A note on computational aspects of the Steiner traveling salesman problem

Eduardo Álvarez-Miranda *1 and Markus Sinnl†2

1 Department of Industrial Engineering, Universidad de Talca, Curicó, Chile
2 Department of Statistics and Operations Research, Faculty of Business, Economics and Statistics, University of Vienna, Vienna, Austria

Abstract

The Steiner traveling salesman problem (StTSP) is a variant of the classical traveling salesman problem (TSP). In the StTSP, we are given a graph with edge distances, and a set of terminal nodes, which are a subset of all nodes. The goal is to find a minimum distance closed walk, which visits each terminal node at least once. Two recent articles proposed solution approaches to the problem, namely, on the exact side, Letchford et al. [9] proposed compact Integer Linear Programming models, and on the heuristic side, Interian and Ribeiro [8] proposed greedy randomized adaptive search procedure, enhanced with a a local search. In these papers, the exact approaches could solve instances with up to 250 nodes to optimality, with runtimes up to 1400 seconds, and the heuristic approach was used to tackle instances with up to 3353 nodes, with runtimes up to 8500 seconds.

In this note, we show that by transforming the problem to the classical TSP, and using a state-of-the-art TSP solver, all instances from literature can solved to optimality within 20 seconds, most of them within a second. We provide optimal solution values for fourteen instances, where the optimal solution was not known.

1 Introduction and methodology

The traveling salesman problem (TSP) is one of the most prominent problems in combinatorial optimization, and in mathematical optimization in general. The TSP and its variants have many applications areas, for example routing problems in logistics and transportations contexts. Given a set of nodes and distances between them, the undirected version of the TSP consists of finding a minimum distance tour that visits each node (which might represent a final client, a retailer or a city) exactly once and returns to the first node. Mathematically, the TSP can be defined as follows: given a graph $G = G(V, E)$, and an edge distance function $\mathbf{c} : E \to \mathbb{R}^{|E|}_{\geq 0}$, find a minimum total distance (Hamiltonian) tour that visits each node in V exactly *once*. Note that the TSP is usually defined on a complete graph. See, e.g., [2, 3, 7] and [13], for comprehensive studies on modeling and algorithmic aspects of the TSP and related problems.

Despite of the relevance of the TSP, in real life applications, routing and delivery operations normally differ with respect to the TSP setting: i) road networks are normally non-complete but rather large sparse graphs; as a matter of fact, since decades sparsity of road networks have been identified as an important issue when tackling practical routing problems (see, e.g., [6, 11, 12]); and ii) often, not all but rather a subset of clients must be visited. Such clients might represent those that have been previously appointed in a particular day, or strategical clients that must be visited under all circumstances (see, e.g., [10]). In other

*ealvarez@utalca.cl
†markus.sinnl@univie.ac.at

words, we are given a subset of *required*, *visiting* or *terminal* nodes $T \subseteq V$, so that the optimization goal is to find a minimum distance *closed walk*, that visits each node in T *at least* once (and other nodes could be visited as well). The resulting closed walk is not necessarily a Hamiltonian tour, as nodes may be visited more than once and edges may be traversed more than once. The resulting problem was independently introduced by [4] and [6]; it was called *Steiner* TSP (StTSP) in the former paper (due to the presence of terminal nodes as in the well-known Steiner tree problem), and as *road network* TSP in the later one (due to the optimization on a sparse road network instead of on a complete one). In both works, complementary proofs of NP-hardness are provided.

Previous Work and Our Contribution The problem was introduced in the 1980s, in the following two papers. In [4], the authors give a polynomial-time algorithm for the StTSP in series-parallel graphs. In [6], a cutting-plane approach to the problem is proposed by the authors. In both [4] and [6], it is mentioned that the StTSP can be solved by transforming the problem into a TSP instance. The transformation works by first computing shortest-paths between all nodes in T, which yields the edge set E', and then solving the TSP on the resulting complete graph $G' = (T, E')$ (see also [8] and [9]).

Recently, there has been renewed interest in the problem. In Letchford et al. [9], a compact Integer Linear Programming (ILP) models is proposed, which allows solving instances with up to 250 nodes to optimality, with runtimes up to 1400 seconds. Additionally, Interian and Ribeiro [8] develop a greedy randomized adaptive search procedure, enhanced with a a local search. This heuristic approach is used to tackle instances with up to 3353 nodes, with runtimes up to 8500 seconds. In both of these recent works, it is mentioned that transforming the StTSP into the TSP results in computationally burdensome TSP instances. The authors argue that while the StTSP instances are usually sparse, the TSP instances associated to G' considers many edges that may never be used. Thus, none of these papers exploit this TSP transformation, and in both the problem is tackled directly, using exact and heuristic approaches, respectively. Therefore, in these papers, a comparison of the proposed algorithms with respect to the TSP-based approach was not performed. In the next section, we provide such a comparison, using the state-of-the-art TSP solver CONCORDE [1]. Our computations reveal that by doing so, all instances from literature can solved to optimality within 20 seconds, most of them within a second. We provide optimal solution values for fourteen instances, where the optimal solution was not known.

2 Computational Results

The shortest-paths needed for transforming the problem into a TSP were calculated using Dijsktra's algorithm [5] implemented in C++. To solve the resulting TSP instances, CONCORDE 03.12.19, as available at http://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm was used. The runs were made on an Intel Core i5 with 2.9 Ghz and 8GB RAM.

2.1 Benchmark Instances

We used the benchmark instances from Letchford et al. [9] and Interian and Ribeiro [8]. The instances are available online at http://www2.ic.uff.br/~rinterian/instances/stsp.html, and they are classified into two groups, as follows:

- The instances of the first group are named as `STSP_`n-α`over`β. These instances are based on random sparse graphs, and are generated as follows. For a given n (which denotes the number of nodes), the probability of any node being a terminal is α/β. Letchford et al. [9] generate instances using $n \in \{50, 75, 100, 125, 150, 175, 200, 225, 250\}$; and for each n, $\alpha \in \{1, 2\}$ and $\beta = 3$. Additionally, Interian and Ribeiro [8] generates two instances with $n = 300$, $\alpha \in \{1, 2\}$ and $\beta = 3$.
- The second group is proposed in Interian and Ribeiro [8], and they are named as `source_`ρ`percent`. These instances are generated from different real-life graph sources. Namely, the road network of Rome (instances `rome`), the main roads between European cities (instances `euroroad`), and the network of

the main internet providers (instances `tech-routers`). The probability ρ in the instance name gives the probability of any node being a terminal ($\rho \in \{0.05, 0.10, 0.15\}$).

2.2 Results

Table 1 gives a comparison of the results from Letchford et al. [9] and Interian and Ribeiro [8] our implementation of the TSP-based approach. The columns $|V|$, $|E|$ and $|T|$ give the number of nodes, edges and terminals of an instance, respectively. Columns $z_{[8]}$ and $t_{[8]}[s]$ give the solution value and runtime of the heuristic from Interian and Ribeiro [8]. The computations in [8] were made on a Intel Core i5 with a 2.9 GHz processor and 8 GB RAM. For instances `STSP*`, runs with different settings of their heuristic were made in [8]; hence, we report in the table, the runs giving the best solution value. Column $t_{[9]}[s]$ gives the runtime of the fastest approach from Letchford et al. [9]. The computations in [9] were made on an Intel Core i7 with 1.9 Ghz and 8 GB RAM, CPLEX 12.3 was used to solve the ILPs. A timelimit of 5000 seconds was imposed, two instances (`STSP_225-2over3` and `STSP_250-2over3`) could not be solved with any approach of [9] within the timelimit. Columns z^*, $t[s]$, $t_{spt}[s]$ give the (proven optimal) solution value obtained by our implementation, the corresponding overall runtime (including the time to transform the instance), and the time to transform the instance, respectively. An entry in *italics* in column z^* indicates that for the corresponding instance, the optimal solution value was not known before. Twelve out of these fourteen instances correspond to those generated from [8]; and the remaining two instances are from [9]). Likewise, a **bold** entry indicates that the found optimal solution value is better than the value found by the heuristic of [8].

Table 1: Comparison with literature . An *italic* entry in column z^* indicates that for this instance, the optimal solution value was not known before a *bold* entry indicates that the found optimal solution value is better than the value found by the heuristic of [8]

name	$\|V\|$	$\|E\|$	$\|T\|$	$z_{[8]}$	$t_{[8]}[s]$	$t_{[9]}[s]$	z^*	$t[s]$	$t_{spt}[s]$
STSP_50-1over3	50	69	16	789	1.71	0.00	789	0.02	0.00
STSP_50-2over3	50	69	33	978	3.95	1.48	978	0.02	0.00
STSP_75-1over3	75	105	25	830	4.06	0.00	830	0.02	0.00
STSP_75-2over3	75	105	50	1029	9.26	3.46	1029	0.07	0.00
STSP_100-1over3	100	139	33	919	7.20	0.00	919	0.05	0.01
STSP_100-2over3	100	139	66	1193	18.43	6.74	1193	0.11	0.01
STSP_125-1over3	125	179	41	1143	12.26	0.00	1143	0.04	0.01
STSP_125-2over3	125	179	83	1420	33.09	26.76	**1416**	0.75	0.01
STSP_150-1over3	150	215	50	1105	18.51	102.10	1105	0.10	0.01
STSP_150-2over3	150	215	100	1562	52.23	202.33	**1545**	0.49	0.02
STSP_175-1over3	175	252	58	1272	27.01	0.00	1272	0.06	0.01
STSP_175-2over3	175	252	116	1652	76.23	162.58	**1645**	2.27	0.02
STSP_200-1over3	200	291	66	1295	35.62	0.00	1295	0.07	0.01
STSP_200-2over3	200	291	133	1867	105.20	457.07	**1845**	1.35	0.03
STSP_225-1over3	225	326	75	1384	48.95	1420.60	**1377**	0.12	0.02
STSP_225-2over3	225	326	150	1928	142.26	TL	***1901***	2.22	0.03
STSP_250-1over3	250	367	83	1487	59.00	290.80	1487	0.12	0.02
STSP_250-2over3	250	367	166	2035	176.04	TL	***1994***	1.79	0.05
STSP_300-1over3	300	440	100	1629	85.95	-	*1629*	0.35	0.04
STSP_300-2over3	300	440	200	2205	296.76	-	***2128***	0.69	0.08
euroroad_05percent	1174	1417	58	31571	147.50	-	*31571*	0.43	0.07
euroroad_10percent	1174	1417	117	49975	354.90	-	*49975*	0.66	0.13
euroroad_15percent	1174	1417	176	58016	684.60	-	***57793***	1.42	0.18
euroroad_20percent	1174	1417	234	71967	1162.00	-	***71469***	2.90	0.24
tech-routers_05percent	2113	6632	105	20258	1030.50	-	***20230***	4.68	0.48
tech-routers_10percent	2113	6632	211	37486	3360.70	-	***37383***	10.47	1.01
tech-routers_15percent	2113	6632	316	51571	6850.60	-	***51323***	20.69	1.57
rome99_05percent	3353	8870	167	481048	1893.50	-	***474569***	2.73	0.67
rome99_10percent	3353	8870	335	622491	5196.20	-	***605047***	4.29	1.33
rome99_15percent	3353	8870	502	795535	8494.40	-	***769236***	4.97	2.27

From the reported results, we see that the shortest-path calculations for the transformation can be done very fast, the longest time is under 3 seconds, for `rome99_15percent`, which is the largest instance and has 502 required nodes. The longest overall runtime is about 20 seconds, for instance `tech-routers_15percent`, and only a second instance, `tech-routers_10percent` also takes over ten seconds to be solved to proven optimality. When comparing columns $t_{[8]}[s]$ and $t[s]$, we can see the runtime of TSP-based approach is up to three magnitudes faster than the heuristic of [8] (the specifications of the computers used in [8] and for our runs is similar.) Moreover, for 16 out of the 30 instances, the optimal solution is better than the solution found by the heuristic. For the instances tackled by the exact approach in [9], our implementation is capable of solving to optimality (and in less than two seconds), the two instances which [9] could not solve to optimality within their timelimit of 5000 seconds. In general, compared to [9], our approach seems to scale much better as the instance becomes larger.

The good performance of the transformation approach, in spite of the fact that the transformed graph G' has many more edges than G (e.g., 125751 against 8870 for the largest instance `rome99`), may be explained by the fact that, state-of-the-art TSP algorithms like CONCORDE use many techniques to deal with the issues associated with complete graphs (e.g., pricing-in of edge variables, see Section 8 of [1]).

Acknowledgments

E. Álvarez-Miranda acknowledges the support of the Chilean Council of Scientific and Technological Research (FONDECYT grant N.1180670 and Complex Engineering Systems Institute ICM:P-05-004-F/CONICYT:FB0816), and of the European Union's HORIZON 2020 research and innovation programme under the Marie Skodowska-Curie Actions (grant agreement N.691149, SuFoRun Project). M. Sinnl is supported by the Austrian Research Fund (FWF, Project P 26755-N19).

References

[1] Applegate, D., Bixby, R., Chvátal, V., Cook, W., 2003. Implementing the dantzig-fulkerson-johnson algorithm for large traveling salesman problems. *Mathematical programming* 97, 1-2, 91–153.

[2] Applegate, D., Bixby, R., Chvatál, V., Cook, W., 2006. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press.

[3] Cook, W., 2012. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation.* Princeton University Press.

[4] Cornuéjols, G., Fonlupt, J., Naddef, D., 1985. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming* 33, 1, 1–27.

[5] Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1, 269–271.

[6] Fleischmann, B., 1983. Distance conserving reductions for nonoriented networks. *Operations-Research-Spektrum* 5, 4, 195–205.

[7] Gutin, G., Punnen, A., 2002. *The Traveling Salesman Problem and Its Variations.* Combinatorial Optimization. Springer.

[8] Interian, R., Ribeiro, C., 2017. A grasp heuristic using path-relinking and restarts for the steiner traveling salesman problem. *International Transactions in Operational Research* 24, 6, 1307–1323.

[9] Letchford, A., Nasiri, S., Theis, D., 2013. Compact formulations of the steiner traveling salesman problem and related problems. *European Journal of Operational Research* 228, 1, 83–92.

[10] McKinsey & Company, 2016. How customer demands are reshaping last-mile delivery. Accessed at 31.01.2018.

[11] Miliotis, P., Laporte, G., Nobert, Y., 1981. Computational comparison of two methods for finding the shortest complete cycle or circuit in a graph. *RAIRO Operations Research* 15, 3, 233–239.

[12] Orloff, C., 1974. A fundamental problem in vehicle routing. *Networks* 4, 1, 35–64.

[13] Toth, P., Vigo, D., 2014. *The Vehicle Routing Problem* (2nd edn.). SIAM.