MISTA 2019

Modifying Kruskal algorithm to solve open loop TSP

Pasi Fränti • Henrik Nenonen

Abstract Traveling salesman problem (TSP) has been widely studied for the classical closed loop variant. However, only little attention has been paid to the open loop variant. Open loop solution represents also a spanning tree, but not necessarily the minimum spanning tree (MST). However, solution for MST contains lots of information useful for solving TSP as well. In this paper, we modify Kruskal algorithm to find heuristic solution for TSP instead. The basic variant achieves 4.6% gap, and its randomized variant 0.65% gap with our O-Mopsi game datasets, on average. The algorithm is studied mainly to fulfill our curiosity, but the results show the method has some usefulness for providing a quick approximation for TSP.

1 Introduction

In the classical *closed loop* variant of *travelling salesman problem* (TSP), the goal is to visit all the nodes and then return back to the start node so that the length of the tour is minimized. A slightly different variant, referred as *open loop TSP*, we can skip the return to the start node. Both variants are known to be NP-hard [1]. Finding the optimum solution fast is therefore not realistic except for very small size inputs.

The open loop variant appears in *O-Mopsi* (http://cs.uef.fi/o-mopsi/) mobile-based orienteering game where the player needs to find a set of real-world *targets* with the help of GPS navigation [2]. Game playing includes three challenges: (1) navigating to the next target, (2) deciding the order of visiting the targets, (3) physical movement. Unlike in the classical orienteering, in which the order of visiting the targets is fixed, O-Mopsi players must optimize the order while playing the game. This corresponds to a special case of the *orienteering problem* [3]. Finding the optimum order is not required but desirable in order to minimize the tour length [4]. According to [2], among players who completed the tour, only 14% managed to find the best visit order, i.e. solving the TSP problem involved.

Pasi Fränti
School of Computing, University of Eastern Finland
E-mail: pasi.franti@uef.fi

Henrik Nenonen
School of Computing, University of Eastern Finland
E-mail: henrikne@student.uef.fi

There is an algorithmic connection between MST and TSP worth to explore. The solution of the open loop TSP is also a spanning tree, but not necessarily the minimum one, see Fig. 1. Thus, both solutions have equal number of links. This connection is interesting because, while TSP is NP-hard, MST can be solved in polynomial time using greedy algorithms like Prim and Kruskal. This connection has been utilized at least in three different ways.

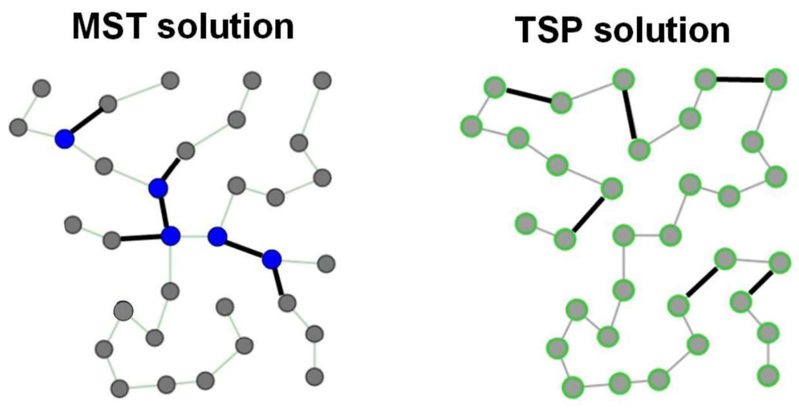


Figure 1: MST and open loop TSP solutions for the same problem instance.

First, classical *Christofides algorithm* [5, 6] starts by creating an MST, and then adds more links so that Euler tour becomes possible. Final solution is obtained by removing nodes from the tour that are visited more than once. Second, MST has also been used to estimate lower limit for the optimal TSP solution [7]. Third, the difficulty of the TSP instance was estimated in [8] by counting the number of branch nodes that branches the tree. The more branches, the more difficult is the instance to be solved both by human and by computer.

In this paper, we show another connection between MST and TSP. In specific, we modify classical *Kruskal algorithm* to create a heuristic solution for the open loop travelling salesman problem. Kruskal algorithm operates on a spanning forest by greedily merging two spanning trees increase the total cost of the spanning forest least. We introduce a simple constraint that allows merging two sub-solutions using only their end points. This not only prevents branches to appear in the tree but also reduces the number of possible links to be considered for the merge. Thus, it enforces the final result to be a solution for the open loop travelling salesman problem and allows potentially more efficient implementation.

Our experiments will show that the proposed *Kruskal-TSP algorithm* provides average gap of about 4.8 % from the optimum. Although this is still rather far from the optimum, it can be useful for providing reasonable estimate for the TSP solution. The algorithm is also fast and provides comparative solutions to other heuristics. It also has no restrictions like Christofides algorithm, which is limited to work only on metric data [5]. In addition, we also propose repeated randomized variant of the algorithm, which further reduces the gap down to 0.77%, on average.

2 Kruskal-TSP algorithm

2.1 Basic variant

The proposed Kruskal-TSP algorithm works similarly as classical Kruskal by always taking the smallest link that does not create a loop. While Kruskal produces a spanning forest (a set of spanning trees), our Kruskal-TSP produces a set of chains (sub-tours). The only limitation we need to add is to allow the merges to be made using only the end points of the two chains. This reduces the number of choices for the merge, and also causes sub-optimal spanning tree to be created; but being chain it gives a solution to the TSP as well.

Pseudo code of the modified Kruskal algorithm is shown in Figure 2. In the beginning, every node is assigned to its own set. The links are sorted and then processed from shortest to longest. Every link that connects two distinct sets *from their end points* are merged; thus, avoiding cycles and *branches*. Merges are continued until only one set remains. Efficient implementation can reach O(N^2) time complexity where N is the number of nodes.

The effect of the modification is shown in Figure 3. The positive side is that the number of choices for the next merge is significantly reduced but the negative effect is that the optimality cannot be anymore guaranteed.

```
Kruskal-TSP(V, E): RETURN T

FOR ALL v∈V
    CreateSet(v);
P←∅;  SORT(E);  // according to increasing weights
FOR each (u,v)∈E DO
    IF Set(u)≠Set(v) AND End(u) AND End(v) THEN
      UNION(u,v);
      P←P∪{(u,v)};
Return P;
```

Figure 2: Pseudo code of the Kruskal-TSP algorithm.
Modifications from standard Kruskal are marked in blue.

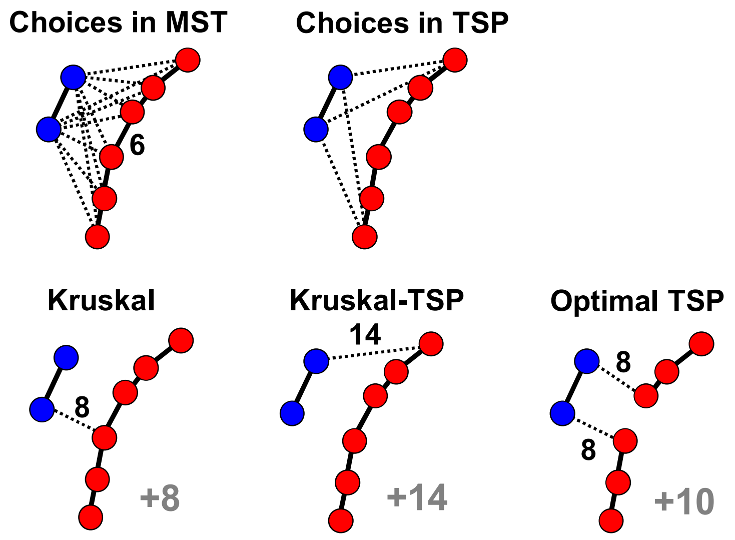


Figure 3: Difference of the merge in Kruskal and in the Kruskal-TSP.

Various similar heuristics were also summarized in [9] but without any details or experimental studies. Among them, Rosenkrantz et al. [10] considered two simple heuristics that are closest to our Kruskal-TSP. The first one is called *Myopic*. It starts from an arbitrary node and then subsequently adds always the nearest node to the one that was last added. The second one, called *Insertion algorithm*, adds the one that increases the tour least. In other words, addition is allowed at both ends of the tour. This idea can be considered as modification of Prim's MST algorithm, and therefore, is closest to our proposal.

The Kruskal-TSP algorithm is pretty much like single link agglomerative clustering except the linkage criterion must use only the end points. The benefit is that the merge is simpler to implement than in case of single link. The drawback is that it does not guarantee optimality. In the same way as with agglomerative clustering, early merges cannot be reversed and can cause sub-optimal choices later.

An example of the algorithm is given in Fig. 4 where selected steps of the algorithm are shown. The 6th merge is locally optimum choice at the time, but this link is not included in the optimal solution. It will cause sub-optimal choice for the 15th merge. In general, most errors happen at the later stages of the algorithm when there are less candidate pairs to be merged.

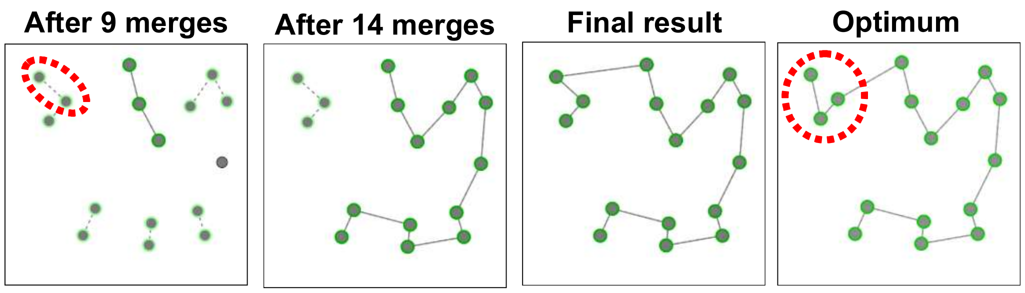


Figure 4: Example of the Kruskal-TSP algorithm with minor difference on the top-left corner.

2.2 Randomized variant

Since the algorithm is relatively fast, it is possible to improve its performance by repeating it several times. This improvement is inspired by [11] where it was shown that k-means clustering algorithm can be significantly improved simply by repeating the algorithm 100 times. The only condition is that the (initialization) algorithm contains randomness so that it produces (slightly) different results after every repeat. The final solution is the best result out of the 100 trial runs.

In our case, we add randomness as follows. Instead of always selecting the shortest link for the merge, we find 3 shortest links and select one of them by random choice. The consequence is that there will be making several sub-optimal choices. However, there are reasons why this can be done. First, the final result will be an approximation anyway, so it is not necessary that every link is optimally chosen. Second, our input is a complete graph, and therefore, there are often several almost equally good links to choose from.

The amount of variation in the process can be increased by changing the parameter 3. The more choices, the more variations there will be among the solutions. However, the average quality of the solutions tends to decrease with the randomness. It is therefore likely that a small value would work better. We tested with the values of 2-5 but the results were virtually the same in all cases. The optimal choice would depend on the size of the problem instance (N), and on the number of repeats.

2.3 Relationship between the open and closed loop variants

The open and closed loop cases are closely related. However, it is not trivial how to convert the solution from one to the other. For example, removing the longest link from the closed loop solution does not necessary lead to the optimum solution for the open loop case; and vice versa; see Fig. 5.

However, it is possible to modify the open loop problem instance to create a closed loop problem instance. One can then solve the problem using algorithm for the closed loop solution and derive solution for the open loop case [1, 12]. This also indicates that the open loop case is a special case of the closed loop.

The modification of the open loop solution happens as follows. First, we add artificial pseudo node with very large distance to all other nodes. The consequence of this is that two links with very large cost must be used, and that one normal link dropped. The nodes connecting to the pseudo node will act as the terminal nodes of the open loop solution. The final solution is then obtained by removing this phantom node and the links connecting to it. The reduction to the other way (closed loop to open loop) is less obvious. However, the proposed Kruskal-TSP can be applied to the closed loop version as such, if so wanted.

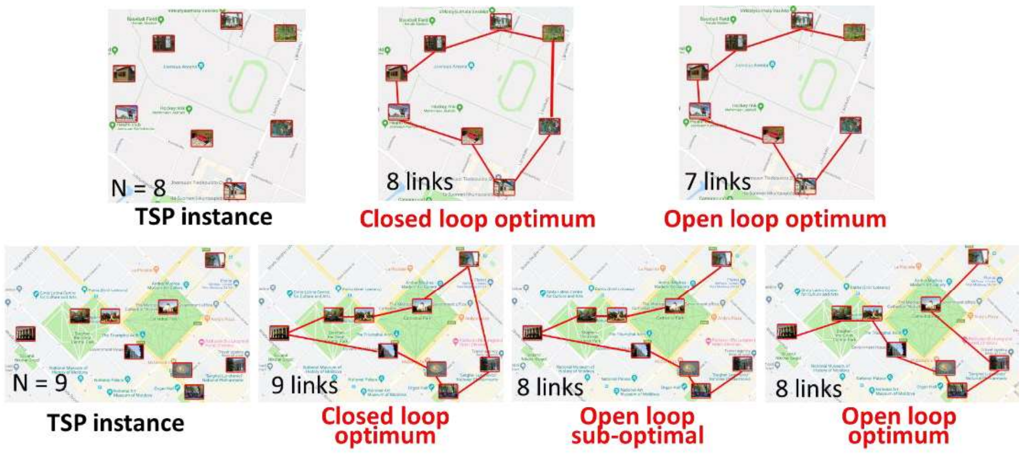


Figure 5: Difference between open and closed loop TSP. [13]

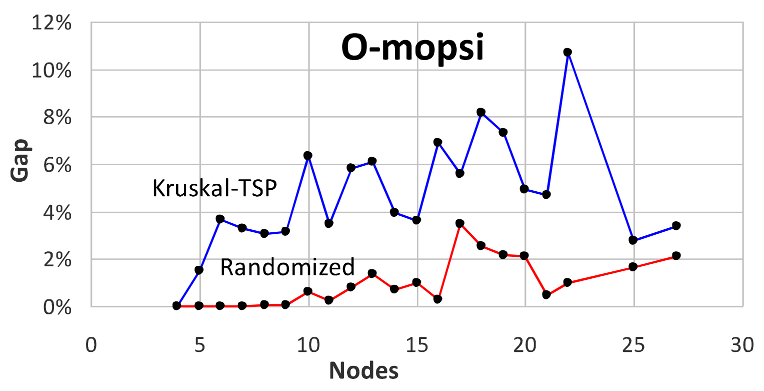
3 Experimental results

We use two open loop datasets found here: http://cs.uef.fi/o-mopsi/datasets/. The O-Mopsi dataset contains TSP instances used in the O-Mopsi game [2]. There are 145 game instances everywhere in the world; although most of them are in Joensuu, Finland. The Dots dataset contain 4226 computer-generated TSP instances that were used for experimenting human problem solving skills in [12]. Optimal ground truth solutions are given for all instances. To evaluate the performance of the algorithms, we calculate the *gap*, i.e. the relative difference of the solution provided by the algorithm, and the optimal solution:

$$gap = \frac{\left| L_{Kruskal} - L_{opt} \right|}{L_{opt}} \tag{1}$$

Table 1: Datasets used in the experiments.

Dataset:	**Reference:**	**Distance:**	**Instances:**	**Problem sizes:**
O-Mopsi	[2]	Haversine	145	4-27
Dots	[13]	Euclidean	4226	4-31



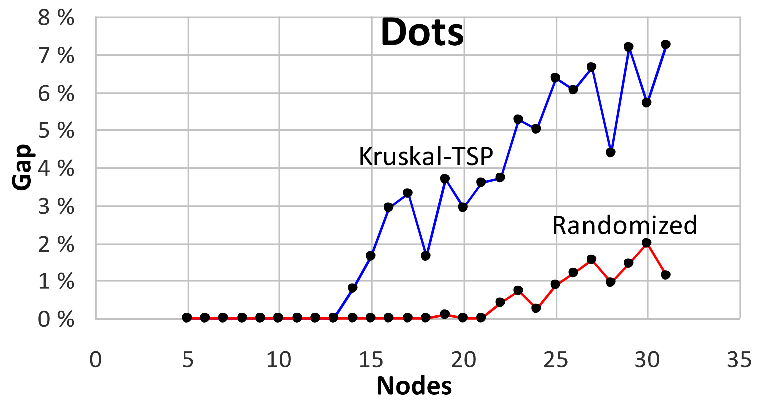


Figure 6: Results for the O-Mopsi (above) and Dots (below) as a function of problem size.

Results are plotted in Figure 6 for the O-Mopsi datasets with respect to the number of nodes. We can see that the Kruskal-TSP reaches the optimal solution only with the smallest dataset with 4 nodes. However, the gap remains less than 10% in almost all cases, being about 5%, on average (O-Mopsi) and <3% (Dots). The randomized variant reduces the gap further down to 0.77% (O-Mopsi) and 0.26% (Dots).

Comparison to Christofides [6] and the local search algorithm in [13] are summarized in Table 2. We apply Christofides using the added pseudo node strategy as explained in Section 2.3. The results show that Kruskal-TSP is better than Christofides but inferior to the local search. Optimal solution is found 35% (basic variant) and 74% (randomized) with the O-Mopsi data while local search finds it 100% times and start to struggle only with larger number of nodes. The results show the practical limitations of the proposed Kruskal-TSP. Nevertheless, we have shown by proof-of-concept that Kruskal algorithm can be modified to find reasonable approximations for TSP as well with real-life data.

Table 2: Summary of the results (gap).

Algorithm:	**Reference:**	**O-Mopsi:**	**Dots:**
Christofides	[6]	12.77%	9.10%
TSP-Kruskal	(proposed)	4.63%	2.66%
Randomized TSP-Kruskal	(proposed)	0.65%	0.26%
Local search	[13]	0.00%	0.001%

4 Conclusions

We introduced algorithm for solving open loop variant of TSP by modifying classical Kruskal algorithm. Although many heuristics exist in literature, extending Kruskal has a special interest as it shows the relationship between an NP-hard problem (TSP), and easier problem (MST).

The proposed Kruskal-TSP can be used when one needs an approximation of the TSP fast. The results also show that the randomized variant reaches <1% gap to the optimum, on average with our datasets. However, the performance weakens with the number of nodes in the graph. This is likely to leave the algorithm merely as theoretical interest to show how good results can be reached by such simple heuristics. Nevertheless, having reasonable approximation in real-time can still be useful. For instance, a good upper bound can also be critical when finding exact solutions for TSP and vehicle routing problems using branch-and-cut or by enumeration [14, 15]. The randomized Kruskal-TSP could also be used to provide input generation for a genetic algorithm instead of using merely random solution.

In principle, the algorithm can also be applied to closed-loop TSP but not recommended. The reason is that there are no alternatives for the last merge, as we just need to connect the two end points whatever they are. Effectively, it is almost like linking two random nodes, which can result in adding one arbitrary long link.

References

1. C.H. Papadimitriou, C.H. The Euclidean travelling salesman problem is NP-complete. Theoretical Computer Science, 4 (3), 237-244 (1977)
2. P. Fränti, R. Mariescu-Istodor and L. Sengupta, O-Mopsi: mobile orienteering game for sightseeing, exercising and education, ACM Trans. on Multimedia, Computing, Communications, and Applications, 13 (4), 56:1-25, (2017)
3. P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden: The orienteering problem: A survey. European Journal of Operational Research. 209, 1, 1–10 (2011)
4. H.H. Chieng, N. Wahid, A performance comparison of genetic algorithm's mutation operators in n-cities open loop travelling salesman problem. Recent Advances on Soft Computing and Data Mining, Springer (2014)
5. M.T. Goodrich, R. Tamassia, 18.1.2 The Christofides approximation algorithm, Algorithm Design and Applications, Wiley, pp. 513-514. (2015)
6. N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Report 388, Graduate School of Industrial Administration, CMU, 1976.
7. C.L. Valenzuela, A.J. Jones, Estimating the Held-Karp lower bound for the geometric TSP, European Journal of Operational Research, 102, 157-175 (1997)
8. L. Sengupta and P. Fränti, Predicting difficulty of TSP instances using MST, IEEE Int. Conf. on Industrial Informatics (INDIN), 847-852, Helsinki (2019)
9. G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. European Journal of Operational Research 59.2, 231-247. (1992)
10. J. Rosenkrantz, R.E. Stearns, P.M. Lewis II, An analysis of several heuristics for the travelling salesman problem, SIAM journal on Computing, 6, 563-581 (1997)
11. P. Fränti and S. Sieranoja, How much k-means can be improved by using better initialization and repeats?, Pattern Recognition, 93, 95-112 (2019)
12. L. Sengupta, R. Mariescu-Istodor and P. Fränti, Planning your route: where to start?, Computational Brain & Behavior, 1 (3-4), 252-265, (2018)
13. L. Sengupta, R. Mariescu-Istodor and P. Fränti, Which local search operator works best for open loop Euclidean TSP, *Applied Intelligence*, 9 (19), 3985, (2019)
14. L. Costa, C. Contardo, G. Desaulniers, Exact branch-price-and-cut algorithms for vehicle routing, Transporation Science, 53, (2019)
15. G. Homsi, R. Martinelli, T. Vidal, K. Fagerholt, Industrial and tramp ship routing problems: closing the gap for Real-scale instances, Arxiv. 1809.10584, (2018)