The Traveling Visitor Problem and the Koper Algorithm for Solving It

Milan Djordjevic, Andrej Brodnik and Marko Grgurovič

Abstract We consider the problem when visitor wants to visit all interesting sites in a city exactly once and to come back to the hotel. Since, the visitors use streets, walking trails and pedestrian zones, the goal is to minimize the visitor's traveling. A new problem the Traveling Visitor Problem (TVP) is then similar to the Traveling Salesman Problem (TSP) with a difference that the traveling visitors, during its visit of sites, can not fly over buildings in the city, instead visitors have to go around these obstacles. That means that all ”air” distances, like those in a TSP, are impossible in this case. The tested benchmarks are combined from three real instances made using tourist maps of cities of Koper, Belgrade and Venice and two instances of modified cases from TSPLIB. We compared two methods for solving the Traveling Visitor Problem. In all tested cases the Koper Algorithm significantly outperforms the Naïve Algorithm for solving the TVP.

1 Introduction

In the Traveling Salesman Problem (TSP) a set $\{C_1, C_2, ...C_N\}$ of cities is considered and for each pair (C_i, C_j) where $i \neq j$, a distance $d(C_i, C_j)$ is given. The goal is to find a permutation π of the cities that minimizes the quantity

$$\sum_{i=1}^{N-1} d(C_{\pi(i)}, C_{\pi(i+1)}) + d(C_{\pi(N)}, C_{\pi(1)}). \tag{1}$$

This quantity is referred to as the tour length since it is the length of the tour a salesman would make when visiting the cities in the order specified by the permu-

Milan Djordjevic
UP DIST, Koper, Slovenia e-mail: milan.djordjevic@student.upr.si

Andrej Brodnik
UP DIST, Koper, Slovenia e-mail: andrej.brodnik@upr.si

Marko Grgurovič
UP DIST, Koper, Slovenia e-mail: marko.grgurovic@student.upr.si

tation π, returning at the end to the initial city. We will concentrate in this paper on the symmetric TSP (STSP) in which the distances satisfy $d(C_i,C_j) = d(C_j,C_i)$ for $1 \leq i, j \leq N$. While the TSP is known to be *NP-hard* [12] even under substantial restrictions. The case with symmetric distances is well researched and there are many algorithms which perform well even on large cases [1, 3]. In the literature [10, 11] the Traveling Salesman Problem is usually represented and considered as a graph theoretical problem.

An instance of the STSP can be seen as a complete graph $G = (V,E)$ where the set of vertices V is given by the cities and edges between each city in the graph with corresponding edge weights $d(C_i,C_j)$. The STSP then translates to the problem of finding a Hamiltonian Tour of minimal length in the graph G.

Applications of the TSP and its variations go way beyond the route planning problem of a traveling salesman and span over several areas of knowledge including mathematics, computer science, operations research, genetics, engineering, and electronics. In addition, there are many different variations of TSP which are described and explored in the literature and also variations derived from everyday life. Some of them are: Machine Scheduling Problems [4, 10], The time dependent TSP [9], The delivery man problem which is also known as the *minimum latency problem* and the *traveling repairman problem*, for details on this problems, we refer to [5, 8] respectively.

Traveling Tourist Problem [13] is a problem in which a tourist wishes to see all monuments (nodes) in a city, and so must visit each monument or a neighbour thereof (it is assumed that a monument is visible from any of its neighbours the edges therefore represent lines of sight). The resulting walk will therefore visit a subset of all nodes in the graph. The Traveling Tourist Problem shares a similar name with our problem but is otherwise a very different problem.

The STSP can be solved using the Grafted Genetic Algorithms (GGA) as was shown in [7]. The currently most efficient implementation of the branch-and-cut method which was introduced by Padberg and Rinaldi [14] for solving the symmetric case of Traveling Salesman Problem is *Concorde* [2]. Concorde's TSP solver has been used to obtain the optimal solutions to the full set of 110 TSPLIB instances, the largest having 85,900 cities. Finally, in a graph G we can find besides shortest closed walk also the shortest path between any pair of vertices. This problem is in the literature known as all-pairs shortest path problem [6]. It aims to compute the shortest path from each vertex u to every other vertex v. The Floyd-Warshall algorithm [6] is an efficient algorithm to find all-pairs shortest paths on a graph G.

2 Traveling Visitor Problem

Visitors have arrived in a hotel in some new town, with a desire to visit all interesting sites in a city exactly once and to come back to the hotel. Visitors in generally use their feet for traveling through the city, for which they use streets, walking trails and pedestrian zones. The goal is to minimize the visitors traveling.

The Traveling Visitor Problem is a version of the Traveling Salesman Problem with a difference that the traveling visitor, during its visit of sites, can not fly over the buildings in the city, instead visitors must go around these obstacles. This difference is demonstrate in the Figure 1. This means that the ”air” distances, as we know them in the TSP, are in this case impossible (direct edge from i to j in Figure 1). Visitors use the walking paths and pedestrian zones of variable length. These limits determine the weight of edges connecting the vertices in the graph.

The Traveling Visitor Problem is stated as: given a sparse, connected, weighted graph $G = (V,E,c)$, with a set of vertices $V = S \cup X$ and $S \cap X = \emptyset$, S belongs to interesting sites in the city (vertices i and j in Figure 1), X belongs to crossroads in the city (vertices k and m in Figure 1), a set of edges E, and a cost of traveling c. The goal is to find the shortest closed walk through all vertices from S, according to c in graph G, although we may travel through vertices from X.

The concepts we summarised above can be modified easily to take the directions of the edges into account. The asymmetric traveling visitor problem (ATVP) is then similar to the symmetric TVP above, i. e. it is the problem of finding a closed walk of minimal length in a sparse weighted graph. The *Euclidean TVP*, or planar TVP, is the TVP with the distance being the ordinary *Euclidean distance*. The Euclidean TVP is then a particular case of the metric TVP, since distances in a plane obey the Euclidian triangle inequality.

This problem, by the knowledge of the authors, has no references in publications due date of writing it.

3 Algorithms for solving TVP

First thinking about possible solution for Traveling Visitor problem is motivated by the intuitive thinking of a tourist when the concerned get in possession of a tourist map. That is: visit the first place from the map, then second one, then n^{th}, until all

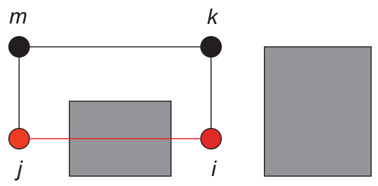

Fig. 1 TSP and TVP, Two rectangles represent buildings (obstacles) in the city. Red nodes represent interesting sites in the city (vertices from set S), black nodes represent crossroads in the city (vertices from set X), the red line represent the euclidean shortest connection between two interesting sites (this is the case in TSP), black lines represent the connection between two interesting sites, going through two crossroads (this is the case in TVP)

sites from the map are visited and then come back to the starting site. The results of this method depend directly on the order in which the interesting sites are listed on the map. Furthermore, this intuitive method does not contain any science value.

First proposed method for solving the Traveling Visitor Problem is the Naïve based algorithm, shown in Algorithm 1. In the first line of pseudocode we can distinguish next parameters: S belongs to interesting sites in the city, X belongs to crossroads in the city, a set of edges E, and W represents the distance matrix of the graph G, $(S \cup X \times S \cup X)$. In the first step of an algorithm the Traveling Visitor Problem is solved as an instance of Traveling Salesman Problem. In next, from the distance matrix W we produce a distance matrix Z $(S \times S)$, which is the solution of all-pairs shortest path problem (APSP). Finally, in the loop block (lines 6 through 8) the solution for TVP is given by applaying the shortest paths from Z into T.

Algorithm 1 Naïve Algorithm

```
1: procedure NAÏVE(S,X,E,W)
2:     $T \leftarrow TSP(W)$
3:     $Z \leftarrow S \times S$
4:     $Z \leftarrow APSP(S \cup X, E, W)$
5:     cost $\leftarrow 0$
6:     for all $(i, j) \in T$ : do
7:         cost $\leftarrow$cost $+Z_{ij}$
8:     end for
9: end procedure
```

The second proposed method for solving the Traveling Visitor Problem is the Koper Algorithm, shown in Algorithm 2. The first line of pseudocode contains the same parameters as naïve algorithm. In the first step we find the all pairs shortest paths in our graph G. As an input a distance matrix W is used and as the output a distance matrix Z is obtained. In the next step we solve the Traveling Salesman Problem on the distance matrix Z. Furthermore, we get the solution T, which is a solution for Traveling Visitor Problem.

The difference in this two proposed methods is whether we first solve the TSP then APSP, it is the case in naïve algorithm, or we first solve APSP and then TSP which is the case in koper algorithm.

Algorithm 2 Koper Algorithm

```
1: procedure KOPER(S,X,E,W)
2:     $Z \leftarrow S \times S$
3:     $Z \leftarrow APSP(S \cup X, E, W)$
4:     $T \leftarrow TSP(Z)$
5: end procedure
```

3.1 Adapted Floyd−Warshall algorithm

The problem stated in the previous section is of finding the shortest paths between each pair of vertices u and v, where $u, v \in S$, in the graph G. This can be cast as a run-of-the-mill all-pairs shortest path problem. Indeed, using the Floyd-Warshall algorithm, we can obtain a solution in time $\Theta(|V|^3)$. However, the nature of our problem is somewhat more restrictive: we are only interested in the shortest paths between $S \times S$, yet we would still like the paths to go through vertices from the set X if they reduce the overall path length. In contrast, the Floyd-Warshall algorithm computes a shortest paths between $V \times V$. To this end, we propose a simple modification which reduces the running time, albeit not asymptotically. The Floyd-Warshall algorithm is shown in Algorithm 3, where W is the distance matrix of the graph G.

Algorithm 3 Floyd-Warshall

```
1: procedure FLOYD-WARSHALL(V,W)
2:     for all $k \in V$ do
3:         for all $i \in V$ do
4:             for all $j \in V$ do
5:                 $W_{ij} := min(W_{ij}, W_{ik} + W_{kj})$
6:             end for
7:         end for
8:     end for
9: end procedure
```

Let $x = |X|$ and $s = |S|$. Using these quantities, the number of iterations of the Floyd-Warshall algorithm can be written as $(s+x)^3 = s^3 + x^3 + 3s^2x + 3x^2s$. We offer a different approach, shown in Algorithm 4.

The number of iterations of algorithm 4 can be plainly seen to equal: $s^3 + x^3 + s^2x + x^2s$. The best gain, when compared to Floyd-Warshall, is when $s = x$ which amounts to exactly one half of all iterations of the Floyd-Warshall algorithm. Although it takes fewer iterations, it also computes fewer shortest paths, since we are only interested in $S \times S$. We will prove the correctness of algorithm 4 by appealing to the graph shown in Fig. 3.1.

In order to examine how algorithm 4 works, it is helpful to visualize sets of vertices, as shown in Fig. 3.1. It should be noted that we will make use of a sparsely connected graph, which simplifies the analysis. The result does not change for complete graphs, since the algorithm itself makes no such assumptions.

The first call to Floyd-Warshall (line 2) in algorithm 4 finds the all-pairs shortest paths between the vertices in X, but using only vertices from X on the paths themselves. Note that there are two such sets shown in Fig. 3.1, i.e. X' and X'', with no direct edges between them. Thus, we can only find the shortest paths inside the individual sets. Once the paths are found, we can find our way from any vertex in X to any vertex in X if a path that does not take us through vertices in S exists.

Algorithm 4 Adapted Floyd-Warshall Algorithm

```
1: procedure ADAPTED(S,X,W)
2:     FLOYD-WARSHALL($X$, $W$)
3:     for all $k \in X$ do
4:         for all $i \in X$ do
5:             for all $j \in S$ do
6:                 $W_{ij} := min(W_{ij}, W_{ik} + W_{kj})$
7:             end for
8:         end for
9:     end for
10:    for all $k \in X$ do
11:        for all $i \in S$ do
12:            for all $j \in S$ do
13:                $W_{ij} := min(W_{ij}, W_{ik} + W_{kj})$
14:            end for
15:        end for
16:    end for
17:    FLOYD-WARSHALL($S$, $W$)
18: end procedure
```

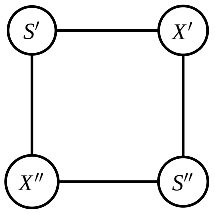

Fig. 2 Each node in the graph represents an arbitrary amount of vertices from a single set that are arbitrarily interconnected. The edges represent (arbitrarily many) connections to other such sets. Note, that $S', S'' \subset S$ and $X', X'' \subset X$.

The first loop block (lines 3 through 9) of Algorithm 4 finds every shortest path starting in X and ending in S, by going through vertices in X only. Every vertex in X knows the path to every other vertex in X, as long as the path does not go through vertices in S. At this point there must exist a pair of vertices $u \in X$, $v \in S$ where $W_{u,v} < \infty$ [1]. Thus, when the first loop block finishes, every vertex in X knows the shortest paths through X to some vertices in S. In Fig. 3.1 this means that the vertices in X' know the shortest paths through X' that end in S' or S''. The same is true for vertices in X''.

Finally, the second loop block (lines 10 through 16) of the algorithm finds every shortest path starting in some vertex in S, going through some vertex in X and ending in some vertex in S. The only vertices in S that have paths to vertices in X are those that have edges that connect them. However, the vertices in X that they are connected to, know the shortest paths through X ending in some vertices in S. Thus, the algorithm connects the sets S' and S'' via the shortest paths through X' and X''.

[1] If there were no such pair, a path from S to S going through X would not exist.

At the end (line 17), we run the Floyd-Warshall algorithm on S. Since the sets S' and S'' have been connected via shortest paths through X, we obtain the APSP solution for $S \times S$ whereby the paths can go through X.

4 Experiment

For testing our strategy and comparing it to other methods we used the real instances of the Traveling Visitor Problem, which were made from official tourist maps of cities of Koper, Belgrade and Venice. In the instance of Belgrade two different cases were made and they differed in the size of the problem, i.e. the number of vertices in the graph. From the publicly available library, TSPLIB, of sample benchmarks for the TSP and related problems, two instances of the symmetric traveling salesman problem were taken, modified and tested.

These two instances were modified in such a way that a new graph G' was made satisfying the conditions of a sparse, connected, weighted graph. Furthermore we split V into a set of vertices S and set of vertices X, such that $|S| = |X| = |V|/2$. A vertex degree 5 is arbitrarily assigned, inspired by the case of real instances, and means that from every vertex from V there is exactly 5 edges going to the other vertex from V. The 5 edges per vertex were chosen randomly, according to a uniform probability distribution.

Altogether 5 instances were tried out, with different sizes which range from 120 to 1002 vertices per instance. We compared two methods for solving the traveling visitor problem. The first method is the naïve algorithm, shown in Algorithm 1. The second tested method is the koper algorithm, shown in Algorithm 2. For solving the TSP, as one step in both algorithms, we used the Concorde Algorithm, presented in Section 1. Furthermore, for solving the APSP, as a part of both algorithms, we used the Adapted Floyd−Warshall algorithm, which was presented in Section 3.1.

5 Results

The results of the experiment are summarized in Table 1. Five instances were tried out, with different sizes, ranging from 120 to 1002 vertices per instance. The names of these cases are in the first column. The second column contains the size of the problem, i.e. the number of vertices. The third column in Table 1 corresponds to the number of vertices in set S and in the top three instances the number of interesting sites from tourist maps. The fourth column contains names of the two tested methods. The fifth column corresponds to the length of the tour i.e. the cost of a solution which was obtained in the experiment. The column is titled Tour Cost and in all six cases the shortest tours are obtained by Koper Algorithm. The results for Koper Algorithm are coloured in red. The last column corresponds to the difference in tested methods displayed in percentages. The first tested method, the naïve algorithm, performed poorly in comparison to the koper algorithm. The quality differs from 6.52% in the case of Belgrade163 to 354.46% in the case of pr1002 instance. Although the algorithms are similar, (the difference is whether we first solve the TSP then APSP, it is the case in naïve algorithm, or we first solve APSP and then TSP which is the case in koper algorithm) the difference in provided solutions between two tested methods is significant.

Name	(V)	(S)	Methods	Tour Cost	Difference
Koper	120	55	Naïve	4738	17.22%
			Koper	4042	0.00%
Belgrade	163	53	Naïve	100389	6.52%
			Koper	94246	0.00%
	250	90	Naïve	122119	8.77%
			Koper	112275	0.00%
Venice	210	72	Naïve	26648	24.24%
			Koper	21448	0.00%
pr1002	1002	501	Naïve	11818732	354.46%
			Koper	2600585	0.00%
lin318	318	159	Naïve	921499	249.08%
			Koper	263983	0.00%

Table 1 Two techniques for solving the Traveling Visitor Problem

6 Conclusions

The goal of this paper was to describe a new problem from graph theory, named the Traveling Visitor Problem (TVP). Although the new problem is similar to the well known Traveling Salesman Problem, when we try to solve it with the Naïve Algorithm we get solutions far from optimal. The minimum cost solutions for the Traveling Visitor Problem instances tested in the paper are provided by Koper Algorithm. The tested benchmarks are combined from three real instances made using tourist maps of cities of Koper, Belgrade and Venice and two instances of modified cases from TSPLIB. In all tested cases the Koper Algorithm significantly outperforms the Naïve Algorithm for solving the Traveling Visitor Problem.

References

1. D. Applegate, R. Bixby, V. Chvatal, and B. Cook. Finding cuts in the TSP. Technical report, Center for Discrete Mathematics and Theoretical Computer Science, 1995.
2. D. Applegate, R. Bixby, V. Chvátal, and W. Cook. TSP cuts which do not conform to the template paradigm. In *Computational Combinatorial Optimization*, pages 261–304, 2001.
3. D. Applegate, R. Bixby, W. Cook, and V. Chvátal. *On the solution of traveling salesman problems*. Rheinische Friedrich-Wilhelms-Universitat Bonn, 1998.
4. M.O. Ball and M.J. Magazine. Sequencing of insertions in printed circuit board assembly. *Operations Research*, pages 192–201, 1988.
5. A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 163–171. ACM, 1994.
6. T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
7. M. Djordjevic and A. Brodnik. Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm. In *Proceedings of the International Conference on Operation Research-OR2011*, pages 130–132. IFOR, ETH Zurich, 2011.
8. A. Garcia, P. Jodrá, and J. Tejel. A note on the traveling repairman problem. *Networks*, 40(1):27–31, 2002.
9. L. Gouveia et al. A classification of formulations for the (time-dependent) traveling salesman problem. *European Journal of Operational Research*, 83(1):69–82, 1995.
10. G. Gutin and A.P. Punnen. *The traveling salesman problem and its variations*. Kluwer Academic Pub, 2002.
11. D.S. Johnson and L.A. McGeoch. The traveling salesman problem: A case study in local optimization. In *Local search in combinatorial optimization*, pages 215–310. 1997.
12. D.S. Johnson and C.H. Papadimitriou. *Computational complexity and the traveling salesman problem*. Massachusetts Institute of Technology, 1981.
13. G.F. Lima, A.S. Martinez, and O. Kinouchi. Deterministic walks in random media. *Physical Review Letters*, 87(1):10603, 2001.
14. M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.