Understanding Geometry in the Euclidean Travelling Salesman Problem using Well Separated Pairs

Ishan Sen
isen@umd.edu

Abstract

The well-separated pair decomposition (WSPD) of a set of points offers great insight to the geometric nature of this information. This decomposition of points enables efficient approximations to computationally expensive problems such as the Travelling Salesman Problem (TSP). We investigate the use of various polynomial-time approximate solutions to the TSP on top of the well-separated decomposition of points in Euclidean space. Based on empirical results we identify settings in which specific algorithms may lend themselves well, and propose metrics by which we decide on an approximation algorithm.

1 Introduction

The Travelling Salesman Problem (TSP) [23] asks, given a set of points in a d-dimensional space $\mathbb{R}^d$, the shortest path that visits every city while also starting and ending at the same point. This is a problem that occurs in many real-world logistics and planning scenarios, such as vehicle route planning [12], manufacturing microchips [10], machine scheduling [22], etc. If a human were to look at a map and attempt to create such a path, they may come close to the optimal solution quickly. Computationally however, this remains an incredibly difficult optimization problem to solve. In fact, the TSP is said to be NP-hard, meaning it cannot be solved in polynomial time.

To find the optimal solution for a set of n points, one needs to enumerate all possible paths and pick the one with lowest cost resulting a computational complexity of $O(n!)$. Researchers have devised a host of approximate algorithms that aim to find a solution that is very close to the optimal solution in polynomial time. Yet, many of these solutions do not account for potential geometric insights one may find when looking at these points and serve as a monolithic solution to the problem. Instead, can we use specific heuristics or properties present in our data to adaptively pick a suitable polynomial-time approximation?

To better understand the geometry of the travelling salesman problem, we look to the well-separated pair decomposition [6] of our set of points. This decomposition allows us to break down the problem space into meaningful sub-problems, and attempt to solve the problem locally on each well-separated pair before looking at the whole picture. Well-separated pairs have offered insight into approximately solving other well-known computationally expensive problems such as approximating minimum spanning trees [16] and shortest paths between any points [26].

In this work, we investigate the use of the well-separated pair decomposition to approximately solve the travelling salesman problem. In particular, we leverage geometric properties of our data on a Euclidean plane to support existing polynomial-time approximations of the TSP. We compare these approximations to those that do not leverage such information and explore results on small sets of data.

2 Definitions and Related Works

2.1 Well-Separated Pair Decomposition

In this subsection we go over definitions, computation, and applications of the Well-Separated Pair Decomposition of a set of points.

2.1.1 Definitions

To understand the Well-Separated Pair Decomposition (WSPD), we must first understand what it means for two sets to be "Well Separated" [6]:

Definition 1: *Let two finite sets of points A and B be subsets of $S \subset \mathbb{R}^d$ and let $s > 0$ be a real number. The sets A and B are well separated with respect to s given there are two disjoint balls C_A and C_B of radius r such that:*

1. *C_A and C_B have the same radius r*
2. *C_A contains all points in A*
3. *C_B contains all points in B*
4. *C_A is at least s times the distance r away from C_B*

The real number s is called the separation ratio.

This can be seen in Figure 1 below.

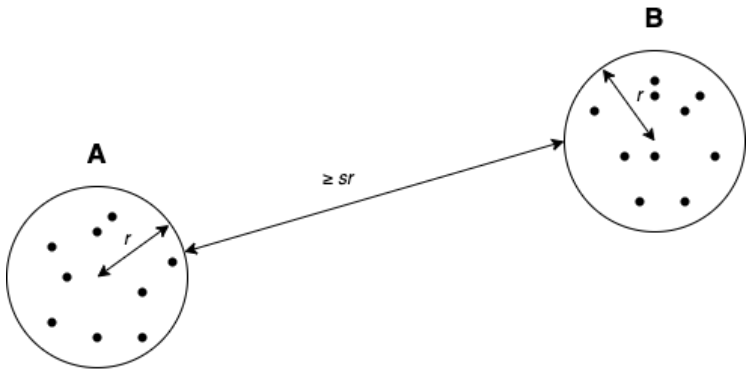

Figure 1: Example of Well-Separated Pairs in the $\mathbb{R}^2$ space. All points in set A and B are contained in circles with radius r and are at least of distance sr away from each other. The parameter s is typically a predetermined parameter.

Building off of this, the Well-Separated Pair Decomposition (WSPD) can be defined as follows:

Definition 2: *Let $S \subset \mathbb{R}^d$ be a set of n points, and let $s > 0$ be a real number. A WSPD of S, with respect to s, is a sequence $\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_m, B_m\}$ of pairs of non-empty subsets of S for some integer m, such that:*

1. *for each $i \in [m]$, A_i and B_i are well-separated with respect to s*
2. *for any two distinct points $p, q \in S$, there is exactly one index $i \in [m]$, such that:*
   (a) *$p \in A_i$ and $q \in B_i$, or*
   (b) *$p \in B_i$ and $q \in A_i$*

In essence, this decomposition of points generates unordered pairs that act as a cover of all our n points in S such that each pair is well-separated, as defined in Definition 1.

2.1.2 Generating the Well-Separated Pair Decomposition

In order to generate well-separated pairs we first build a point region quadtree (PR-quadtree) to store our input data points. This generation works by recursively subdividing our space into four square quadrants as we add points to the data structure. We split on a point if the bucket it is in has more points than a user defined limit. An example of a final PR-quadtree with bucket limit of 1 can be seen as follows:

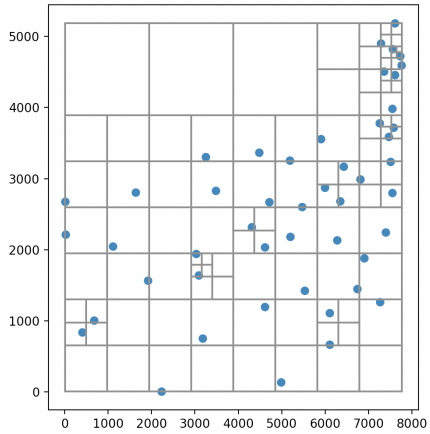

Figure 2: Example of a PR-quadtree with 48 points and a bucket limit of 1.

Next we detail how we compute the well-separated pair decomposition. At a high level, we compare pairs of non-empty nodes and check if they can be decomposed into well-separated pairs with a separation factor of s. If so, we store these pairs into our WSPD dictionary. Otherwise, we identify the larger cell and subdivide this cell into children and investigate each child recursively with the smaller cell. Clearly, this algorithms runs proportional to the number of blocks and is proven to have a space complexity of $O(n)$. An example of a WSPD of a more simple case with $n = 9$ and a separation factor of $s = 1$ can be seen in the following figure:

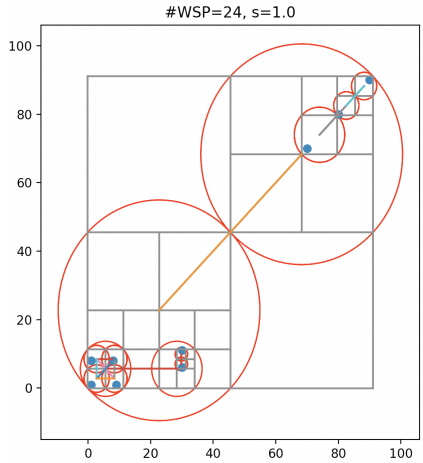

Figure 3: Example of the WSPD of 9 points with a separation factor of 1

This powerful geometric decomposition of points can be done in $O(nlog(n))$ time[1] and has lead to many developments in polynomial-time approximate algorithms for difficult problems.

[1] `https://www.cs.umd.edu/class/spring2020/cmsc754/Lects/lect16-wspd.pdf`

2.1.3 Applications of Well-Separated Pair Decomposition

The WSPD has been used in various settings to optimally solve problems such as the closest pair problem, k closest pairs problem, all-nearest neighbors problem [8], approximate minimum spanning tree problem[16; 7], and approximate shortest paths [26]. For the application to k-nearest neighbors, Callahan et. al are able to identify this in $O(kn)$ time by using the WSPD to prune non-useful sets of nodes [8]. Of particular interest is the use of WSPD in approximating the minimum spanning tree of a set of points by Li et. al. Here, the use of WSPs enables data storage optimizations as well as algorithmic optimizations by pruning options not included by the WSPD.

2.2 Travelling Salesman Problem

The Travelling Salesman Problem (TSP) [25; 23] is an age-old combinatorial optimization problem that has a variety of real-world applications. A simple description of the problem is as follows:

Definition 3: Given n cities out of a set of cities C with distances between two cities $d(c, c')$ for $c \in C$, where the triangle inequality holds, the goal is to find a tour of the cities that minimizes a tour length. A tour of $c_1, c_2, ..., c_n$ has a total length of $\sum_{i=1}^{n-1} d(c_i, c_{i+1}) + d(c_n, c_1)$.

2.2.1 Solving the TSP

To find the optimal solution, one must check every possible combination of paths, which can result in a complexity of $O(n!)$. This is a well studied problem however, and as such, there are a host of algorithms that approximately solve it in a more reasonable time. A general breakdown of how some algorithms aim to solve it are as follows:

1. Heuristic-Based Approximation
   (a) Nearest Neighbor [4; 24]
   (b) Love & Norback [20]
   (c) Lin-Kernighan (3-opt) [17]
   (d) Minimum Spanning Tree + shortcuts (Christofides) [15]
   (e) Divide & Conquer [9]
2. Enumeration
   (a) Cutting Planes [21]
   (b) Branch & Bound [2]
   (c) Dynamic Programming [3]
3. Learning Based
   (a) Deep Learning [18; 14]
   (b) Genetic Algorithms [5]

Heuristic Approximation and Enumeration based solutions have been studied extensively and are used in similar problems such as shortest-path finding. These approximation algorithms run in polynomial time and using polynomial storage [19]. Other modern techniques form a blend of these approaches to approximate a more optimal solution while maintaining low computational costs.

While the focus of this paper looks at heuristic and enumeration based solutions to the TSP, we acknowledge the use of learning based solutions, such as using graph convolutional networks [14] and and Ant Colony Optimization [13] as powerful and efficient methods for TSP problems.

2.3 Datasets

In this work, we use data generated by TSP instances found in VLSI data [2] and Country data [3]. To properly evaluate TSP approximation algorithms, these datasets know the optimal solution OPT.

[2] https://www.math.uwaterloo.ca/tsp/vlsi/index.html
[3] https://www.math.uwaterloo.ca/tsp/world/countries.html

3 Approximation Algorithms

In this section we take a closer look at existing algorithms for the TSP, and the potential use of WSPD to further augment these. We can classify an approximate algorithm as $\alpha - OPT$ if it is at most α times the optimal solution OPT in worst case.

3.1 Brute Force with WSP Pruning

The Brute Force method to solve the TSP looks to enumerate all $(n-1)!$ possible paths and picks the one with lowest cost. The application of WSPD here is to prune paths that are clearly redundant by looking at edges between well-separated pairs. We still build permutations for all possible paths for each point that is not well separated, and then connect the group by an edge to the remaining graph. After pruning undesirable choices, we pick the minimum cost permutation. At a high level the steps for this are as follows:

1. Form a well-separated dictionary for each point
2. For each point p from point set P:
   (a) Identify set S, all of which p is well-separated from
   (b) Add path permutations for p with all $d \in P \setminus S$ that have not yet been visited (pruning "inefficient" paths)
   (c) Add a point $s \in S$ to path (well separated from points in above step)
   (d) Do the same process with point s until all points have been visited
3. Evaluate all path permutations and return minimum cost path

What this does is similar to other heuristic-based approaches, where we are able to break down the problem space into smaller sub-problems, and solve each individually and cheaply join them to each other. However, in situations where each point is well-separated from the majority of the rest, it is clear that the algorithm becomes exponential in the worst case. This pruning method still has a time complexity of $O(n!)$. However, we still use this as a baseline metric to compare other approximate strategies. Results for this can be seen in Table 1.

3.2 Nearest Neighbor Heuristic

The nearest neighbor heuristic approximation [4; 24] is a greedy algorithm that, given a starting point, creates paths by picking closest point to it as the next step. This is a simple algorithm to implement yet naïve in nature, as it may leave a neighborhood of points by greedily selecting a nearest neighbor. The algorithm on its own has a complexity of $O(n^2)$. This heuristic-based approach has shown to result in an approximation of being within 25% of the Held-Karp lower bound [19].

To prevent the algorithm from erroneously leaving a neighborhood of our points, we can use the WSPD to identify neighborhoods of points that we want to first travel through, before leaving the neighborhood. As such, we can apply a constraint on the nearest neighbor heuristic to first exhaust all points in our WSP set before continuing. At a high level, this algorithm is as follows:

1. Form a well-separated pair decomposition of all points P
2. For each set S in our decomposition (sub-problems)
   (a) If $|S| < t$, for some threshold t, identify brute-force solution to points in S, otherwise use nearest neighbor heuristic
   (b) Identify entry and exit points to connect to other sub-problems
3. Connect each sub-problem to form a tour as our solution

Connecting sub-problems to each other is done by looking at the minimum projection of points to a line connecting the sub-problems. We identify entry- and exit-points for each sub-problem in this

manner. Identifying the optimal connections is still the TSP, and we could solve this recursively. For simplicity, we do so in a brute-force manner in the current implementation of the algorithm.

This approximation strategy has a time complexity of $O(m^2)$ for sub-problems of size $m > t$ and complexity of $O(t!)$ for those with size smaller than our threshold. We can pick such a threshold to be small such that the exponential complexity that arises from this is low. Letting k be the number of sub-problems, connecting these has a complexity of $O(k!)$. Our overall complexity of the algorithm is thus $O(k(m^2) + k!)$. Guarantees on optimality are yet to be analyzed. Results for this can be seen in Table 1 denoted as "NN (WSP)".

3.3 Minimum Spanning Tree (Christofides

The Christofides algorithm, also known as the $3/2$-OPT algorithm, is a powerful approximation of the solution for the TSP by using a minimum spanning tree as its basis [11]. The original algorithm has a worst-case complexity of $O(n^3)$ [11] and results in a $3/2$-OPT approximation of the solution given an optimal minimum spanning tree.

The use of WSPD here is in approximating the minimum spanning tree of our points, and using this to approximate the solution to the TSP. As such, the algorithm is as follows:

1. Compute minimum spanning tree G of our points P
2. Compute a minimum-cost matching M on the set of odd-degree vertices of the MST G
3. Add M to G to form an Eulerian graph
4. Find an Eulerian tour T of our Eulerian Graph
5. Convert T into a Hamiltonian tour H by travelling the tour T and skip vertices that were already visited
6. Return our tour H to solve the TSP

The computational complexity of approximating the MST is $O(nlogn)$ [16], and the Christofides algorithm, as mentioned, has a complexity of $O(n^3)$. As such, the resulting complexity is still $O(n^3)$, which is still much better than the brute-force solution. We do not have an analysis on optimality here, but I suspect, given some error ϵ on the MST, that the algorithm is $3\epsilon/2$-OPT. Results for this can be seen in Table 1.

3.4 Notes on Grid Snapping

When constructing our quadtree, if points are very close to each other (i.e. $d(p_1, p_2) < 1$) we must continuously subdivide our space until they are split. This can become expensive from a time and space perspective. One solution to this would be to snap points to integers. When we snap points to a grid in the 2-D plane, a vertex is moved by at most $\sqrt{2}/2$ (diagonal in a 1x1 unit). However, an edge length is changed by at most $2\sqrt{2}/2$, i.e. $\sqrt{2}$ as each end point has moved by at most $\sqrt{2}/2$. Given that T is some solution to the original problem and T_s is the same solution using snapped points, we clearly see that $T_s \leq T + n\sqrt{n}$ [1]. In some cases, where the distances are very long, the error term of $n\sqrt{n}$ may be negligible, but this would be a case-by-case scenario where one would use this.

4 Results

In this section we go over some empirical results of using the aforementioned algorithms and compare tour lengths to the optimal solution. We also compare the heuristic-based algorithm to a non-WSP method to observe benefits gained from using the WSPD. The other methods are the Nearest Neighbor method without using the WSPD, Divide & Conquer method, and Genetic algorithm.

Table 1 shows tour lengths for 4 different datasets using the heuristic algorithms that are described in section 3, as well as some additional approximate algorithms. The number in the dataset name indicates the number of points in it. the "att48", "uy734" and "pr1002" datasets are geographic datasets, while the "xqf131" is on VLSI data. The numbers in the cell are the tour length followed by the approximation factor of the optimal solution. We also highlight the best approximation length for each dataset.

Table 1: Tour Lengths for Approximation Algorithms with Approximation Factor

Dataset	Optimal	NN (WSP)	NN	Christofides	Divide & Conquer	Genetic
att48	33,523	37,718 *(+13%)*	40,526 *(+21%)*	38,777 *(+16%)*	36,957 *(+10%)*	**36,385 *(+9%)***
xqf131	564	684 *(+21%)*	709 *(+26%)*	923 *(+64%)*	729 *(+29%)*	**677 *(+20%)***
uy734	79,114	**94,430 *(+19%)***	102,594 *(+30%)*	270,743 *(+242%)*	108,769 *(+37%)*	102,273 *(+29%)*
pr1002	259,045	328,148 *(+27%)*	**315,596 *(+22%)***	519,286 *(+100%)*	364,564 *(+41%)*	315,596 *(+22%)*

5 Discussion

5.1 Algorithms using WSP

Strictly comparing approximation algorithms that use WSPD, it seems like the nearest neighbor heuristic performs better than the Christofides algorithm, likely due to the propagation of error from approximating the MST. However, the deviation of Christofides from the NN algorithm seems to be very narrow on the dataset with 48 points. Perhaps on smaller sub-problems, the Christofides algorithm may perform better.

The Christofides algorithm seems to have very large deviations from the optimal solution if one were to look at the "uy734" and "pr1002" datasets, which have a greater number of points than the other two. Reconciling this with the distribution of the points, it seems like the points are somewhat uniformly distributed, with a few clusters of points. Perhaps using other heuristics to describe the data may be useful to classify and further investigate types of data on which this approximation method is not as efficient.

5.2 Comparison of Nearest Neighbor Heuristic

We implement the nearest-neighbor heuristic approximation to the TSP using the WSPD of our points, and without. We find that in the "uy734" dataset our algorithm using the WSPD outperforms the general NN algorithm by 21%, but fails to do so on the "pr1002" dataset with a 5% difference. Further analysis on the data would be needed to better understand why the algorithms have differing results, but it may be due to luck using the non-WSP algorithm or that our WSP algorithm requires further fine-tuning of parameters. It does seem like, up until the graph with 1002 nodes, that the WSP decomposition leads to better approximations of the solution than the non-WSP method. This may be an indicator of how variance in points may lead to challenges in using the WSPD.

5.3 Other performance comparisons

We find that the genetic algorithm outperforms all other algorithms on the smaller datasets "att48" and "xqf131". They marginally beat the NN algorithm using WSP, however, further tuning of both algorithms may affect the final result. This method falls off on larger datasets though, and after 1200 iterations on the baseline path (NN no WSP), we do not see major improvements.

The divide & conquer method, which recursively subdivides the points in a naïve manner and optimizes the merge of paths, seems to perform worse, on average, than heuristic-based methods. This validates the thought that using geometry to decompose our point-space is a better than trying to evenly divide the point space and joining points that way.

6 Future Work

There is still much work to be done to better understand how we can leverage the WSPD of our points for algorithmic optimizations. This can be done in three fronts, 1) improvements in sub-problem solving, 2) improvements in sub-problem identification, and 3) computing additional heuristics to guide the selection of algorithms.

From a sub-problem solving perspective, this can still be extended to using genetic algorithms or swarm techniques on a small number of points, where convergence happens quickly, or using other heuristic-based approaches such as the Christofides or other algorithms. The next step is to optimize the connection of sub-problems to one another, as this problem is another instance of the TSP.

Looking at improvements in sub-problem identification, alternative methods of decomposing our point-space into sub-problems exist, such as using k-means clustering. Other works uses this as means of decomposing the point space, so a logical step would be to compare the WSPD to these methods. Alternatively, one could potentially use the geometric decomposition of points to guide better centers for k-means.

Finally, work can be done on using WSPD or other heuristics to guide the choice of our approximation algorithm. There are certainly applications of WSPD in pruning or scoping the problem space, but other global heuristics may be useful in our approximation selection. One thing to look at could be a notion of balance between subgroups, and know that one may need to further subdivide a group using a different method or separation factor to result in more manageable sub-problems.

7 Conclusion

Well-separated pairs offer strong geometric insights and clearly has potential in geometric problems such as the travelling salesman problem. Particularly, this offers a logical subdivision of points that allows us to work on a smaller problem space using potentially expensive algorithms. We find that using WSP to divide our points into geometric sub-problems yields mixed results on the TSP when using the nearest neighbor heuristic. Using other approximate solutions to solve sub-problems may yield additional optimizations. This work further motivates research into exploring heuristics of our data beyond its WSPD to guide the use of existing algorithms, such as distributional information or alternative decompositions of points. Overall, we find that the geometry of points do indeed play a big role in approaches to solving the TSP.

References

[1] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, sep 1998. doi: 10.1145/290179.290180

[2] E. Balas and P. Toth. Branch and bound methods for the traveling salesman problem. 1983.

[3] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962.

[4] M. Bellmore and G. L. Nemhauser. The traveling salesman problem: A survey. *Oper. Res.*, 16:538–558, 1968.

[5] H. Braun. On solving travelling salesman problems by genetic algorithms. In *International Conference on Parallel Problem Solving from Nature*, pp. 129–133. Springer, 1990.

[6] P. B. Callahan. Dealing with higher dimensions: the well-separated pair decomposition and its applications. 1995.

[7] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete*

Algorithms, SODA '93, p. 291–300. Society for Industrial and Applied Mathematics, USA, 1993.

[8] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to ¡i¿k¡/i¿-nearest-neighbors and ¡i¿n¡/i¿-body potential fields. *J. ACM*, 42(1):67–90, jan 1995. doi: 10.1145/200836.200853

[9] G. Cesari. Divide and conquer strategies for parallel tsp heuristics. *Computers & operations research*, 23(7):681–694, 1996.

[10] D. CHAN and D. MERCIER. Ic insertion: an application of the travelling salesman problem. *International Journal of Production Research*, 27(10):1837–1841, 1989. doi: 10.1080/00207548908942657

[11] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[12] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Manage. Sci.*, 6(1):80–91, oct 1959. doi: 10.1287/mnsc.6.1.80

[13] M. Dorigo and L. M. Gambardella. Ant colonies for the travelling salesman problem. *biosystems*, 43(2):73–81, 1997.

[14] C. K. Joshi, T. Laurent, and X. Bresson. An efficient graph convolutional network technique for the travelling salesman problem, 2019. doi: 10.48550/ARXIV.1906.01227

[15] G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[16] C. Li and A. D. Mount. Euclidean minimum spanning trees based on well separated pair decompositions. 2014.

[17] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[18] S. Miki, D. Yamamoto, and H. Ebara. Applying deep learning and reinforcement learning to traveling salesman problem. In *2018 international conference on computing, electronics & communications engineering (ICCECE)*, pp. 65–70. IEEE, 2018.

[19] C. Nilsson. Heuristics for the traveling salesman problem. *Linkoping University*, 38:00085–9, 2003.

[20] J. P. Norback and R. F. Love. Geometric approaches to solving the traveling salesman problem. *Management Science*, 23(11):1208–1223, 1977.

[21] 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. doi: 10.1137/1033004

[22] J.-C. Picard and M. Queyranne. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations research*, 26(1):86–110, 1978.

[23] G. Reinelt. The traveling salesman: computational solutions for tsp applications. 1994.

[24] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, 6(3):563–581, 1977.

[25] S. Sangwan. Literature review on travelling salesman problem. *International Journal of Research*, 5:1152, 06 2018.

[26] J. Sankaranarayanan and H. Samet. Distance oracles for spatial networks. In *2009 IEEE 25th International Conference on Data Engineering*, pp. 652–663. IEEE, 2009.