Studying the Wiener Index for spatially embedded graphs

Atishaya Maharjan
maharjaa@myumanitoba.ca

University of Manitoba— August, 2025

Abstract

We study the minimization problem for Wiener index of spatially embedded graphs in the Euclidean setting. Motivated by the PTAS of the metric embedding approach of Abuaffash et al. [1], we aim to find and implement a simple divide and conquer heuristic algorithm. Our empirical evaluation on random point sets with 4 to 7 points show that the algorithm achieves approximation ratios between 1.00 and 1.52, with $O(n^2 \log(n))$ time. Finally, we also study and give an intuition on why the open problem for the unconstrained problem on proving hardness of the unconstrained version of Euclidean Spanning Tree Minimizing the Wiener Index is difficult to show and hence still open.

1 Introduction

The Wiener index is a topological index that is defined as the sum of distances between all pairs of vertices in a graph. It is widely used in chemistry to predict the properties of molecules based on their structure. In this paper, we will study the Wiener index for spatially embedded graphs, which are graphs that can be drawn in a Euclidean space.

Mathematically, the *Wiener index* of a weighted graph $G = (V, E)$ is the sum of the distances between all pairs of vertices in G:

$$W(G) = \sum_{u,v \in V} \delta_G(u, v)$$

Where $\delta_G(u, v)$ is the weight of the shortest (minimum-weight) path between vertices u and v in the graph G. Formally, the Wiener index was introduced by the chemist Harry Wiener in 1947 [4]. It and its variations have several applications in chemistry, biology, and network analysis.

2 Preliminaries

Let P be a set of n points in the Euclidean plane $\mathbb{R}^2$. The *Euclidean distance* between two points $p_i, p_j \in P$ is defined as:

$$|p_i p_j| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

We say that the points are in *general position* if no three points of P are collinear. This assumption avoids degenerate configurations and is standard in geometric optimization.

Let $G = (P, E)$ be the complete graph on the set of points P, where the edge set E consists of all pairs of points in P. The weight of an edge $(p_i, p_j) \in E$ is defined as the Euclidean distance between the points p_i and p_j and is denoted as $w(p_i, p_j) = |p_i p_j|$. More generally, a *weighted graph* is a graph $G = (V, E)$ with a function $w : E \to \mathbb{R}_{\geq 0}$ assigning nonnegative weights to edges [2].

A *spanning tree* of G is a connected, acyclic subgraph $T = (P, E_T)$ that contains all vertices in P. Standard combinatorial facts about trees will be used:

- A tree on n vertices has exactly $n - 1$ edges.
- In a tree, there is exactly one simple path between any pair of vertices.

For a tree T, let $\delta_T(p, q)$ denote the weight of the unique path between p and q in T, i.e., the sum of edge weights along that path.

Definition 2.1 (Wiener Index)**.** *The* Wiener index *of a tree T is the sum of pairwise distances:*

$$W(T) = \sum_{p,q \in P} \delta_T(p, q).$$

For a vertex $p \in P$, we also define

$$\delta_p(T) = \sum_{q \in P} \delta_T(p, q),$$

the total distance from p to all other points in T.

Definition 2.2 (Weighted Wiener Index)**.** *If each vertex $v \in P$ has a weight $w(v) \geq 0$, the* weighted Wiener index *is*

$$\mathrm{WWI}(T) = \sum_{\{u,v\} \subseteq P} w(u)w(v)\, \delta_T(u, v).$$

Remark 1. The distinction between unweighted and weighted Wiener index will be important for capturing combinatorial structure in reductions and analyzing how distances contribute to global sums.

3 Computing the Wiener index for spatially embedded graphs

Abuaffash et al. [1] showed the following theorems:

Theorem 3.1 (Abuaffash et al. 2023)**.** *The spanning tree of P that minimizes the Wiener index is planar.*

Intuition. Any non-planar edge crossing in a spanning tree introduces unnecessarily long detours between vertices across the crossing. By rerouting edges to avoid crossings, one can always obtain a planar tree with no increase in the Wiener index. Thus, optimality forces planarity [1].

Theorem 3.2 (Abuaffash et al. 2023)**.** *There exists a $O(n^4)$ time algorithm that constructs a spanning tree of P that minimizes the Wiener index.*

Intuition. The algorithm essentially tries all possible candidate structures by dynamic programming over point partitions, leveraging the planarity property [1]. The $O(n^4)$ bound arises from enumerating subsets and possible edge choices in a controlled way.

Theorem 3.3 (Abuaffash et al. 2023)**.** *Given a cost W and a budget B, computing a spanning tree of P whose Wiener index is at most W and whose total weight of the paths from each point to every other point is at most B is weakly NP-hard.*

Intuition. The reduction encodes the classical PARTITION problem into a geometric gadget construction. The weight bound B forces binary choices in the gadget design, corresponding to selecting a subset that balances the partition. This makes the decision problem computationally intractable [1].

Theorem 3.4 (Abuaffash et al. 2023)**.** *The Hamiltonian path of P that minimizes the Wiener index is not necessarily planar and computing it is NP-hard.*

Intuition. Unlike trees, Hamiltonian paths must visit all points sequentially. Forcing such a linear order can require edge crossings even in the optimal case. Moreover, reducing from known NP-hard Hamiltonian path variants shows that minimizing the Wiener index in this restricted setting inherits the hardness [1].

We tackle the problem of computing the Wiener index for points in general position as well as give an approximation-heuristic with a divide and conquer greedy algorithm for the problem of computing the Hamiltonian path of P that minimizes the Wiener index.

4 Complexity and Approximation Results

The Wiener index minimization problem is closely related to the *Minimum Routing Cost Spanning Tree (MRCST)* problem:

Definition 4.1 (MRCST)**.** *Given a weighted undirected graph $G = (V, E)$ with nonnegative weights, find a spanning tree T minimizing*

$$c(T) = \sum_{u,v \in V} \delta_T(u, v).$$

It is known that MRCST is NP-complete, though a PTAS exists [2]. In the geometric setting, Abuaffash et al. showed [1]:

- The spanning tree of P minimizing the Wiener index is planar.
- When P is in convex position, the problem can be solved in polynomial time.
- The Hamiltonian path minimizing the Wiener index is not necessarily planar and is NP-hard to compute.
- There exists a $(1 + \varepsilon)$-approximation algorithm for the Hamiltonian path version, based on metric embeddings.

4.1 A Divide-and-Conquer Approximation Algorithm

We designed a recursive divide-and-conquer algorithm for approximating the Wiener-minimizing Hamiltonian path. We did this because the implementation for the metric embeddings is hard to understand and implement. Our hope was for another PTAS, unfortunately, we were not able to fully analyze this algorithm:

Algorithm 1 DivideAndConquerWiener(P, variant)

```
procedure DivideAndConquerWiener(P, depth = 0, maxDepth = 10)
    if |P| ≤ 4 or depth > maxDepth then
        return BruteForceHamiltonianCycle(P)
    end if
    (P_L, P_R) ← PartitionPoints(P, variant)
    π_L ← DivideAndConquerWiener(P_L, depth + 1)
    π_R ← DivideAndConquerWiener(P_R, depth + 1)
    return MergeCycles(π_L, π_R)
end procedure
```

The PartitionPoints procedure implements either bounding box bisection (splitting along the longer axis) or median bisection (splitting along the more balanced coordinate). The MergeCycles procedure finds the pair of connection points (i, j) that minimizes the cost of joining the two cycles into a single cycle.

4.1.1 Time Complexity

Each level of recursion partitions the points and solves two subproblems, giving $O(\log n)$ levels. At each level, the merge operation examines $O(n^2)$ pairs of connection points. The base case brute force on 4 points is $O(1)$. Thus, the overall complexity is $O(n^2 \log n)$.

4.1.2 Intuition and Theoretical Guarantees

The intuition behind this algorithm is very similar to KD-trees [3] with how the place is partitioned and recursively dealt with.

Although difficult to analyze theoretically (initially conjectured as a $O(\log n)$-approximation), our empirical experiments suggest it performs well in practice. Implementations and experimental data are available at github. The reason behind why this was really difficult to analyze is that the Wiener index is a global

metric; it is extremely hard to localize changes and show that it does not affect the global Wiener index overall.

4.2 Empirical Evaluation

We implemented our divide-and-conquer algorithm and evaluated it against brute force optimal solutions on instances with $4 \leq n \leq 7$ points. We tested two point generation models: points in *convex position* (vertices of a convex polygon) and points in *general position* (randomly distributed). For each configuration, we ran 3 trials with different random seeds, generating 24 total experiments. All experiments were conducted using Python with multiprocessing for the brute force baseline.

We implemented two partitioning strategies for the divide-and-conquer approach:

- **Bounding box bisection** (`divide_conquer`): Partitions the point set along the longer dimension of its axis-aligned bounding box, splitting at the midpoint of that dimension.
- **Median bisection** (`divide_conquer_median`): Partitions along the coordinate median, choosing between x- and y-coordinates to produce the most balanced split.

Both variants recursively solve subproblems until reaching the base case of 4 or fewer points, at which point brute force enumeration is applied. The solutions to subproblems are then merged by finding the optimal connection points between the two Hamiltonian cycles.

4.2.1 Results

Table 1 presents the average Wiener index and execution time for each algorithm across all configurations. For small instances ($n = 4$), all three algorithms consistently find optimal solutions with comparable execution times. However, as n increases, the divide-and-conquer heuristics become significantly faster—for $n \geq 5$, they execute in less than 1 millisecond on average, compared to over half a second for brute force. This represents a speedup of approximately 500–800× for instances with 5–7 points.

Table 1: Average Wiener index and execution time by point count and configuration type. Each entry represents the mean over 3 independent trials. The divide-and-conquer algorithms achieve substantial speedups over brute force for $n \geq 5$.

n	Type	Trials	`brute_force`		`divide_conquer`		`divide_conquer_median`	
			Wiener	Time(s)	Wiener	Time(s)	Wiener	Time(s)
4	convex	3	339.03	0.0007	339.03	0.0006	339.03	0.0007
4	general	3	226.28	0.0007	226.28	0.0006	226.28	0.0006
5	convex	3	572.88	0.5749	669.09	0.0007	667.72	0.0007
5	general	3	426.81	0.5354	460.47	0.0005	462.29	0.0004
6	convex	3	841.88	0.5449	1160.49	0.0007	1008.94	0.0006
6	general	3	642.80	0.5340	713.58	0.0007	728.35	0.0005
7	convex	3	1218.04	0.5634	1620.30	0.0010	1644.84	0.0010
7	general	3	1005.42	0.5676	1108.96	0.0009	1077.98	0.0010

Table 2 summarizes the overall performance characteristics of each algorithm across all 24 experiments. The brute force algorithm serves as the optimal baseline, achieving an average Wiener index of 659.14 across all instances. The divide-and-conquer variants produce slightly higher Wiener indices on average (787.28 and 769.43, respectively), reflecting their approximation nature. Notably, the execution time for brute force is dominated by instances with $n \geq 5$, where factorial growth makes enumeration expensive; for $n = 4$, brute force completes in 0.0007 seconds, comparable to the heuristics.

The most informative metric for evaluating approximation quality is the *approximation ratio*: the ratio of the heuristic's Wiener index to the optimal value found by brute force. Table 3 presents these ratios for each configuration. For $n = 4$, both divide-and-conquer variants achieve perfect optimality (ratio = 1.0000) on all trials. However, approximation quality degrades as n increases. For convex position instances, ratios range from 1.20 to 1.52, with the worst performance occurring at $n = 7$ for the bounding

Table 2: Overall algorithm performance statistics across all 24 experiments. Average execution time for brute force is dominated by $n \geq 5$ instances; for $n = 4$, brute force averages 0.0007s, comparable to the heuristic algorithms.

Algorithm	Total Runs	Avg Wiener	Std Wiener	Avg Time(s)
`brute_force`	24	659.14	348.43	0.4152
`divide_conquer`	24	787.28	493.95	0.0007
`divide_conquer_median`	24	769.43	462.36	0.0007

box variant. For general position instances, the degradation is more modest, with ratios between 1.05 and 1.20.

Table 3: Approximation ratios relative to optimal brute force solutions. Both divide-and-conquer algorithms achieve optimal solutions for all $n = 4$ instances but exhibit degrading performance as n increases, with worst-case ratios reaching 1.38–1.52 for convex position at $n = 7$. The "Optimal" column shows the number of trials (out of 3) where the heuristic matched the optimal solution.

n	Type	Trials	`divide_conquer`				`divide_conquer_median`			
			Avg Ratio	Min	Max	Optimal	Avg Ratio	Min	Max	Optimal
4	convex	3	1.0000	1.0000	1.0000	3/3	1.0000	1.0000	1.0000	3/3
4	general	3	1.0000	1.0000	1.0000	3/3	1.0000	1.0000	1.0000	3/3
5	convex	3	1.2080	1.0000	1.3167	1/3	1.2053	1.0000	1.3167	1/3
5	general	3	1.0797	1.0473	1.1376	0/3	1.0840	1.0473	1.1508	0/3
6	convex	3	1.3784	1.3218	1.4105	0/3	1.2441	1.0000	1.4105	1/3
6	general	3	1.1095	1.0752	1.1674	0/3	1.1323	1.0752	1.2024	0/3
7	convex	3	1.3075	1.1941	1.5151	0/3	1.3386	1.2582	1.3970	0/3
7	general	3	1.1023	1.0532	1.1989	0/3	1.0721	1.0490	1.0950	0/3

4.2.2 Discussion

Several trends emerge from our experimental results:

Small instances. For $n = 4$, both divide-and-conquer variants consistently find optimal solutions across all trials and point configurations. This is unsurprising, as the base case uses brute force enumeration, and with only 4 points, the recursive decomposition does not introduce approximation error.

Scalability vs. optimality trade-off. As n increases beyond 4, the algorithms sacrifice solution quality for computational efficiency. The speedup over brute force grows dramatically: for $n = 7$, our heuristics are approximately 560× faster. However, approximation ratios degrade to 1.31–1.52 for convex instances and 1.07–1.20 for general position instances. This trade-off may be acceptable for applications where near-optimal solutions suffice and computational resources are limited.

Point configuration sensitivity. The algorithms perform notably better on general position instances than on convex position instances. For $n = 7$, general position yields average ratios of 1.07–1.10, compared to 1.31–1.34 for convex position. We hypothesize that convex position introduces more geometric structure, making it easier for the optimal solution to exploit global properties that our local partitioning strategy fails to capture. In general position, the randomness may reduce the gap between locally-optimal and globally-optimal choices.

Partitioning strategy comparison. Surprisingly, median bisection does not consistently outperform bounding box bisection. While median bisection achieves slightly better average ratios for general position at $n = 7$ (1.0721 vs. 1.1023), it performs worse for convex position at the same n (1.3386 vs.

1.3075). This suggests that partition balance is less critical than the merge strategy for connecting subproblem solutions. The optimal connection points between two Hamiltonian cycles may depend more on geometric proximity than on partition size. However, this could also just be due to a low number of tests. In expectation, this is not much evidence. Theoretically, the median bisection makes better sense.

Limitations. Our evaluation uses only 3 trials per configuration due to the computational expense of brute force for $n \geq 5$. While our results demonstrate consistent trends, larger-scale experiments with more trials would be needed to establish statistical significance. Additionally, we are limited to small instances ($n \leq 7$) by the factorial complexity of brute force. Future work should evaluate the heuristics on larger instances where optimal solutions are unknown, using alternative quality metrics such as comparison to lower bounds or to other heuristics.

Open theoretical question. Despite the practical performance demonstrated here, we remain unable to prove any theoretical approximation bound for our algorithm. The fundamental difficulty is that the Wiener index is a *global* metric: every edge in the Hamiltonian cycle contributes to distances between all pairs of points. Local partitioning decisions can have far-reaching effects on the total Wiener index, making it extremely hard to bound the error introduced by subproblem decomposition. This contrasts with problems like Euclidean TSP, where local optimality (e.g., non-crossing tours) often implies global near-optimality due to the triangle inequality.

5 Hardness reduction

In *Geometric Spanning Trees Minimizing the Wiener Index* (A. Karim Abu-Affash) [1], one of the listed open questions is:

> Another challenging open question is to determine the complexity of computing a spanning tree of minimum Wiener index for a set P of n points in general position: Is this problem NP-hard? In this paper, we considered a variant of this problem when the total weight of the tree is bounded, and we proved that this variant is NP-hard.

We are interested in the unconstrained Euclidean version (no bound on total weight). The conjecture is that it is NP-hard.

Remark 2. Readers of this paper might be wondering what the difference between the MRCST and the unconstrained version of computing the spanning tree of minimum Wiener index for a set P of n points is. The difference is that the unconstrained version needs to respect the triangle inequality, while the general MRCST could have any arbitrary edge weights that may or may not respect the triangle inequality [1].

5.1 Identity for Wiener Index of a Tree

Let $T = (P, E)$ be a tree on $n = |P|$ vertices with edge lengths $\ell : E \to \mathbb{R}_{>0}$. For an edge $e \in E$, let s_e be the size of one of the two components obtained by removing e (the product $s_e(n - s_e)$ is symmetric, so the choice does not matter).

Theorem 5.1 (Edge Decomposition Formula)**.**

$$\mathrm{WI}(T) \;=\; \sum_{\{u,v\}\subseteq P} \mathrm{dist}_T(u,v) \;=\; \sum_{e\in E} \ell(e)\, s_e\,(n - s_e).$$

Proof. For each edge $e \in E$ and unordered pair $\{u, v\} \subseteq P$, define

$$I_e(u,v) = \begin{cases} 1, & \text{if } e \text{ lies on the unique } u\text{–}v \text{ path in } T, \\ 0, & \text{otherwise.} \end{cases}$$

Then

$$\mathrm{dist}_T(u,v) = \sum_{e\in E} \ell(e)\, I_e(u,v).$$

Summing over all unordered pairs and exchanging summations,

$$\sum_{\{u,v\}\subseteq P} \mathrm{dist}_T(u,v) = \sum_{e\in E} \ell(e) \sum_{\{u,v\}\subseteq P} I_e(u,v).$$

When e is removed, the tree splits into two components of sizes s_e and $n - s_e$. A pair $\{u, v\}$ uses e iff u and v lie in different components. There are exactly $s_e(n - s_e)$ such pairs. Thus

$$\sum_{\{u,v\}\subseteq P} \mathrm{dist}_T(u,v) = \sum_{e\in E} \ell(e)\, s_e\, (n - s_e).$$

□

Corollary 5.2 (Weighted version)**.** *If each vertex $v \in P$ has weight $w(v) \geq 0$, then the* weighted Wiener index

$$\mathrm{WWI}(T) = \sum_{\{u,v\}\subseteq P} w(u)w(v)\,\mathrm{dist}_T(u,v)$$

satisfies

$$\mathrm{WWI}(T) = \sum_{e\in E} \ell(e)\, W_e\, (W_{\mathrm{tot}} - W_e),$$

where W_e is the sum of vertex weights in one component after removing e and $W_{\mathrm{tot}} = \sum_{v\in P} w(v)$.

5.2 Why hardness for the unconstrained version is elusive

The bounded-weight version of the Wiener index minimization problem fits naturally into the framework of "bicriteria" optimization: one criterion is the total length of the spanning tree, the other is the Wiener index. The NP-hardness result of Abu-Affash exploits this tension by encoding combinatorial structure into the edge-length budget. In contrast, in the *unconstrained* Euclidean setting, the weight budget disappears, and we can no longer directly force the reduction gadgets to obey global length restrictions. This deprives us of a straightforward lever for embedding NP-hard problems.

At the same time, the Edge Decomposition Formula highlights why the unconstrained case is subtle. The Wiener index is not simply a function of the tree's total weight, but of how that weight is "distributed" across the edge cuts. For a tree T, each edge e contributes proportionally to $s_e(n - s_e)$, which favors balanced partitions. This global averaging effect makes it harder to encode rigid local constraints, a common ingredient in geometric hardness reductions.

Another difficulty is that, unlike in many geometric NP-hardness proofs (e.g., Euclidean Steiner Tree or Euclidean Traveling Salesman), the candidate optimal structures for Wiener index are not well understood. While it is known that paths maximize the Wiener index and stars minimize it in the abstract graph setting, the Euclidean embedding introduces competing geometric constraints: stars may be geometrically long, and short trees may have poor balancing in the $s_e(n - s_e)$ factors. Thus, unlike Steiner Tree reductions that can be built from "wires" and "terminals," here the optimizer may prefer very different combinatorial shapes depending on the geometry.

Finally, it is conceivable that the unconstrained Euclidean problem admits a polynomial-time approximation scheme (PTAS), as is the case for several Euclidean optimization problems (TSP, Steiner tree, etc.). If that were true, the exact hardness would have to be proved with reductions that survive approximation, which is generally more delicate.

In short, the core obstacle is that the bounded-weight NP-hardness reductions rely on the tension between weight and distance, whereas in the unconstrained setting the optimization criterion collapses to a more global, averaged structure that resists local gadget constructions. This explains why the open question has persisted: the techniques that work in the constrained case do not obviously extend to the unconstrained Euclidean problem.

References

[1] A. Karim Abu-Affash, Paz Carmi, Ori Luwisch, and Joseph S. B. Mitchell. Geometric spanning trees minimizing the wiener index, 2023.

[2] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[3] Jon L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[4] Harry Wiener. Structural determination of paraffin boiling points. *Journal of the American Chemical Society*, 69(1):17–20, 1947.