. The most simple yet efficient idea is to use a circle. All possible tours for the basic TSP given by P remain inside the bounding circle of the set P. We will use exactly such a circle C centered at the centroid of the vertices in P using the minimal radius including all points from P to split the given problem.

As a first consideration, it is easy to see, that there is no area inside the circle, which can be left out without considering the geometry of the set Q. Figure 1 shows a triangle and an arbitrarily positioned equivalent vertex q inside the circle around P. This vertex can, of course, be part of an optimal solution (e.g., when no other vertex is available or other vertices are far away) and can be at any point inside the circle. Hence, each algorithm has to consider the complete set of equivalent places lying inside the constructed circle $C \cap Q$.

B. Outer Circle Algorithm

This consideration can be turned over into a quite effective algorithm, which we will call *Outer Circle*. This algorithm only considers equivalent points, which lie inside the circle $C = \operatorname{Circ}(c, R)$ given by the centroid c of P with radius $R = \max(||p - c||)$. Therefore, we construct the set of problems

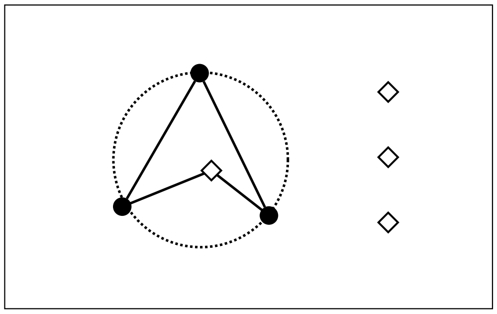

Fig. 1. Example showing an arbitrarily positioned equivalent point ($\Diamond$) being part of the optimal solution. This shows, that no equivalent point inside the circle can be left out of consideration.

$G_q = P \cup \{q\}$ for all equivalent points $q \in Q \cap C$, solve them, and from the set of solutions $\{\operatorname{Opt}(G_q)_{q\in Q}\}$, we again select the best one. This algorithm performs fairly well, as long as at least one equivalent point is inside the circle. In these situations, the algorithm is able to find a tour and this tour is often the overall shortest tour. But in cases where there is no equivalent point inside C, the algorithm will not find a tour. However, even in cases where there are equivalent points inside the circle, the optimal solution is not always found. Figure 2 shows an example, where the *Outer Circle Algorithm* does not find the optimal tour due to a good vertex lying outside the circle, but very near to a vertex from P.

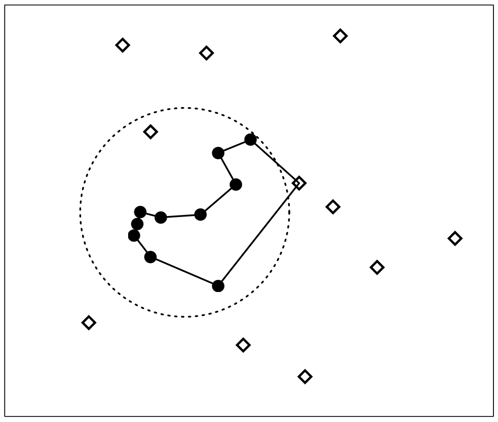

Fig. 2. Equivalent places ($\Diamond$), points of interest ($\bullet$), the circle C and the optimal tour. The *Outer Circle Algorithm* would choose the equivalent place lying inside the circle resulting in a suboptimal solution.

For evaluation purposes, we decided to complete the *Outer Circle Algorithm* to always find a tour with a simple heuristic: If no equivalent point lies inside the circle C, we include the equivalent point q, which is nearest to the set P. In this way, the algorithm always finds a tour.

There are also other possibilities to define geometric splits aside from the given circle. However, we did not find a good tradeoff between complexity of the geometric object and usefulness of the split. One idea was to define an extended neighborhood around the optimal tour in P to efficiently find deviations. However, this split is only useful when the optimal solution including one equivalent point actually is a deviation from the optimal tour in P. In other cases, we are left with a very complicated complement for which the test, whether a point lies inside this complement, is of non-constant complexity (e.g., point in polygon).

C. Extended Circles

Motivated from Figure 2 showing that the optimal solution is not always lying inside the centroid circle C, the idea of enlarging the circle quickly arises. We defined two enlarged circle algorithms. The basic idea is the same: We split the space into a circle and its complement and complete the algorithm for cases, where the (enlarged) circle does not contain equivalent points. Then we only solve the problem by exhaustive search inside this circle.

We called the first variant *Doubled Outer Circle* and it simply enlarges the circle to twice the size of the choice from the *Outer Circle* algorithm. This algorithm finds the optimal tour in much more cases. However, this also increases the number of equivalent points $Q \cap \mathrm{Circ}(c, 2R)$ resulting in a possible waste of computation time in cases, where the optimal solution already resides inside a smaller circle.

Another idea is to incorporate the distribution of equivalent points in the choice of enlargement. The basic idea of the *Three Sigma Algorithm* is, that for equivalent points with nearly uniform distributions, the enlargement can be controlled by the standard deviation of the pairwise distances of the vertices in Q. In other words, we construct the set $D = \{\|q_i - q_j\|\}$ of pairwise distances in Q and take the associated standard deviation σ_D and define the splitting circle by $C = \mathrm{Circ}(c, R + 3\sigma_D)$. This leads to a more conservative split than the *Doubled Outer Circle Algorithm*.

D. Ordered Inclusion Algorithm

From these geomtric splits and motivated by the examples in Figure 1 and in Figure 2 the following idea is generated: As we should include the complete circle as defined for the *Outer Circle Algorithm*, we start with this. However, the completion of this algorithm with respect to the problematic cases now works as follows: We order the remaining set $Q \cap (\mathbb{R}^2 \setminus C)$ by the distance from the centroid c and denote this ordering by $(q_k)_{k \in \mathbb{N}}$.

For the cases motivated by Figure 2, where a solution is not found, because a given equivalent vertex is outside C, we solve the ordered sequence of problems $G_k = P \cup \{q_k\}$ until a specific breaking condition is met and use the best tour already found as the result of the *Ordered Inclusion Algorithm*.

For finding an efficient breaking condition, let $T_k = (v_i)_{i=1\ldots n}$ be an optimal tour for the problem G_k. Then we can construct a tour S for G_{k+1} by removing both edges that are connected to q_k and adding two edges connecting the open spots to q_{k+1}, e.g., we perform a specific 2-opt-exchange. This gives an example tour S for the problem G_{k+1} such that $\mathrm{Opt}(G_{k+1}) \leq \mathrm{len}(S)$.

Let R denote the maximal distance between two points in G_k

$$R = \max\{||v - w||, v, w \in G_k\}$$

and r denote the minimal distance between q_{k+1} and P

$$r = \min\{||q_{k+1} - p||, p \in P\}$$

See Figure 3 for an example of constructing S and the definitions of R and r.

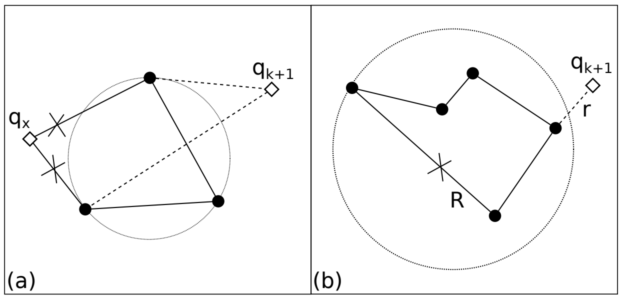

Fig. 3. One two-opt exchange for the construction of the example S and the sizes R and r.

As the tour S is constructed from leaving out two edges and adding two edges, a bound can be given: The tour S is longer than taking out the two longest possible edges from G_k and adding the two shortest edges in G_{k+1}.

$$\mathrm{len}(S) \geq \mathrm{Opt}(G_k) - 2R + 2r$$

Considering the sequence G_k of problems and assuming that from some k on the length of the tours will increase as the cost of connecting the set P with q_k exceeds the post-optimization capabilites due to changing q_k, a breaking condition can be given by $R \leq r$. A geometric motivation is, that the edges in P will not change much, when the distance to the considered points in Q is large enough. Because then, the shortest tour in P will be taken, the edge nearest to the current point q_k will be removed and the tour back and forth to q_k will be added. These considerations are not correct and there exist counterexamples. Figure 4 shows such a counterexample situation.

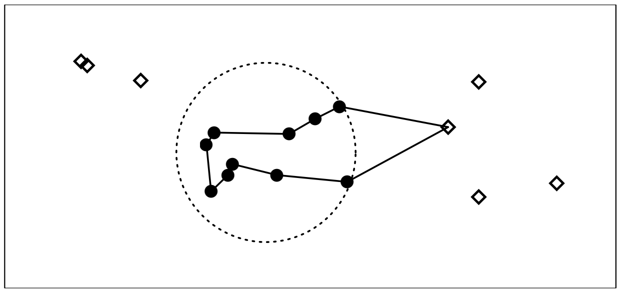

Fig. 4. Example in which the *Ordered Inclusion Algorithm* does not find the optimal tour. It will choose the equivalent point at 10 o'clock instead the one at 3 o'clock resulting in the optimal tour.

Hence, the algorithm need not find the shortest tour. However, as we will show in the evaluation, for many problems this breaking condition gives quite convincing results and outperforms the other algorithms.

To describe the algorithm completely: First calculate the centroid (e.g., mean of the coordinates) of the set P. This step is linear in $|P|$. Then, we sort the set Q increasing with the distance from the centroid. For equal distances, an arbitrary ordering can be used. In these cases, however, all equivalent points of equal distance to the centroid should be tried. This can be performed in $|Q| \log |Q|$ time. For large sets Q, however, a spatial subdivision (e.g., a spatial index tree) should be used, such that the complete set does not have to be sorted. For the resulting sequence $G_0, G_1, \ldots$ of problems $G_k = P \cup \{q_k\}$ we search for the optimal tour $Opt(G_k)$ using an external TSP solving algorithm. This usually takes the largest amount of time of the algorithm. As long as q_k is inside the circle, this solution is triggered always. For q_k lying outside the circle, we use the breaking condition $R \leq r$. The algorithm is given in pseudocode in Algorithm 1. Note, that the choices of R and r are quite conservative and can be made stronger (e.g., choose R from the best known solution and not from the last one). However, this will increase defective breaks for faster expected running time.

Algorithm 1 Ordered Inclusion Algorithm

```
$Q \leftarrow sortByDistance(getCentroid(P), Q)$
**for** $q \in Q$ **do**
    $R \leftarrow getLongestEdgeFromLastTour()$
    $r \leftarrow getMinDistance(q, P)$
    **if** $insideOuterCircle(q, P) \parallel r < R$ **then**
        $T_{tmp} \leftarrow solveTSP(P \cup q)$
        $T \leftarrow (T_{tmp} < T) ? T_{tmp} : T$
    **else**
        $break;$
    **end if**
**end for**
**return** $T$
```

IV. Evaluation

The evaluation of our algorithms takes place through several experiments that can basically be divided into two categories. In the following sections we explain the two categories, describe the metrics to evaluate the algorithms and finally discuss the results. For the solution of the individual Traveling Salesman Problems, we applied a solver based on the Held-Karp bound [8], [1].

A. *Test Setup*

In the first experiment we assume a square area on which a certain amount of uniformly distributed equivalent points are placed (Q). On another square area, which is centred within the area Q, points of interest are also arranged (P) following a uniform random distribution. A test series consists of executing each of the proposed algorithm on 500 random instances of the problem. Each test series is performed with different ratios of areas P/Q. The ratio starts with 5% and is increased with steps of 5% up to 100%.

The assumption, that the amount of equivalent places is larger than the amount of points of interest and may be randomly distributed, is based on the fact that this is also the case in most practical scenarios. As equivalent places are typically places of general interest, they will be spread out uniformly to give comparable service quality to the complete public space. Let the toilets in an airport, the mailboxes in a city or the gas stations of a region serve as examples.

The second experiment refers to the real distribution of gas stations in Germany. The base area of this problem is the territory of Germany, the equivalent places are represented by three percent of the gas stations as extracted from OpenStreetMap [3] and the points of interest are large cities. Two test series will be performed: first, the 20 largest cities in Germany and second, the 20 largest cities in Bavaria (a state covering Southern Germany)[2]. Figure 5 shows the test setup.

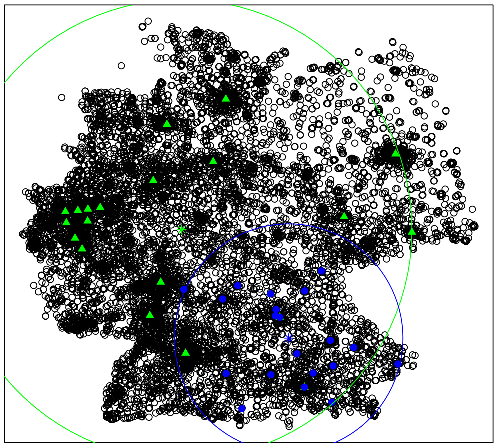

Fig. 5. Gas stations (black ○), the 20 largest cities in Germany (green △), and Bavarias 20 largest cities (blue ●). Furthermore, the circle for the *Outer Circle Algorithm* are given.

In order to evaluate the algorithms we performed three experiments of the randomly generated test setup and two experiments of the gas station test setup. Table I summarizes the characteristics of the experiments.

TABLE I
SUMMARY OF THE EXPERIMENTS

Problem	# P	# Q	# *Iterations*
random_30	10	30	500
random_100	10	100	500
random_500	10	500	500
petrol_germany	20	389	1
petrol_bavaria	20	389	1

B. *Metrics for Evaluation*

This work focusses on the efficient planning of a route through a set of points of interest as well as exactly one equivalent place. This results in two quality characteristics for the algorithms, that will serve as metrics for our evaluation.

The first metric is related to the amount of equivalent places to be examined. The less equivalents an algorithm must consider, the faster a shortest tour visiting the vertices can be found. With the subsequent evaluation we measure the average portion of equivalent points that were not considered by the corresponding algorithm. The higher the value, the better the algorithm's performance. In this setting, the exhaustive search algorithm, in which each individual equivalent is considered, has a zero improvement.

The second metric refers to the quality of the best tour found by the respective algorithm. The smaller the sum of the edge's costs, the better a tour. As the optimal solution is known for all problems in the evaluation by the exhaustive search algorithm, we measure the proportion of the solutions found whose costs are optimal.

C. Results

The results for the first experiment (random type problems) are given as pairs of graphs. The left hand graph always shows the improvement in computational complexity given by the respective algorithm. The algorithm names have been abbreviated as follows:

Outer Circle Algorithm (oc), *Doubled Outer Circle Algorithm (doc)*, *Three Sigma Algorithm (ts)*, and *Ordered Inclusion Algorithm (oi)*.

Figure 6 shows the results for ten points of interest and thirty equivalent points. The x-axis always shows the share of the POI's area in the total area. In general, all algorithms start off with a high improvement for small shares. With increasing share, the size of the circle is increasing and hence the efficiency of the splits is decreasing resulting in a reduced improvement measure.

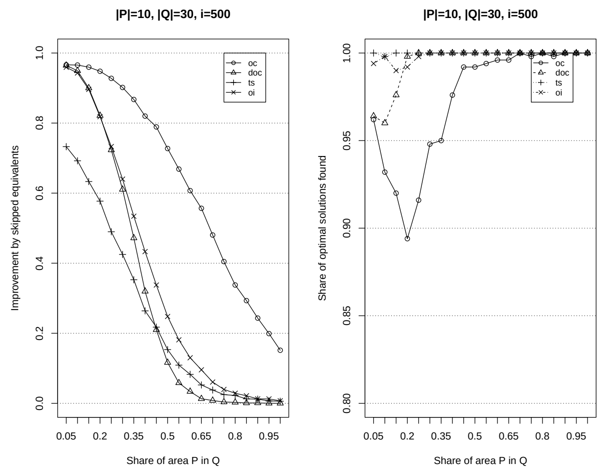

Fig. 6. Reults for a test setup with $|P| = 10, |Q| = 30, i = 500$

As a higher improvement also increases the probability of a wrong shortest tour, the right hand graph always shows the portion of optimal solutions of the different algorithms, again plotted versus the share of the two areas. In general, the quality of the algorithms increases as the portion increases. This is due to the fact, that the splitting circle is getting larger and hence the algorithms converge to an exhaustive search, which is always correct.

The main result from Figure 6 is that the *Outer Circle Algorithm* has the best running time. However, this algorithm does not find the best solutions especially for small shares. Though, in around 90% of the cases, the best solution is found. The second-fastest algorithm is our proposed *Ordered Inclusion Algorithm*. It is clearly slower than the *Outer Circle Algorithm*, as this algorithm constitues the first step of the *Ordered Inclusion Algorithm*. However, the *Ordered Inclusion Algorithm* has a quite acceptable error rate, better than all proposed algorithms except for the *Three Sigma Algorithm*. Unfortunately, the *Three Sigma Algorithm* has only a bad improvement with respect to running time and incorporates a complex initialization step calculating the standard deviation of all pairs of vertices in Q.

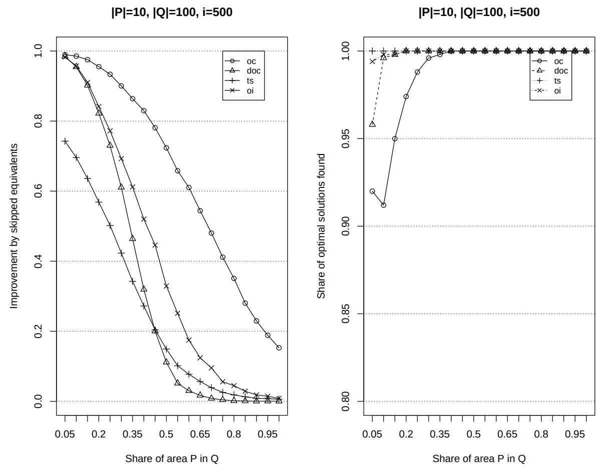

Fig. 7. Reults for a test setup with $|P| = 10, |Q| = 100, i = 500$

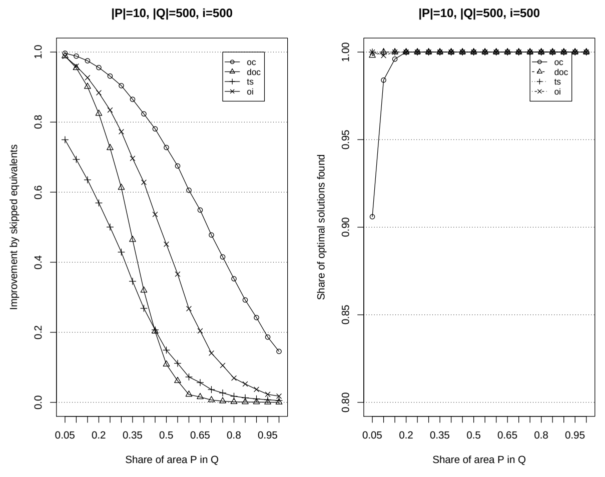

Fig. 8. Reults for a test setup with $|P| = 10, |Q| = 500, i = 500$

Figures 7 and 8 show the same evaluation each time with a larger amount of equivalent points. The results are basically comparable to Figure 6, but all algorithms converge faster to a 100% success rate. This is to be expected, as more equivalent places simplify the problem in general.

From this random setup, we conclude, that the *Ordered Inclusion Algorithm* provides the best overall performance in terms of a good improvement with respect to computation time together with a high probability of success.

TABLE II
IMPROVEMENT OF THE ALGORITHMS APPLIED TO THE GAS STATION SITUATION

Algorithm	Germany 3%	Bavaria 3%
Exhaustive Search	389 (impr=0.00%)	389 (impr=0.00%)
Outer Circle	367 (impr=5.66%)	99 (impr=74.55%)
Doubled Outer Circle	389 (impr=0.00%)	292 (impr=24.94%)
Three Sigma	389 (impr=0.00%)	267 (impr=31.36%)
Ordered Inclusion	389 (impr=0.00%)	119 (impr=69.41%)

Table II shows the results of a realworld problem situation motivated by electric mobility. Assuming that 3% of the gas stations provide electric recharging services, we solved the problem of visiting the twenty largest cities in Germany and in Bavaria, respectively, including a visit with one of the three percent of existing gas stations. In this scenario, the best tour was found by all algorithms. For the scenario including complete Germany, only the *Outer Circle Algorithm* provides an improvement. Figure 5 shows the reason. The circle defining the possible cut contains most gas stations and especially all twenty largest cities. Hence, in most cases nearly all gas stations have had to be considered. For the case of Bavaria, in which the set of points of interest covers a significantly smaller area compared to the area covered by gas stations, the improvements are in general higher. The best result was given by the *Outer Circle Algorithm*. This is a lucky case, because the *Outer Circle Algorithm* actually found the best solution for this problem with an improvement of nearly 75%. However, the significantly more reliable *Ordered Inclusion Algorithm* also provides a significant improvement of 69%.

As a summary, we propose to use the *Ordered Inclusion Algorithm* in all cases. This algorithm provides considerable reduction of computational complexity while providing very low error probabilities assuming a uniform distribution of equivalent places.

V. CONCLUSION AND FUTURE WORK

This paper provides an algorithm reducing the problem of efficient path planning involving equivalent places to a set of path planning problems without special places, which can be solved by various techniques from the literature.

This algorithm showed strength in cases where the points of interest cover a small area compared to the available equivalent places and where the equivalent places are well-distributed. This case arise from application scenarios very often. Equivalent places in applications can be places where a specific service is provided. As this service might have to be provided everywhere with the same service quality, a good distribution (approximately a uniform one) is to be expected. Examples are recharging stations for mobile robots, Internet access facilities, toilets in indoor navigation scenarios or gas stations outside.

Future work should consider other classes of problems. For example, many problems might not be Euclidian or might be constrained, for example, by a minimal (resp. maximal) distance to be traveled before or after visiting the equivalent place or a precedence relation. In these cases, the geometric definition of the splitting is questionable or even impossible. Another interesting question, though not yet well-motivated by the given examples, is the case where more than one equivalent point comes into a planning problem and where direct edges between vertices from Q can be considered. Another situation of interest is the case where different categories of equivalent places have to be considered. This translates to a collection of sets Q_k and problems such as to include at least one (resp. exactly one) point from each (resp. only one) of these sets.

REFERENCES

[1] Concorde TSP Solver. Online, 2011. http://www.tsp.gatech.edu/concorde.html.

[2] Largest Cities in Germany and Bavaria. Online, 2013. http://en.wikipedia.org/wiki/Bavaria and http://en.wikipedia.org/wiki/Germany.

[3] OpenStreetMap. Online, 2013. http://openstreetmap.org/.

[4] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. *Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh*, 1976.

[5] Mark de Berg, Joachim Gudmundsson, Matthew J. Katz, Christos Levcopoulos, Mark H. Overmars, and Frank A. van der Stappen. Tsp with neighborhoods of varying size. *Journal of Algorithms*, 57(1):22–36, 2005.

[6] Adrian Dumitrescu and Joseph S. B. Mitchell. Approximation algorithms for tsp with neighborhoods in the plane. *Journal of Algorithms*, 48(1):135–159, 2003.

[7] Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, 2004.

[8] Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[9] Christos H. Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science*, 4(3):237–244, 1977.

[10] Peter Ruppel, Florian Gschwandtner, Corina Kim Schindhelm, and Claudia Linnhoff-Popien. Indoor navigation on distributed stationary display systems. *International Computer Software and Applications Conference*, pages 37–44, 2009.

[11] Martin Werner. Selection and ordering of points-of-interest in large-scale indoor navigation systems. In *IEEE 35th Annual Computer Software and Applications Conference (COMPSAC), 2011*, pages 504 –509, 2011.