**Methods for Solving the Traveling Visitor Problem**

**Milan Djordjevic1, Marko Grgurovič1, Janez Žibert1 , and Andrej Brodnik2**

*1University of Primorska, Department of Information Science and Technology,*

*Koper, Slovenia
2University of Primorska, Department of Information Science and Technology,*

*Koper, Slovenia and University of Ljubljana, Faculty of Computer and*

*Information Science, Ljubljana, Slovenia,*

*E-mail(s): milan.djordjevic@student.upr.si, marko.grgurovic@student.upr.si, janez.zibert@upr.si, andrej.brodnik@upr.si*

**Abstract**

We consider a new problem named the Traveling Visitor Problem (TVP). Visitors start from a

hotel with desire to visit all interesting sites in a city exactly once and to come back to the hotel. Since, the visitors use streets and pedestrian zones, the goal is to minimize the visitor’s traveling distance. This problem is similar to the Traveling Salesman Problem (TSP). The difference is that during visit, traveling visitors cannot fly over buildings in the city. Instead they have to go around these obstacles. That means that all Euclidean distances, like those in Euclidean TSP, are not valid in this case. The tested benchmarks used come from three real instances made using tourist maps of cities of Venice, Belgrade and Koper and two modified instances from TSPLIB. We introduced and compared two exact methods for solving the TVP. In all tested cases the Koper Algorithm significantly outperforms the Naïve Algorithm for solving the TVP - quality of solutions differs from 6.52% to 354.46%.

**Keywords:** Traveling Salesman Problem, Traveling Visitor Problem, Floyd-Warshall

Algorithm, Koper Algorithm for TVP, Naïve Algorithm for TVP

**JEL classification:** C61 - Optimization Techniques; Programming Models; Dynamic Analysis

**Acknowledgements**

This work was partially supported by the Institute “Andrej Marušič” at the University of Primorska, Slovenia.

**1. Introduction**

In the Traveling Salesman Problem (TSP) a set of cities is considered and for each pair where, a distance is given. The goal is to find a permutation of the cities that minimizes the quantity

This quantity is referred to as the tour length since it is the length of the tour a salesperson would have to travel when visiting the cities in the order specified by the permutation, 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(Ci, Cj)=d(Cj,Ci)* for *1 ≤ i, j ≤ N*. The TSP is known to be *NP-hard* (Johnson and Papadimitriou, 1981). The case with symmetric distances has been well studied and there are many algorithms which perform well even on large cases (Applegate et al., 1995), (Applegate et al., 1988). In the literature the Traveling Salesman Problem is usually represented and considered as a graph theoretical problem (Gutin and Punnen, 2002), (Johnson and McGeoch, 1997).

An instance of the STSP can be seen as a complete graph where the set of vertices is given by the cities and edges between each city in the graph with corresponding edge weights. The STSP then translates to the problem of finding a Hamiltonian Tour of minimal length in the graph.

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* (Ball and Magazine, 1988), (Gutin and Punnen, 2002), the *time dependent TSP* (Gouveia and Vob, 1995), the *delivery man problem* which is also known as the *minimum latency problem* and the *traveling repairman problem*, for details on these problems, we refer to (Blum et al., 1994), (Garcia et al., 2002).

The Traveling Tourist Problem (Lima et al., 2001) is a problem in which a tourist wishes to see all monuments (nodes) in a city, and so must visit each monument or a neighbor thereof. It is assumed that a monument is visible from any of its neighbors and therefore, the edges 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 however it is a very different problem.

The STSP can be also solved using the grafted genetic algorithms (GGA) as was shown in (Djordjevic et al., 2009). The currently most efficient implementation of the branch-and-cut method that was introduced by Padberg and Rinaldi (Padberg and Rinaldi, 1991) for solving the symmetric case of TSP is named *Concorde* (Applegate et al., 2001).

Finally, in a graph besides finding the shortest closed walk we can also find the shortest path between any pair of vertices. This problem is in the literature known as *all-pairs shortest path problem* (APSP), (Cormen et al., 2009). It aims to compute the shortest path from each vertex to every other vertex. The Floyd-Warshall algorithm (Cormen et al., 2009) is an efficient algorithm to find all-pairs shortest paths on a graph.

**2. Traveling Visitor Problem**

Suppose that visitors have arrived in a hotel in some town, with a desire to visit all interesting sites in a city exactly once and to come back to the hotel at the end of their journey. Visitors in general move from one place to another using the existing streets, walking trails and pedestrian zones. The goal is to minimize the visitors traveling distance.

 The Traveling Visitor Problem is a version of the Traveling Salesman Problem with a difference that the traveling visitor, during its visit of sites, cannot fly over the buildings in the city; instead, visitors must go around these obstacles. This difference is demonstrated in Figure 1. This means that the Euclidean distances, as we know them in the Euclidean

TSP, are in this case not valid (direct edge from to 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.

Figure 1. TSP and TVP, Two rectangles represent buildings (obstacles) in the city. Red nodes represent interesting sites in the city (vertices from set ), black nodes represent crossroads in the city (vertices from set ), the red line represent the Euclidean distance 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)



 The Traveling Visitor Problem is stated as: given a connected, weighted graph , with a set of vertices and , is the set of interesting sitesin the city (vertices and in Figure 1), is the set including the nodes corresponding to crossroads in the city (vertices and in Figure 1), a set of edges , and a cost of traveling . The goal is to find the shortest closed walk through all vertices from, according to in graph, although we may travel through vertices from.

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

**3. Algorithms for solving TVP**

One simple approach of solving TVP is to visit all places as are ordered in the city's tourists maps and then come back to the starting site. The result of this method depend directly on the order in which the interesting sites are listed on the map and does not necessary find the shortest closed walk through all tourist sites.

The first proposed method for solving the Traveling Visitor Problem is the Naïve algorithm, shown in Algorithm 1. In the first line of pseudocode, we can distinguish next parameters:is the set of interesting sites in the city, is the set of crossroads in the city, a set of edges, and represents the distance matrix of the graph. The First step of the algorithm is to solve a TSP considering every node in. Then a tour, denoted by , is obtained. In the next step, a distance matrix is constructed by solving the APSP for every node in. Note that to calculate these distances we do use distance matrix (nodes in ). Finally, we calculate the final cost combining the shortest path solution obtained in the second step with the optimal tour obtained in the first step.

**Algorithm 1** Naïve Algorithm

1. **procedure**Naïve
2. cost *0*
3. **for all** *:* **do**
4. cost cost *+*
5. **end for**
6. **end procedure**

The second proposed method for solving the Traveling Visitor Problem is our Koper algorithm, shown in Algorithm 2. The first line of pseudocode contains the same parameters as Naïve algorithm. In the first step, we find 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. Furthermore, we get the solution, which is a solution for Traveling Visitor Problem. The authors of this paper named algorithms presented in this section.

**Algorithm 2** Koper Algorithm

1. **procedure**Koper
2.
3. **end procedure**

**3.1. Adapted Floyd-Warshall algorithm**

The problem stated in Section 1 is of finding the shortest paths between each pair of vertices and, where, in the graph. This can be cast as a run-of-the-mill all-pairs shortestpath problem. Indeed, using the Floyd-Warshall algorithm, we can obtain a solution in time However; the nature of our problem is somewhat more restrictive: we are only interested in the shortest paths between, yet we would still like the paths to go through vertices from the set if they reduce the overall path length. In contrast, the Floyd-Warshall algorithm computes shortest paths between . 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 is the distance matrix of the graph.

**Algorithm 3** Floyd-Warshall Algorithm

1. **procedure**Floyd - Warshall
2. **for all**
3. **for all**
4. **for all**
5. **end for**
6. **end for**
7. **end for**
8. **end procedure**

Let and. Using these quantities, the number of iterations of the Floyd- Warshall algorithm can be written as. We offer a different approach, shown in Algorithm 4.

**Algorithm 3** AdaptedFloyd-Warshall Algorithm

1. **procedure**Adapted
2. Floyd-Warshall
3. **for all**
4. **for all**
5. **for all**
6. **end for**
7. **end for**
8. **end for**
9. **for all**
10. **for all**
11. **for all**
12. **end for**
13. **end for**
14. **end for**
15. Floyd-Warshall
16. **end procedure**

The number of iterations of Algorithm 4 can be plainly seen to equal: . The best gain, when compared to Floyd-Warshall, is when 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. We will prove the correctness of Algorithm 4 by appealing to the graph shown in Figure 2.

Figure 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 and .



In order to examine how Algorithm 4 works, it is helpful to visualize sets of vertices, as shown in Figure 2. 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, but using only vertices from on the paths themselves. Note that there are two such sets shown in Figure 2, i.e. and, 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 to any vertex in if a path that does not take us through vertices in exists.

The first loop block (lines 3 through 9) of Algorithm 4 finds every shortest path starting in and ending in, by going through vertices in only. Every vertex in *X* knows the path to every other vertex in, as long as the path does not go through vertices in. At this point there must exist a pair of vertices, where. Thus, when the first loop block finishes, every vertex in knows the shortest paths through to some vertices in. In Figure 2 this means that the vertices in know the shortest paths through

that end in or . The same is true for vertices in.

Finally, the second loop block (lines 10 through 16) of the algorithm finds every shortest path starting in some vertex in, going through some vertex in and ending in some

vertex in . The only vertices in that have paths to vertices in are those that have edges that connect them. However, the vertices in that they are connected to, know the shortest paths through ending in some vertices in. Thus, the algorithm connects the sets and via the shortest paths through and.

At the end (line 17), we run the Floyd-Warshall algorithm on. Since the sets and

have been connected via shortest paths through , we obtain the APSP solution for whereby the paths can go through .

**4. Experiments**

For testing our strategies, described in Section 3 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 Belgrade example two different cases were considered 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 selected, modified and tested.

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

Altogether five 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 Algotithm 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 (Applegate et al., 2001). 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. All experiments were conducted on a computer with Pentium®2.8 GHz CPU and Windows 7 operating system.

**5. Results**

The results of the experiment are summarized in Table 1.

Table 1 Two techniques for solving the Traveling Visitor Problem

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **(V)** | **(S)** | **Methods** | **Tour Cost** | **Difference** |
| **Koper** | 120 | 55 | NaïveKoper | 47384042 | 17.22% |
| **Belgrade** | 163 | 53 | NaïveKoper | 10038994246 | 6.52% |
| 250 | 90 | NaïveKoper | 122119112275 | 8.77% |
| **Venice** | 210 | 72 | NaïveKoper | 2664821448 | 24.24% |
| **lin318** | 318 | 159 | NaïveKoper | 921499263983 | 249.08% |
| **pr1002** | 1002 | 501 | NaïveKoper | 118187322600585 | 354.46% |

The names of these instances are in the first column. The second column contains the size of the problem, i.e. the number of vertices in set . The third column in Table 1 corresponds to the number of vertices in set 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. In all six cases, Koper Algorithm obtains the shortest tours. The last column corresponds to the quality of the result. The comparison of both algorithms is computed by the relative difference of cost solutions that is defined as

 *(2)*

where is a length of the tour obtained by Koper algorithm and is a length of the tour obtained by Naïve algorithm. The relative difference is expressed in percents, where, for example, 17.22% difference means that Naïve algorithm performed 17.22% worse than Koper algorithm.

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. The difference in the quality of the solutions of two tested algorithms grows with more complex instances. Since, the adapted Floyd-Warshall Algorithm has the same time complexity as the original one; it would be very interesting to have some comparisons on only this part for all the instances. Therefore, we can see if the modifications have at least impact in the actual execution time. Furthermore, it would be interesting to have more insights into the lin318 and pr1002 instances and see why the differences of the two algorithms are so big.

Although these algorithms are similar in terms of components (both rely on solving an APSP and TSP), the difference on the quality of the solutions indicates that there is a gain by using the Koper algorithm. Note also that the running CPU time for both algorithms is the same.

**6. Conclusions**

The goal of this paper is to describe a new problem from graph theory, named the Traveling Visitor Problem. Although the new problem is similar to the Traveling Salesman Problem, when we try to solve it with the Naïve algorithm we get solutions far from optimal. Our Koper algorithm provides the minimum cost solutions for the Traveling Visitor Problem instances tested in the paper. The tested benchmarks are obtained from three real instances coming from tourist maps of cities of Koper, Belgrade and Venice and two modified instances from TSPLIB. In all tested cases, the Koper algorithm significantly outperforms the Naïve algorithm for solving the Traveling Visitor Problem.

The paper opens a number of interesting questions for future research. The first one is related to the size of the problem. Namely, how will some heuristic algorithm behave in cases, where *Concorde* can no longer compute the optimal solution? A very interesting question is also how to apply our technique to Asymmetric Traveling Visitor Problem.

**References**

Applegate, D., Bixby, R., Chvátal, V., and Cook, B. (1995). Finding cuts in the TSP. Technical report, Discrete Mathematics and Theoretical Computer Science, Rutgers.

Applegate, D., Bixby, R., Chvátal, V., Cook, W. (2001). TSP cuts which do not conform to the template paradigm. In Jünger, M., Naddef, D. (Eds.), Computational Combinatorial

Optimization (pp. 261–304).

Applegate, D., Bixby, R., Cook, W., Chvátal, V. (1988). On the solution of traveling salesman

problems. Rheinische Friedrich-Wilhelms-Universitat Bonn, Bonn.

Ball, M.O., Magazine, M.J. (1988). Sequencing of insertions in printed circuit board assembly. Operations Research, 36(2), 192–201.

Blum, A., Chalasani, P., Coppersmith, D., Pulleyblank, B., Raghavan, P., Sudan, M. (1994). The minimum latency problem. In Proceedings of the twenty-sixth annual ACM symposium on Theory of computing (pp. 163–171).

Cormen, T., Leiserson, C., Rivest, R., Stein, C. (2009). Introduction to Algorithms, 3rd edition. The MIT Press.

Djordjevic, M., Tuba, M., Djordjevic, B. (2009). Impact of grafting a 2-opt algorithm based

local searcher into the genetic algorithm. In Proceedings of the 9th international conference on Applied informatics and communications (pp. 485–490). Stevens Point, WSEAS.

Garcia, A., Jodrá, P., Tejel, J. (2002). A note on the traveling repairman problem. Networks,

40(1), 27–31.

Gouveia, L., Vob, S. (1995). A classification of formulations for the (time-dependent) traveling salesman problem. European Journal of Operational Research, 83(1), 69–82.

Gutin, G., Punnen, A. P. (2002). The traveling salesman problem and its variations. Kluwer

Academic Publishers, Dordrecht.

Johnson, D. S., McGeoch, L. A. (1997). The traveling salesman problem: a case study in

local optimization. In Local search in combinatorial optimization (pp. 215–310). J. Wiley,

Chichester.

Johnson, D.S., Papadimitriou, C.H. (1981). Computational complexity and the traveling salesman problem. Massachusetts Institute of Technology, Cambridge.

Lima, G.F., Martinez, A.S., Kinouchi, O. (2001). Deterministic walks in random media.

Physical Review Letters, 87(1), 10603.

Padberg, M., Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale

symmetric traveling salesman problems. SIAM review, 33(1), 60–100.