Simple and fast TSP initialization by Delaunay graph

Jimi Tuononen, Pasi Fränti*
School of Computing, University of Eastern Finland, 80101 Joensuu, FINLAND

ABSTRACT

Finding an initial solution for traveling salesman fast and with reasonable quality is needed both by the state-of-the-art local search and branch-and-bound optimal algorithms. Classical heuristics require O(n^2) and their quality may be insufficient to keep the efficiency of the local search high. In this paper, we present two fast variants that utilize a Delaunay neighborhood graph. The best methods achieve 2.66% gap for Dots datasets, and 15.04% gap for the selected TSPLIB datasets. However, these results were obtained with slower algorithms. The best of the fast methods achieved 5.76% gap for Dots datasets, and 19.07% gap for the selected TSPLIB datasets in 307 milliseconds, on average.

Keywords: Travelling salesman problem, TSP, initialization, Delaunay graph, greedy

1. INTRODUCTION

Traveling salesman is a well-known NP-hard problem. Finding optimal solutions requires exhaustive search such as branch-and-bound or Concorde [1] and becomes very slow for larger problem sizes. A compromise is to find a good quality but sub-optimal solution using local search such as Lin-Kerigan heuristic (LKH) [2]. It is based on the so-called *k*-opt operation and has been state-of-the-art for more than two decades already. Both approaches require an initial solution to start with.

There are many seemingly good heuristics available including nearest neighbor, greedy, and insertion algorithm that provide a reasonable solution [3, 4, 5]. However, their quality may be too low for branch-and-bound, or they are too slow. Optimal algorithms strongly depend on the quality of initialization. The higher the quality of the initial solution, the more branches can be cut early. One rule of thumb is that the initial solution should have a gap of about 1% or less to the optimal solution to achieve significant savings compared to exhaustive search.

The efficiency of local search also depends on the initialization. The higher the quality of the starting solutions, the fewer iterations are needed. Starting from random initialization can easily multiply the running time. Moreover, the simple heuristics also require O(n^2) time to run which adds to the total processing time. It is therefore an open question about which algorithm to use.

Divide-and-conquer approaches have been used to address large-scale problem instances. They divide the problem instance into several sub-problems by clustering. However, the results for Santa Claus data consisting of 1.4M nodes were not any better than the-state-of-the-art local search algorithm utilizing efficient neighborhood search [6]. The key trick is to apply a neighborhood graph to select the breaking point for the k-opt operation in the same neighborhood. This avoids wasting time on obviously bad trials connecting far away.

In this paper, we explore two simple heuristics based on the Delaunay graph. The neighborhood graph itself is already used in the LKH algorithm, but it has been less studied for the use of generating initial solutions. The LKH software [2] itself includes a few faster variants but they are not well documented and lack empirical evidence of which one should be used. We implement two new simple variants using the Delaunay graph and compare them against those existing variants by measuring the gap to optimal solution (when known), and the proportion of correctly chosen edges.

*pasi.franti@uef.fi; phone +358 50 442 22651; cs.uef.fi/ml

2. EXISTING HEURISTICS FOR TSP

We next briefly recall the existing heuristics for solving TSP. We consider only simple heuristics; referred as classical in [7]. They should be something simple but also generate fast to serve the purpose of being an initial solution for local search. High quality of the solution would also be desirable when using it as an initial solution for exhaustive search like branch-and-cut. However, this is only a secondary property and something below $O(n^2)$ is our primary goal. We consider the following algorithms:

- Nearest neighbor [3, 4]
- Nearest neighbor (fast) [2]
- Greedy graph [2]
- Kruskal-TSP [8]
- Moore [9]
- Sierpinski [10]
- Boruvka [11]
- Quick-Boruvka [7, 12]
- Christofides [13, 14]
- Walk [2]
- Delaunay shortest edge (new)
- Delaunay random edge (new)

The first thing to note is that a straightforward implementation of even the simplest nearest neighbor and greedy algorithms require $O(n^2)$ time. Fortunately, the LKH implementation has paid attention to this and utilizes a neighborhood graph [2]. The package implements fast variants of several of the above-mentioned heuristics.

Nearest neighbor starts from a random node and then continues to the nearest node until all nodes have been visited. It would be possible to create a heap from the distances to speed up the search to $O(n \log n)$ but the distance matrix still requires $O(n^2)$ time to generate. A faster variant in [2] utilizes the neighborhood graph. It considers only the neighbor nodes when considering the breaking points for k-opt. This trick speeds up the method to $O(nk)$, assuming that k is the number of neighbors. This is a solid approach for 2-D data.

The greedy algorithm sorts the edges in increasing order and then selects the shortest edge at each step. There are two alternative ways to implement this. The first variant constructs a single path by adding new nodes to its end nodes. This corresponds to the Prim's algorithm for finding a minimum spanning tree (MST) with the difference not allowing to create branches. The second variant maintains a forest of multiple trees (or paths in the case of TSP). The variants were referred to as Prim-TSP and Kruskal-TSP in [8] according to the corresponding MST algorithms they resemble. Here we used the second variant.

Boruvka is another greedy approach inspired by MST. The first step includes calculating the nearest neighbor of each node. It then processes these edges in ascending order and connects them into the tour if that does not create a branch or a cycle. It is analogous to Boruvka MST algorithm. The Quick-Boruvka skips the edge sorting step presumably sacrificing quality for speed [7, 12].

Christofides [13, 14] is a famous school-book algorithm. It uses MST as a starting point and adds supplementary edges by optimal pairing so that all nodes would have even edges to allow Euler tour. The result is pruned by shortcuts based on triangular inequality which holds for planar graphs. The algorithm has mediocre performance compared to other MST-based approaches [15] but is famous due to its 3/2 upper bound.

Space-filling curves have also been considered [16]. They divide the space into four quadrants, which are recursively divided further until every node has its own cell. The cells are indexed, and the space-filling curve traveling through the cells according to the order created by this index. TSP path is obtained by sorting the nodes according to their index (order in this curve) and requires only $O(n \log n)$ time due to the sorting. We consider the Moore curve [9] and Sierpinski curve [10] as they are implemented in the LKH software [2].

3. DELAUNAY-BASED HEURISTICS

A neighborhood graph has been used in [2] for speeding up the local search. In case of large problem instances, it is inefficient to consider randomly selected candidates for the k-opt operation. Connecting points located far away from each other is mostly a waste of time and should be avoided. Instead, the operator should only consider candidates in the same neighborhood. This is the key component in the-state-of-the-art algorithm where the so-called alpha nearest criterion is used to select the candidates [2].

It is logical to utilize the neighborhood graph also for the initialization as well. In the package [2], this has been done and a few fast variants of the classical heuristics have been implemented utilizing the alpha nearest criterion. However, in the recent Santa Claus challenge [6], it was observed that using the simpler Delaunay graph can improve the gap of LKH further from 109,284 to 108,996. While Delaunay is potentially too overwhelming in structure and time-consuming to create, this is not the case for 2-dimensional planar graphs. Motivated by this, we designed two simple variants based on the Delaunay: Delaunay random edge and Delaunay shortest edge.

3.1 Delaunay graph

The Delaunay graph is based on the Voronoi diagram which divides the space by drawing borders between the points so that it corresponds to the nearest neighbor mapping of every space point to its nearest reference point (node). Each node conquers its own region in this space. Connecting neighboring nodes creates so-called Delaunay triangulation and the graph created from these connections is called the Delaunay graph. While the size of the graph can become overwhelming in higher dimensional spaces, it is useful and efficient to construct in O(n log n) time in 2-D space. Some examples of Delaunay graphs are given in Fig. 1.

Simpler variants of Delaunay that would work better in higher dimensional spaces are the Gabriel graph [17] and its heuristic variant caked XNN graph [18]. It was shown in [18] that 97% of the links in the optimal TSP path are included in the XNN graph. This makes it a suitable data structure for constructing TSP as we can find most links by much faster search within the graph.

LKH heuristic uses the alpha-nearness criterion to prioritize the candidate selection. However, the results in [6] showed that the simple Delaunay graph works slightly better, and it can be created in O(n log n) time.

3.2 Delaunay random edge

The first variant follows a randomized strategy. It starts with a random point and adds all its neighbors as available nodes to be chosen. The path is expanded by randomly choosing one of the available nodes and inserting it as part of the path. The node can be inserted into the path before or after the node which makes the node available. The less costly option is always chosen. The algorithm continues by adding new available nodes and iterates n times. With an efficient implementation, the time complexity is nearly linear in relation to the number of edges. Therefore, the time complexity is bounded by the time complexity of calculating the Delaunay graph, which is O(n log n).

3.3 Delaunay shortest edge

The second variant follows the greedy principle. It starts with a random point and expands the path by finding the shortest edge available in the neighborhood. The corresponding new node is then inserted to the path (see Fig. 2). After every step, the set of selected nodes expands by one and the number of candidates keeps increasing. The number of available edges fluctuates during the process as those edges that would create a loop are discarded. Assuming each node has k neighbors, the size of the neighborhood theoretically increases from k to nk. Fortunately, the nodes share the same neighbors and there are fewer than k new candidates, in practice.

For example, the first node has k=3 candidates in Fig. 2. The second node has four neighbors but only three of them are new. The added neighbor is also removed from the list of candidates, and the total size of the neighborhood increases only to k=4. After adding six nodes, the neighborhood size is still rather small, k=6. The development of the neighborhood size is demonstrated in Fig. 3 for the Santa dataset [6]. As we can see, the number of neighbors remains highly limited during the initialization. The maximum number of candidate edges reached 70k in this 1.4M node dataset, while the average number of edges during the whole process was 21,362. The storage and retrieval of edges can be done with a priority queue in O(E log E) time. However, the E inside the logarithm remains limited, which makes the initialization fast. In practice, the generation of the Delaunay graph is the limiting factor. The graph can be calculated in 10 seconds for 1.4M points, while the initialization takes about 2.5 seconds.

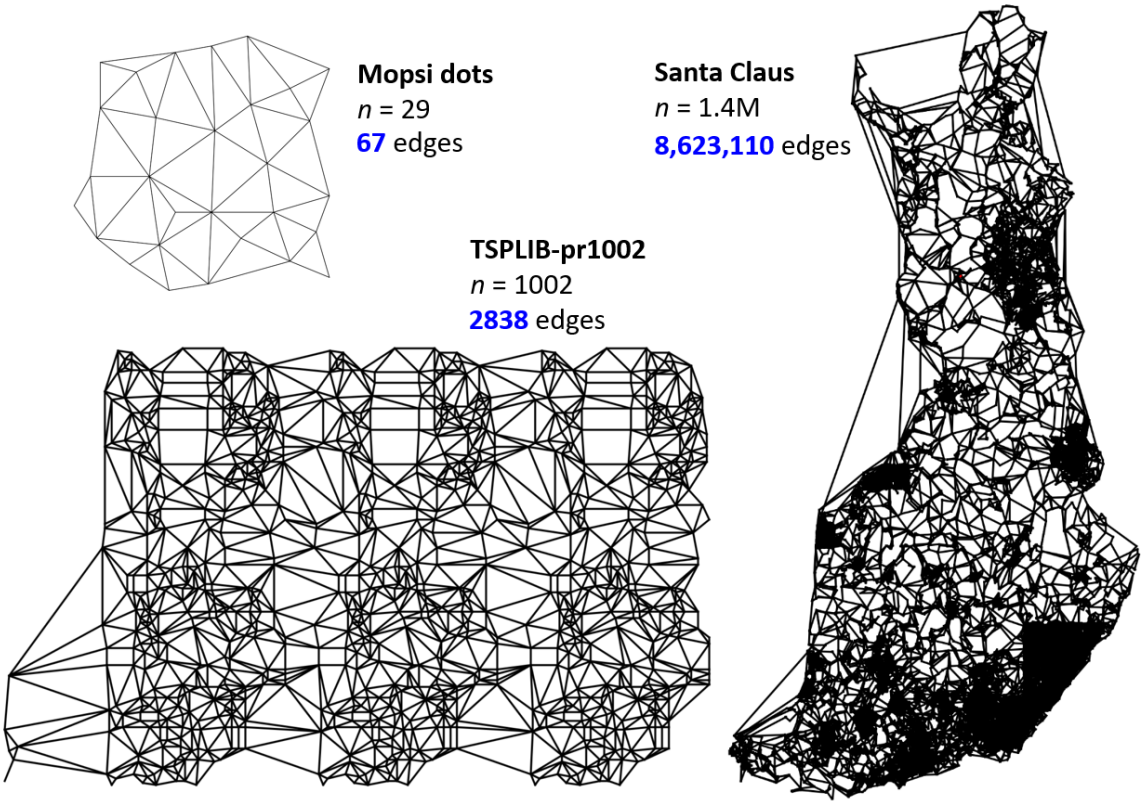

Figure 1. Examples of Delaunay graphs for used datasets.

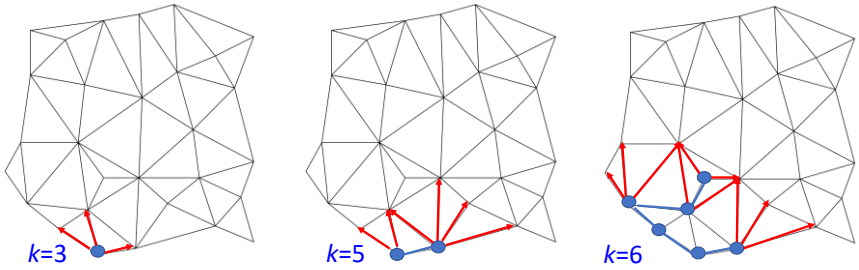

Figure 2. The process of the greedy expansion from a given seed point.

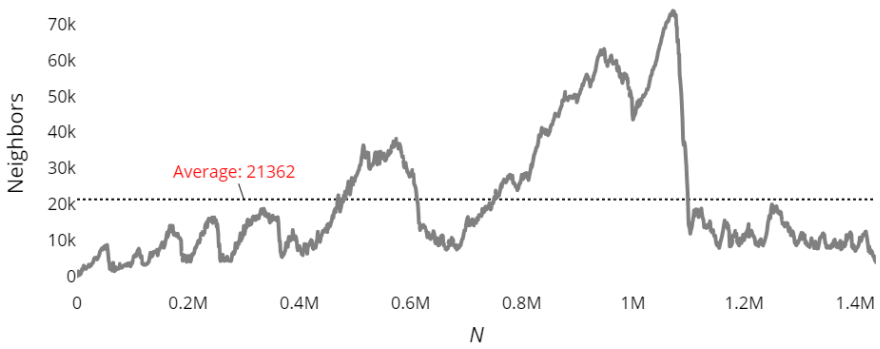

Figure 3. The development of neighborhood size during the initialization process (n = 1.4M).

4. RESULTS

The results of all methods are summarized in Table 1. The TSPLIB results include 10 selected instances from TSPLIB: d198, d657, fl3795, gr666, pcb442, pr76, pr1002, pr2392, u159 and u1060 [19]. The details of each dataset are presented in Table 2. The results are rather modest and do not meet the requirement of the 1% gap for branch-and-bound but can still be useful for initialization of local search. The results can be roughly divided into four categories: poor (>100% gap), weak (~45%), modest (13-24%) and good (2-7%). The results of random walk and Delaunay random edge are poor. The results of space-filling curves are somewhat better but clearly worse than the greedy heuristics, which achieve modest results. The only good results were achieved by Kruskal-TSP but by a slower algorithm. However, Kruskal-TSP is not the best method for larger datasets (TSPLIB) as Christofides provides better results. Both are slow, and the best speed and quality trade-off for the TSPLIB sets was achieved by the Boruvka algorithm.

The result of the Delaunay shortest edge algorithm is shown in Fig. 4. The total length is 150,962 km which is still far from the best result found for this data, which is 109,284 km [6]. The example also shows the weak point of greedy heuristics. There is an obvious pattern of the algorithm creating branches that start somewhere, circulate, and then return almost to the same locations where it started. The branches almost touch each other and indicate clear points of improvement. The result might not be the greatest from a local search point of view. Current testing showed that optimization starting from the greedy arrangement is a tiny bit better in practice, so some tuning is needed.

Table 1. Summary of the results for Dots and TSPLIB datasets. The percentages present the gap to the optimal solution. LKH is a package containing both initialization and optimization steps for reference.

Algorithm	**Ref.**	**Average**		**Worst**		**Time (ms)**
		Dots	TSPLIB	Dots	TSPLIB	TSPLIB
LKH	[2]	0.01%	0.08%	1.07%	0.54%	700787
Nearest neighbor	[3, 4]	9.25%	25.03%	52.70%	41.89%	61253
Nearest neighbor (fast)	[2]	6.74%	30.02%	59.76%	42.26%	301
Greedy (graph)	[2]	5.76%	21.17%	34.51%	29.76%	317
Moore	[9]	14.47%	45.10%	58.64%	62.35%	57
Sierpinski	[10]	13.28%	43.14%	70.20%	66.87%	58
Boruvka	[11]	6.53%	19.07%	57.95%	25.18%	307
Quick-Boruvka	[7, 12]	6.69%	23.17%	77.46%	58.80%	300
Christofides	[13, 14]	9.10%	15.04%	39.21%	18.39%	268535
Kruskal-TSP	[8]	2.66%	19.94%	28.40%	36.37%	184766
Walk	[2]	14.16%	114.59%	120.84%	210.77%	286
Delaunay shortest edge	(new)	9.87%	26.89%	51.61%	32.55%	710
Delaunay random edge	(new)	19.98%	110.97%	90.44%	491.95%	709

5. CONCLUSION

Greedy approaches can achieve modest results for all datasets, and many of them have fast variants by using neighborhood graph. Two new variants utilizing a Delaunay graph were proposed. However, the most promising for the smaller open-loop Dots instances turned out to be a simple heuristic, Kruskal-TSP, but it is currently slow. As a future work, we plan to modify a fast variant of it using the Delaunay graph. For the closed-loop TSPLIB instances, none of the algorithms were particularly good.

Table 2. Datasets used in this study.

Dataset	Type	Distance	Instances	Sizes
Dots[1]	Open loop	Euclidean	6449	5–31
TSPLIB10[1]	Closed loop	Euclidean / haversine	10	76–3,795
Santa[2]	Closed loop	Euclidean	1	1,437,195

[1] https://cs.uef.fi/o-mopsi/datasets/ [2] https://cs.uef.fi/sipu/santa/

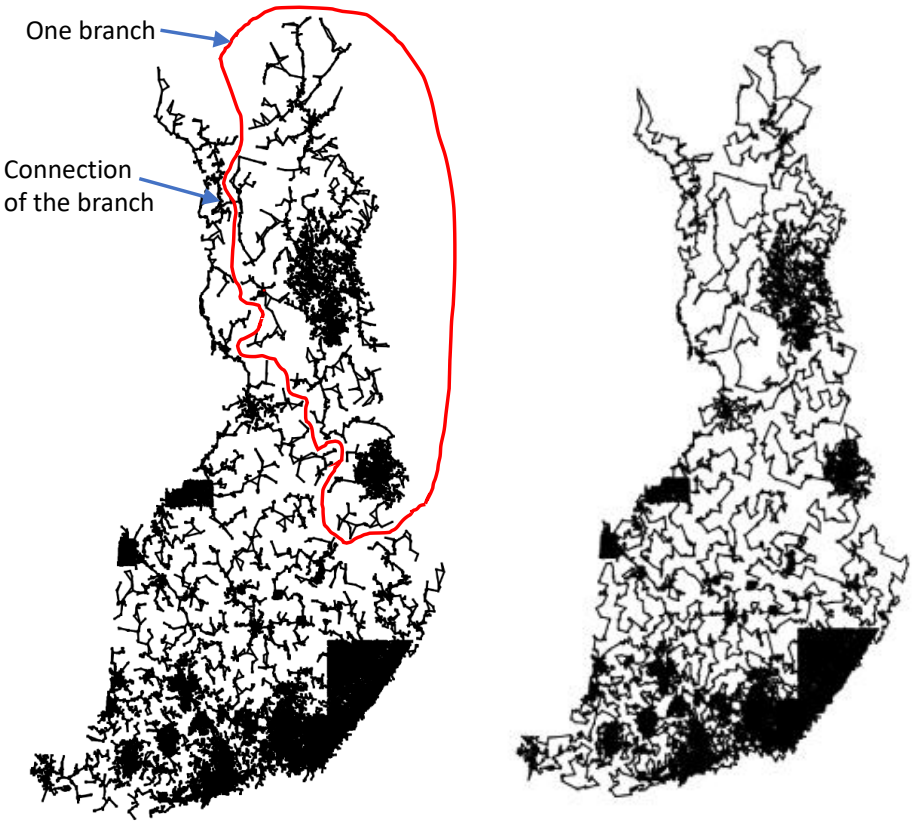

Figure 4. The process of the greedy expansion from a given seed point (150,962 km) (left), and a good result calculated by LKH in Santa competition [6] (109,284 km) (right). Some similarities between the two structures can be seen.

REFERENCES

[1] Applegate, D., Bixby, R.E., Chvátal V., and William, J.C. (1999). Concorde: a code for solving traveling salesman problems. Available at: http://www.tsp.gatech.edu/concorde.html.

[2] Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur. J. Oper. Res. 126 (1), 106–130. doi:10.1016/s0377-2217(99)00284-2

[3] Applegate, D.L.; Bixby, R.E.; Chavatal, V. (2006); Cook, W.J. The Travelling Salesman Problem, a Computational Study; Princeton Univesity Press: Princeton, NJ, USA.

[4] Kizilates, G.; Nuriyeva, F. (2013). On the nearest neighbor algorithms for the traveling salesman problem, Advances in Computational Science, Engineering and Information Technology; Springer: Germany; Volume 225.

[5] Bentley J.L. (1992). Fast Algorithms for Geometric Traveling Salesman Problems, ORSA Journal on Computing, Vol. 4, 4, pp. 387-411.

[6] Mariescu-Istodor R. and Fränti P. (2021). Solving large-scale TSP problem in 1 hour: Santa Claus Challenge 2020, Frontiers in Robotics and AI, 8, 1-20.

[7] David S. Johnson and Lyle A. McGeoch (2007). Experimental Analysis of Heuristics for the STSP. Combinatorial Optimization, vol 12.

[8] Fränti, P., and Nenonen, H. (2019). Modifying Kruskal algorithm to solve open loop TSP. Multidisciplinary International Scheduling Conference (MISTA), Ningbo, China, 12–15; pp. 584–590.

[9] Moore E.H. (1900). On certain crinkly curves. Trans. Amer. Math. Soc. N1, pp. 72–90.

[10] Sierpinski W. (1912). O pewnej krzywej wypetniajacej kwadrat. Sur une nouvelle courbe continue qui rempllt toute une aire plane. Bulletin de l'Acad, des Sciences de Cracovie A. 463-478.

[11] Borůvka, O., (1926). O jistém problému minimálním [About a certain minimal problem]. Práce Mor. Přírodověd. Spol. V Brně III (in Czech and German). 3: 37–58.

[12] Applegate, D., Cook, W. & Rohe, A. (2003). Chained Lin-Kernighan for large traveling salesman problems. INFORMS Journal on Computing, 15, 82–92.

[13] Christofides, N. (1976) Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem; Report 388; Graduate School of Industrial Administration, CMU: Pittsburgh, PA, USA.

[14] Goodrich, M.T.; Tamassia, R. (2015). 18.1.2 The Christofides Approximation Algorithm. In Algorithm Design and Applications; Wiley: Hoboken, NJ, USA, pp. 513–514.

[15] P. Fränti, T. Nenonen and M. Yuan, (2021). Converting MST to TSP path by branch elimination, Applied Sciences, 11 (177), 1-17.

[16] Platzman, L.K.; Bartholdi, J.J., III (1989). Space filling curves and the planar traveling salesman problem. Journal of the Association for Computing Machinery. 36 (4): 719–737.

[17] Gabriel, K.R., Sokal, R.R. (1969): New statistical approach to geographic variation analysis. Syst. Zool. 18, 259–278.

[18] Fränti P., Mariescu-Istodor R., and Zhong C. (2016). XNN graph, Joint Int. Workshop on Structural, Syntactic, and Statistical Pattern Recognition (S+SSPR 2016), Merida, Mexico, LNCS 10029, 207-217.

[19] Reinelt, G. (1991). A traveling salesman problem library. INFORMS J. Comput., 3, 376–384.