On the Travelling Salesman Problem with Neighborhoods in a Polygonal World

Miroslav Kulich[1], Jan Vidašič[2], and Jan Mikula[1,2]

[1] Czech Institute of Informatics, Robotics, and Cybernetics,
Czech Technical University in Prague, Prague, Czech Republic
{miroslav.kulich,jan.mikula}@cvut.cz
web: http://imr.ciirc.cvut.cz
[2] Faculty of Electrical Engineering,
Czech Technical University in Prague, Prague, Czech Republic
vidasic.jan@gmail.com

Abstract. The Travelling Salesman Problem with Neighborhoods (TSPN), as an extension of the broadly studied Travelling Salesman Problem, has many practical applications in robotics. Although several approaches to the problem have been introduced, they assume an environment without obstacles. In this paper, we introduce a method for the TSPN with polygonal neighborhoods taking polygonal obstacles into account. The method splits the problem into two subproblems: the Generalized Travelling Salesman Problem and the Touring Polygons Problem, which are solved sequentially. While the mature metaheuristic called Generalize Large Neighborhoods Search is employed to solve the first subproblem, the second one is solved by modifying the rubber-band algorithm. The experimental results show that the proposed approach outperforms a state-of-the-art algorithm modified for the environment with obstacles by 10-20% in most cases.

Keywords: Travelling Salesman Problem with Neighborhoods, Touring Polygons Problem, combinatorial optimization, path planning

1 Introduction

Assume a salesman that wants to meet several customers. Each customer specifies a compact set in a plane, his neighborhood, where he is willing to meet. These sets can be of arbitrary shape, for example, a disk or a polygon, and they are also allowed to intersect. The salesman aims to find a tour of the shortest length that intersects the neighborhoods of all the customers and returns to the initial point of departure.

This problem was first formulated as the Travelling Salesman Problem with Neighborhoods (TSPN) by Arkin and Hassin [4]. The TSPN is NP-hard as it extends the Travelling Salesman Problem (TSP), and it has recently received increasing attention in robotics and logistics. Example scenarios include sequencing of effective movements for robotic welding or cutting [2,1], vision-based inspection [7], or data collection in distributed wireless sensor networks [17].

Arkin and Hassin [4] studied the TSPN for various neighborhood types, including parallel unit segments, translates of a polygonal region, and circles. They derived lower bounds on the optimal tour lengths and presented simple heuristic procedures for tour construction. Gentilini et al. [7] formulated the problem as a non-convex Mixed-Integer Nonlinear Program (MINLP). They proposed a fast heuristics employing a Branch-and-Bound algorithm for convex MINLP to generate an initial good-quality solution. Furthermore, they devised an approach based on the global MINLP solver Couenne solving the instances with several neighborhoods up to 15 to optimality.

Several researchers take advantage of the availability of advanced solvers for similar optimization problems and use them for the TSPN. Mennel [10] suggests representing each neighborhood by a single point and finding the shortest tour visiting these points by a TSP solver. Optimization of the resulting tour with a fixed order of neighborhoods is consequently done by formulating it as the Touring Polygons Problem (TPP) [6]. TPP is an NP-hard problem of finding a point in each neighborhood such that the tour visiting all such points in the defined order is the shortest possible. Another approach splitting TSPN into subproblems was introduced in [1], where the insertion heuristic was applied to TSP and the rubber-band algorithm to TPP. The key distinction from the former approaches is that the two problems were solved simultaneously. Finally, the combination of 3-opt and the rubber-band algorithm was presented in [2].

The paper's contribution is twofold. First, we propose a decoupled approach for the TSPN consisting of three steps. Individual neighborhoods are sampled first, so each neighborhood is represented by a set of sample points. A sequence in which the neighborhoods are visited together with initial visiting points of individual neighborhoods is found as a solution to the Generalized Travelling Salesman Problem (GTSP) [16]. This sequence is subsequently refined by solving the TPP. Second, we propose a modification of the TPP, which takes polygonal obstacles in the environment into account and thus allows solving the TSPN in a polygonal world.

The rest of the paper is organized as follows. The problem is defined in Section 2, the proposed approach is introduced in Section 3, and the experimental results are presented in Section 4. Finally, Section 5 is dedicated to concluding remarks and future directions.

2 Problem Definition

Assume a set of disjoint polygons $O = \{O_i\}_{i=1}^{m}$ in plane representing obstacles and a set of possibly intersecting polygons $P = \{P_i\}_{i=1}^{n}$ representing neighborhoods. A neighborhood can touch an arbitrary obstacle (i.e., they have a common point(s) or line(s)) but do not intersect (do not share any area). The goal is to find a minimal-length cyclic tour $T = p_1, p_2, \dots p_{n+1}$ such that:

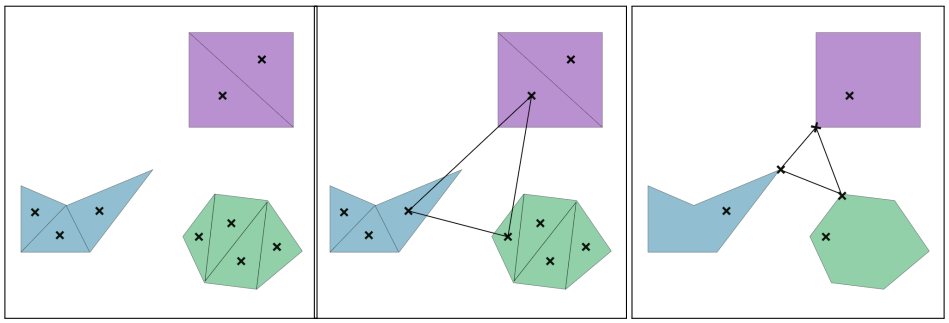
- $p_1 = p_{n+1}$
- For every neighborhood P_k, there exists a point $p_l \in T$ that lies in the neighborhood: $p_l \in P_k$.

– T is collision-free, i.e. it does not intersect with any obstacle.

The goal is thus to find the order in which the polygonal neighborhoods are visited together with points in these neighborhoods, which guarantee to visit the neighborhoods in an optimal way.

3 Approach

The proposed approach works in three phases, as illustrated in Fig. 1. The first phase starts by creating a triangular mesh of each polygon and pre-computing all-to-all distances between the center points of the triangles (Fig. 1a). The GLNS metaheuristic is employed on these points in the second phase (Fig. 1b) The solution that GLNS returns is a sequence of individual points, each representing a polygon of the instance, but this solution is generally far from the optimum for TSPN. Though, it gives a good approximation of the optimal order of the polygons in the tour. The third phase subsequently uses this order and employs the modified TPP algorithm, giving the output points and the order of polygons from the first phase as the input. The TPP algorithm's output forms the final solution, as shown in Fig. 1c. A triangular mesh is generated by the Delaunay refinement algorithm [15]. The rest of the algorithm is detailed in the following paragraphs.



(a) (b) (c)

Fig. 1: General steps of GLNS-TPP. (a) The triangular mesh is created from the input polygons, the points in their centers are computed, and the distances between them are determined. (b) GLNS finds the approximate shortest GTSP tour among the sets (polygons) and their points. (c) TPP takes the tour found by GLNS and tries to optimize its points.

3.1 Generalized Large Neighborhood Search

The Generalized Large Neighborhood Search (GLNS) is a solver for the exactly one-in-a-set Generalized Travelling Salesman Problem (GTSP) [16]. It belongs

to the state-of-the-art heuristics for solving the GTSP together with the GLKH [9], and GK [8] solvers. Experimental results show that it is competitive with the best-known algorithms solving GTSP, often finding better solutions given the same amount of time.

The general framework under which the GLNS operates is called Adaptive Large Neighborhood Search (ALNS) [13,12]. The main idea of the ALNS and, therefore, the GLNS is simple. An initial solution is found first by, e.g., a greedy approach, which is iteratively destroyed and repaired. It is accepted if a better solution than the current best one is found. The search is finished when a terminating condition is met. ALNS uses two sets of heuristics for removals and insertions of the solution. Each heuristic holds a weight representing how successful the heuristic is in improving the solution. This weight is updated as the algorithm runs; thus, the word adaptive in the ALNS. The more successful heuristics are selected more often than the less successful ones. GLNS also regularly optimizes the solution by local search.

The following section gives a simplified description of the original GLNS algorithm. More information can be found in [16].

Algorithm The GLNS algorithm, shown in pseudocode in Algorithm 1 works in iterations. Each iteration i starts by constructing a random initial tour T (line 2) and making it a baseline best tour $T_{best,i}$ for this iteration (line 3). After that, the following cycle is repeated (line 4) until the stopping criterion is met. A single removal heuristic R and insertion heuristic I are selected first, according to their current weights (line 5). Next, a number N_r is uniformly randomly selected from between $1, \ldots, N_{max}$ (where N_{max} is a parameter of the algorithm), marking a number of vertices that are going to be removed (line 6). After creating a copy T_{new} of the current tour (line 7), we remove N_r vertices from T_{new} using the selected removal heuristic R (line 8). Then, using the selected insertion heuristic I, we insert a vertex into T_{new} for each removed vertex so that the tour T_{new} revisits all sets (line 9). The tour T_{new} is then locally optimized using ReOpt and MoveOpt heuristics. These two heuristics try to re-optimize the order of the sets and the vertex in each set (line 10). If the length of the tour T_{new} is smaller than the current best tour of the iteration $T_{best,i}$, then T_{new} becomes $T_{best,i}$ (line 11). The tour T_{new} is accepted or declined as a new tour T for this iteration based on the standard simulated annealing criterion (line 14). The stopping criteria of each iteration have two phases - initial descent and several warm restarts (line 18). The first phase, the initial descent, terminates after a fixed amount of non-improving iterations. Each warm restart starts with the best solution found in the second phase but a lower simulated annealing temperature. Each warm restart ends after several non-improving iterations.

Insertion Heuristics The four insertion heuristics are used in GLNS – nearest, farthest, random and cheapest insertion. Each of these heuristics takes a GTSP instance (G, P_V) and a partial tour $T = (V_T, E_T)$ of graph $G = (V, E, w)$ as an input. $P_V = \{V_1, \ldots, V_m\}$ is a partition of V. The partial tour T is a cycle in G

Algorithm 1: GLNS(G,P_v)

```
Data: A GTSP instance (G, P_v)
Result: A GTSP tour on G
for i = 1 to num_trials do
    T ← initial_tour(G, P_v);
    T_best,i ← T;
    repeat
        Select a removal heuristic R and insertion heuristic I using the
         selection weights;
        Select the number of vertices to remove, N_r, uniformly randomly from
         1, ..., N_max;
        Create a copy of T called T_new;
        Remove N_r vertices from T_new using R;
        For each of the N_r sets not visited by T_new, insert a vertex into T_new
         using I;
        Locally re-optimize T_new;
        if w(T_new) < w(T_best,i) then
            T_best,i ← T_new;
        end
        if accept(T_new, T) then
            T ← T_new;
            Record improvement made by R and I;
        end
    until stop criterion is met;
    Update selection weights based on improvements of each heuristic over
     trial;
end
```

such that each set in the partition P_V is visited at most once. The sets that are visited by the partial tour T are denoted $P_T \subseteq P_V$. The insertion heuristic then takes this partial tour T, inserts a vertex from an unvisited set into it (according to the heuristic-specific rules), and returns a tour that visits one more set than the input tour.

The heuristics use the set-vertex distance for their decision making about which set to insert (V_i in $P_V \backslash P_T$). The set-vertex distance is computed for each set V_i, $i \in \{1, \ldots, m\}$ and vertex $u \in V \backslash V_i$ at the beginning of the algorithm and is defined as:

$$dist(V_i, u) = min_{v \in V_i}\{min\{w(v, u), w(u, v)\}\}$$

Each heuristic has a specific way it selects the set to visit according to this distance. `Nearest Insertion` selects the set V_{opt}, which contains a vertex v at a minimum distance to any vertex on the partial tour T.

$$V_{opt} = \underset{V_i \in P_V \backslash P_T}{\operatorname{argmin}} \min_{v \in V_T} dist(V_i, v)$$

`Farthest Insertion` selects the set V_i, that contains a vertex v which is at a maximum distance to any vertex on the partial tour T.

$$V_{opt} = \underset{V_i \in P_V \setminus P_T}{\operatorname{argmax}} \min_{v \in V_T} dist(V_i, v)$$

`Random Insertion` selects the set V_i uniformly randomly from $P_V \setminus P_T$.

`Cheapest Insertion` selects the set V_i that contains a vertex v that minimizes the insertion cost.

$$V_{opt} = \underset{V_i \in P_V \setminus P_T}{\operatorname{argmin}} \min_{(x,y) \in E_T, v \in V_i} \{w(x,v) + w(v,y) - w(x,y)\}$$

Removal heuristics GLNS uses three different removal heuristics – worst, distance, and segment removal. Each of these heuristics removes N_r vertices from a tour $T = (V_T, E_T)$ using a heuristic-specific strategy.

The goal of the `Worst Removal` heuristic is to remove the vertex that results in the most significant reduction in the tour length. Given a partial tour T we remove the vertex v_j that maximizes the removal cost

$$r_j = w(v_{j-1}, v_j) + w(v_j, v_{j+1}) - w(v_{j-1}, v_{j+1}).$$

`Distance Removal` aims to remove vertices that are close to each other. Given a complete tour T a random vertex is removed from T and added to a set $V_{removed}$. Then an iteration follows, where a seed vertex v_{seed} is uniformly randomly selected from $V_{removed}$ and for each $v_j \in V_T$ the removal cost r_j is computed as

$$r_j = min\{w(v_{seed}, v_j), w(v_j, v_{seed})\}.$$

The vertex with minimal r_j is removed. This process is repeated until N_r number of vertices is removed.

`Segment Removal` removes a continuous segment of the size N_r from the tour T. Given a complete tour T with vertices $V_T = \{v_1, \ldots, v_m\}$, a vertex v_j is uniformly randomly selected and then the vertices $v_j, v_{j+1}, \ldots, v_{j+N_r-1}$ are removed from the tour T.

Local optimizations Before evaluating the acceptance condition in an iteration of GLNS, the new tour is locally optimized. Two techniques are used for this optimization and they are called `ReOpt` and `MoveOpt`.

The goal of `ReOpt` is to optimize the choice of a vertex in each set, keeping the set order fixed. This goal is achieved by creating a directed acyclic graph (DAG) containing all vertices of the GTSP graph. The shortest path between each vertex of the first set of the tour and its copy in DAG is then determined, and the minimal shortest path becomes the optimized tour.

`MoveOpt` is a particular case of GLNS where only one set is removed and reinserted with minimum insertion cost. Given a complete tour $T = (V_T, E_T)$, a random vertex v_j is selected and removed from the tour. A vertex from the same

set V_j with a lowest insertion cost $w(x,u) + w(u,y) - w(x,y)$ for all $u \in V_j$ and all $(x,y) \in E_T$ is reinserted afterwards. MoveOpt is repeated a specified number of times.

3.2 Touring Polygons Problem

The solution route found by the GLSN is refined by solving the Touring Polygons Problem (TPP). The TPP aims to find point p_i in each polygon P_i such that a closed collision-free route τ visiting all such points in a defined order is the shortest possible. We employ a rubber-band algorithm introduced by Pan et al. [11] for pairwise disjoint polygons in a plane. The algorithm takes a route found by the GLNS and a sequence of polygons in the order they are visited in the route and tries to shorten it sequentially. In each iteration, the algorithm goes over all triplets of consecutive points $\langle p_{i-1}, p_i, p_{i+j} \rangle$ and substitutes the middle point $p_i \in P_i$ by the one which minimizes the sum of distances to p_{i-1} and p_{i+1}, i.e.

$$p_i^{new} = \underset{p \in P_i}{\operatorname{argmin}}(dist(p_{i-1}, p) + dist(p, p_{i+1})). \tag{1}$$

The process stops when the change of routes' lengths between two iterations is lower than a specified threshold.

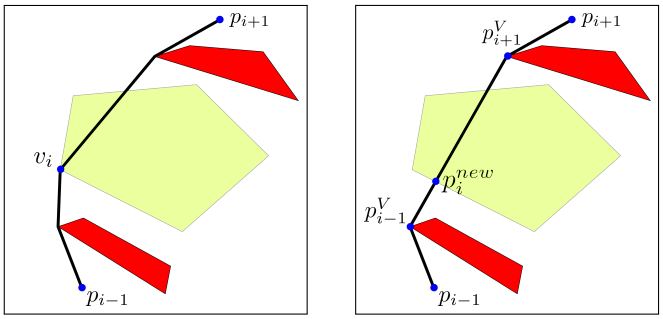
To determine p_i^{new} in Eq. 1, we go over all edges of polygon P_i and compute a point minimizing the sum of the two distances to the edge. p_i^{new} is then the point with the minimal sum:

$$p_i^{new} = \underset{p \in e}{\operatorname{argmin}}(dist(p_{i-1}, p) + dist(p, p_{i+1})) \quad \forall \text{edges } e \text{ of } P_i.$$

Computing p_i^{new} minimizing the sum of distances for a given edge can be realized numerically by Newton's method [14] or analytically using the law of reflection.

3.3 Adaptation to a polygonal world

Several modifications of the described approach are needed to reflect the presence of obstacles in the environment. The modification of the first phase is straightforward. Instead of computing the Euclidean distance between all pairs of center points of the triangles, the visibility graph of the points in the environment with obstacles $\mathcal{W}$ is generated, and the shortest path distances are computed in this graph by the Floyd-Warshall algorithm [5]. Given the all-pairs distances reflecting obstacles, the GLNS algorithm can stay intact. The situation for the TPP is more complex. When looking for the optimal point $p_i \in P_i$ in regards to the points p_{i-1} and p_{i+1} in Eq. 1, it is necessary to take obstacles into consideration. The polygon P_i might be partially or entirely hidden from the view of the points p_{i-1} and p_{i+1}. The idea is to find the vicarious points p_{i-1}^V and p_{i+1}^V on the obstacle(s) for both p_{i-1} and p_{i+1} and then calculating the optimal point p in regards to p_{i-1}^V and p_{i+1}^V instead. The pseudocode of the modified algorithm



(a) (b)

Fig. 2: The process of finding the optimal point in a polygonal world. (a) The shortest polyline connecting a vertex v_i and the points p_{i-1} and p_{i+1} is found. The grey polygons are the obstacles. (b) The vicarious points p^V_{i-1} and p^V_{i+1} are used to find the optimal point p_i^{new}. Obstacles are red, neighborhoods are yellow.

is shown in Algorithm 2, while Fig. 2 illustrates its idea. The algorithm iterates over all vertices $v \in P_i$ and computes the shortest polylines L^{-1} and L^{+1} in $\mathcal{W}$ from v to p_{i-1} and from v to p_{i+1} respectively. The vertex with a minimal sum of the lengths of L^{-1} and L^{+1} is selected, and the points directly before v on the polylines are chosen as the vicarious points. p_i^{new} is then determined using Eq. 1 with the vicarious points instead of p_{i-1} and p_{i+1}.

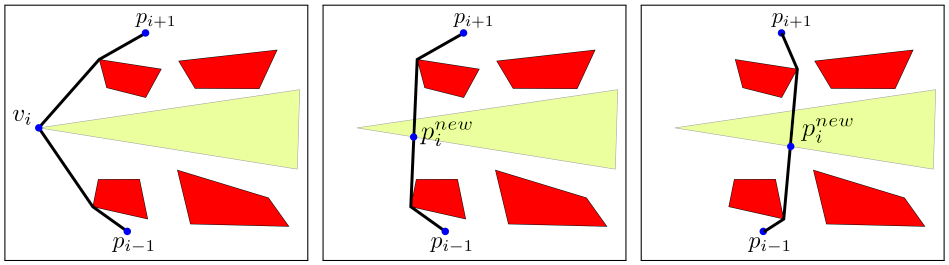
It should be noted that the presented approach is just suboptimal, as shown in Fig. 3. Moreover, another bottleneck is that the approach finds the optimal point on all edges of the input polygon, which leads to calling the time-consuming calculation of the shortest path many times. We thus work with the assumption that the vertex $v^* \in P_i$ minimizing length(L^{-1}) + length(L^{+1}) in Algorithm 2 is

Algorithm 2: Optimal point in a polygon through obstacles

Data: Point p_{i-1}, Point p_{i+1}, Polygon P_i
Result: Point p_i^{new}

```
minLength = infinite;
for vertex v ∈ P_i do
    polyline L^{-1} = shortest polyline from p_{i-1} to v ;
    polyline L^{+1} = shortest polyline from p_{i+1} to v ;
    currentLength = length(L^{-1}) + length(L^{+1});
    if currentLength < minLength then
        p^V_{i-1} = last point before v on the polyline L^{-1};
        p^V_{i+1} = last point before v on the polyline L^{+1};
        minLength = currentLength;
    end
end
p_i^{new} = argmin_{p∈P_i}(dist(p^V_{i-1}, p) + dist(p, p^V_{i+1})) ;
Return p_i^{new};
```

most likely around the area of the optimal point p_i^{new} that is determined at line 12. Instead of iterating over all edges of P_i, we try only the edge preceding and succeeding v^*. Of course, this leads to other inaccuracies in the optimal point determination. However, experimental results show that this suboptimality does not influence the quality of the whole TSPN solver much.



(a) (b) (c)

Fig. 3: Example of a solution that is not optimal found by Algorithm 2. (a) The shortest polyline connecting a vertex v_i and the points p_{i-1} and p_{i+1} is found. (b) Then the vicarious points p_{i-1}^V and p_{i+1}^V are used to find the point p_i^{new}. This solution is not optimal. (c) The correct optimal solution.

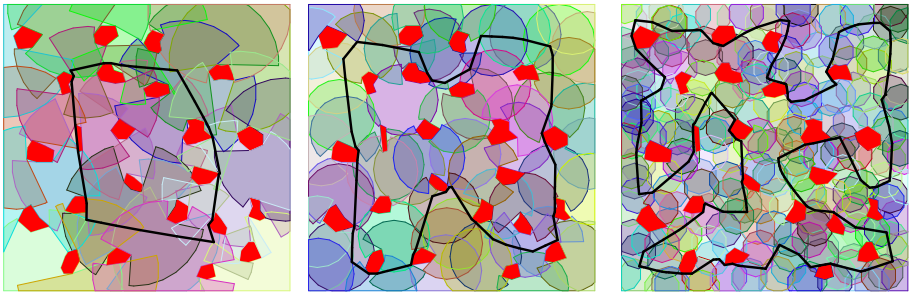
4 Experiments

The performance of the proposed approach in an environment with polygonal obstacles was statistically evaluated and compared with the state-of-the-art TSP-based approach, which solves the TSP on the centroids of the neighborhoods and refines the found route solving the TPP the same way as in the proposed approach. The TSP solver used is the Chained Lin-Kernighan heuristic from the Concorde package [3].

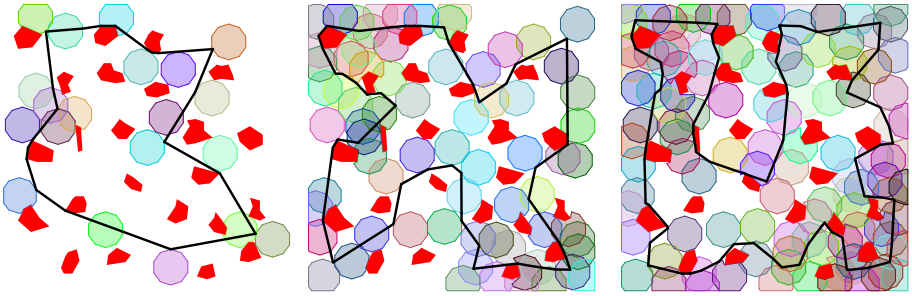
As there is no dataset for TSPN in a polygonal world, two sets of instances were created [1]. The first set A consists of six instances, with the same polygonal map of 59 obstacles and the varying number n of neighborhoods, $n \in \{12, 24, 33, 59, 99, 224\}$. The neighborhoods were generated to almost cover the whole map, so they have varying numbers of vertices and various shapes.

The second set B is based on the same map with 59 obstacles. We generated 150 random points in the map from which we computed visibility regions and cut them by regular decagons centered in these points to simulate a limited visibility range. 150 polygons of a similar size with around ten vertices were generated. We gradually removed polygons from this set of polygons to get instances with $n = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 130, 150\}$ neighborhoods. Examples of the instances with found solutions are shown in Fig. 4.

[1] The generated instances can be found at http://imr.ciirc.cvut.cz/Datasets/TSPN



(a) A24 (b) A59 (c) A224



(d) B20 (e) B70 (f) B110

Fig. 4: Examples of instances and their solutions found by the proposed approach.

All experiments were performed within the same computational environment: a workstation with the Intel®Core™ i7-6770 CPU at 3.4GHz running Ubuntu Linux with the kernel 5.4.0. The algorithms have been implemented in C++ and compiled with "-O2" flag. Fifty runs of both approaches were performed for each instance to provide statistically significant results.

The obtained results are summarized in Table 1, where *name* stands for the problem name, *size* is the number of generated triangles, and thus vertices in the GLNS/TSP, and *min* stands for the cost of the best solution found by the algorithm. Similarly, *mean* is the average cost over all 50 runs for the given problem, and *time* is the average time needed to find a solution. *Ratio* stands for the ratio of the minimal cost of the proposed and TSP-based approach.

We can state that the proposed algorithm finds solutions of better quality than the TSP-based approach for almost all instances. The only exception is the small B-instances where both algorithms found the same or very similar solutions. In the majority of instances, on the other hand, solutions found by the proposed approach are consistently better by $10 - 20\%$.

The main drawback of the GLNS-based approach is its computational time, which is up to four times higher than the TSP-based one. This is caused almost equally by (1) computation of all-pairs distances in the first phase, where the number of vertices is much higher for GTSP than for TSP, and (2) GLNS itself, which is slower than the TSP solver.

Problem		TSP			proposed			Ratio
name	size	min	mean	time [s]	min	mean	time [s]	[%]
A12	988	26.46	26.46	7.359	11.33	11.37	16.569	42.8
A24	826	68.50	68.50	24.032	64.03	64.57	22.977	93.5
A33	857	87.60	87.60	22.931	74.72	75.02	25.527	85.3
A59	1204	133.16	133.16	22.286	105.58	107.26	43.145	79.3
A99	1472	154.73	154.73	28.758	136.67	139.03	83.295	88.3
A224	2096	227.03	227.03	48.817	194.77	200.24	387.651	85.8
B10	81	81.36	81.36	0.752	81.36	81.36	1.219	100.0
B20	177	91.69	91.69	2.775	91.69	91.69	2.830	100.0
B30	265	115.68	115.68	7.584	116.36	116.45	4.032	100.6
B40	359	125.36	125.36	3.970	123.46	123.56	6.753	98.5
B50	456	129.99	129.99	7.778	127.63	127.71	9.474	98.2
B60	543	138.47	138.47	7.617	130.96	131.03	14.513	94.6
B70	633	155.96	155.96	9.511	142.66	142.83	19.174	91.5
B80	724	171.06	171.06	10.291	147.66	148.69	24.727	86.3
B90	814	173.42	173.42	14.575	152.78	156.58	35.954	88.1
B100	901	185.58	185.58	17.596	154.66	155.75	44.165	83.3
B110	1000	190.52	190.52	17.330	163.85	166.82	55.679	86.0
B130	1203	207.67	207.68	20.288	177.68	182.01	82.415	85.6
B150	1403	206.66	206.66	29.073	182.80	186.46	118.986	88.5

Table 1: Comparison of the proposed algorithm with the TSP-based approach.

5 Conclusion

Although the addressed Travelling Salesman Problem with Neighborhoods looks purely theoretically, it received the attention of many roboticists due to its applicability in robotic tasks. We studied the problem variant with polygonal obstacles present in the working environment, further increasing its application potential. We introduced a novel approach to the TSPN combining the solution of two subproblems: the GTSP and the TPP.

As the TPP is widely used in many other approaches incorporated not only in the TSPN, its proposed modification for a polygonal world is not limited to the TSPN only. However, it can also be used as a refinement procedure for similar problems, e.g., Watchman Route Problem or Close-Enough Travelling Salesman Problem. We will thus investigate possibilities of applying the presented principles to solve these problems. As mentioned, the weakest point of the method is its relatively high computational complexity. In future work, we thus want to focus on decreasing it by using advanced methods in the preprocessing phase and by optimizing GLNS.

Acknowledgement

This work was supported by the European Regional Development Fund under the project Robotics for Industry 4.0 (reg. no. CZ.02.1.01/0.0/0.0/15 003/0000470)

and by the Grant Agency of the Czech Technical University in Prague, grant no. SGS21/185/OHK3/3T/37.

References

1. Alatartsev, S., Augustine, M., Ortmeier, F.: Constricting insertion heuristic for traveling salesman problem with neighborhoods. In: Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS-2013). AAAI (2013)
2. Alatartsev, S., Mersheeva, V., Augustine, M., Ortmeier, F.: On optimizing a sequence of robotic tasks. In: Proceedings of the International Conference on Intelligent Robots and Systems (IROS). IEEE (2013)
3. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: Concorde TSP Solver (2010)
4. Arkin, E.M., Hassin, R.: Approximation algorithms for the geometric covering salesman problem. Discrete Appl. Math. **55**(3), 197218 (1994)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Third Edition, 3rd edn. The MIT Press (2009)
6. Dror, M., Efrat, A., Lubiw, A., Mitchell, J.S.B.: Touring a sequence of polygons. In: Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing, STOC '03, p. 473482. Association for Computing Machinery, New York, NY, USA (2003)
7. Gentilini, I., Margot, F., Shimada, K.: The travelling salesman problem with neighbourhoods: Minlp solution. Optimization Methods and Software **28**(2), 364–378 (2013)
8. Gutin, G., Karapetyan, D.: A memetic algorithm for the generalized traveling salesman problem. Natural Computing **9**(1), 47–60 (2010)
9. Helsgaun, K.: Solving the equality generalized traveling salesman problem using the lin–kernighan–helsgaun algorithm. Mathematical Programming Computation **7**(3), 269–287 (2015)
10. Mennell, W.: Heuristics for solving three routing problems: Close-enough traveling salesman problem, close-enough vehicle routing problem, sequence-dependent team orienteering problem. Ph.D. thesis, University of Maryland, USA (2009)
11. Pan, X., Li, F., Klette, R.: Approximate shortest path algorithms for sequences of pairwise disjoint simple polygons. pp. 175–178 (2010)
12. Pisinger, D., Ropke, S.: A general heuristic for vehicle routing problems. Computers & operations research **34**(8), 2403–2435 (2007)
13. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science **40**(4), 455–472 (2006)
14. Ryabenkii, V.S., Tsynkov, S.V.: A theoretical introduction to numerical analysis. Chapman and Hall/CRC (2006)
15. Shewchuk, J.R.: Delaunay refinement algorithms for triangular mesh generation. Computational geometry **22**(1-3), 21–74 (2002)
16. Smith, S.L., Imeson, F.: Glns: An effective large neighborhood search heuristic for the generalized traveling salesman problem. Computers & Operations Research **87**, 1–19 (2017)
17. Yuan, B., Zhang, T.: Towards solving TSPN with arbitrary neighborhoods: A hybrid solution. In: M. Wagner, X. Li, T. Hendtlass (eds.) Artificial Life and Computational Intelligence, pp. 204–215. Springer International Publishing (2017)