Approximate and Exact Geometric Generalized Minimum Spanning Trees

Majid Mirzanezhad
Electrical Engineering and Computer Science, Ohio University, USA

Arash Rafiey
Indiana State University, IN, USA

Abstract

We revisit the geometric variant of the generalized minimum spanning tree (GGMST) problem: we are given a set of n points in $\mathbb{R}^d$ inside an integer grid with N non-empty clusters where each cluster is a unit hypercube, the aim is to find a minimum spanning tree that contains exactly one point from each cluster while respecting the adjacencies of the clusters. For any $\epsilon > 0$, we give an $(1+\epsilon)$ approximation algorithm whose running time is $O^*\big(N(\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})} + N^{d+1} + n\log n\big)$, where O^* hides factors polynomial in $1/\epsilon$. We also give an exact FPT algorithm that runs in $O\big(N^2\rho^2\big(2^{O(\kappa^4)}\big)\big)$, where κ is an implicit parameter that relates to the input integer grid containing the non-empty clusters. Here, ρ is the maximum number of points across all clusters. When a GGMST instance is given in the plane ($\mathbb{R}^2$), we present $(1+\epsilon)$-approximation with running time $O\big(N\rho^2 280^{(\frac{1}{\epsilon})}\big)$. In the case that for any non-empty cluster, either the row or column it belongs to, consists of at most $\gamma < N$ many non-empty clusters, we give a faster FPT algorithm that runs in $O\big(N\rho^2 560^{\gamma}\big)$ time.

2012 ACM Subject Classification CCS

Keywords and phrases Computational Geometry, Minimum Spanning Tree, Approximation Algorithm

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Funding *Arash Rafiey* : Supported by Bailey College Faculty Fellowship

1 Introduction

In *Generalized Minimum Spanning Tree* (GMST) problem, we are given an undirected abstract graph $G = (V, E)$ with vertex set V and edge set $E \subseteq V \times V$ together with a weight function $w : E \to \mathbb{R}^+$ and a partition of vertices V into non-intersecting clusters. The goal is to find a minimum weight tree containing exactly one vertex from each cluster. This problem is a generalization of the well-known *Minimum Spanning Tree* (MST) Problem where each vertex of the graph is a cluster itself. In the *Generalized Traveling Salesman Problem* (GTSP) we are given a graph $G = (V, E)$ such that V is a set of vertices and $E \subseteq V \times V$. Moreover, a weight function $w : E \to \mathbb{R}^+$ assigns weights to the edges and the set of vertices V is partitioned into non-intersecting clusters. The goal is to find a minimum weight cycle that goes through all clusters and from each cluster, there exists exactly one point in that cycle.

In the geometric variant of the problem above, we are given N non-empty unit d-dimensional axis-aligned unit hypercubes, which we refer to as *clusters*, and a set of n points in $\mathbb{R}^d$ spread across the N clusters. Each cluster contains at least two points. The aim is to pick exactly one point from each cluster and connect the chosen N points by straight-line segments (Euclidean edges) to form a minimum spanning tree (MST). We only allow edges between points chosen from *adjacent* clusters, where adjacency means the clusters share at least a corner. This problem is also known as *Geometric Generalized Minimum Spanning Tree* (GGMST). We refer to this problem as GGMST problem.

1.1 Related work

The GGMST problem occurs in telecommunications network planning, where a network of node clusters need to be connected via a tree architecture using exactly one node per cluster [7]. More precisely, local subnetworks must be interconnected by a global network containing a gateway from each subnetwork. For this inter-networking, a point has to be chosen in each local network as a hub and the hub point must be connected via transmission links such as optical fiber [13]. Additionally, the problem has some applications in design of backbones in large communication networks, energy distribution, and agricultural irrigation [9].

The GMST problem was first introduced by Myung et al. (1995) [13]. However, the conventional MST problem is polynomially solvable [6], it was shown in [13] that the GMST problem is strongly NP-hard and there is no constant factor approximation for it unless P = NP. It is worth mentioning that some heuristic algorithms have been developed for GMST. For more information, see [7, 9].

When approximation is allowed, Pop et al. [14] lay out a $2\rho-$approximation algorithm for GGMST where the size of each cluster is bounded by ρ. They take advantage of Integer programming and LP relaxation to tackle the problem. Feremans, Grigoriev, and Sitters consider GGMST [5] and show a strong NP-hardness, even if we restrict to instances in which all non-empty clusters are connected and each cluster contains at most two points (Theorem 1 in [5]). They design a dynamic programming algorithm which solves the problem with connected non-empty clusters in time $O(l\rho^{6k}2^{34k^2}k^2)$ where the grid has size $l \times k$ and it is polynomial when k is fixed (Theorem 2 in [5]). Moreover, authors in [5] suggest a polynomial time approximation scheme (PTAS) for GGMST problem when the non-empty grids are connected. It is easy to see that their dynamic programming algorithm is highly exponential in k, as a result their PTAS using the dynamic program may not be perceived as efficient in practice. To overcome this issue, Bhattacharya et al. designed a simple polynomial-time approximation algorithm for GGMST with a guaranteed ratio of $1 + 4\sqrt{2} + \epsilon$ [4].

Later, Kachooei et al. [10] showed that the problem is still NP-hard when there are at most two points in each cluster even with equal y-coordinates, and it is unlikely to have a fully polynomial time approximation scheme (FPTAS) for it unless P = NP. It is also known that the GGMST is NP-hard even if the cluster graph forms a tree [15].

1.2 Our contribution

We obtained two primary results in higher dimensions and then we show how they can be improved when the instance is given in the plane. In particular, we propose approximation and FPT algorithms for each case as follows:

1. We give a $(1+\epsilon)$-approximation algorithm that runs in $O\big(n \log n + n/\epsilon^d + (N/\epsilon)^{d+1} + N(\frac{1}{\epsilon})^{O(1/\epsilon)}\big)$, in any constant dimension d (Thm. 8). Our approach utilizes the idea of Arora's original PTAS technique for TSP in [2]. Arora's technique requires portal *pairings* (entering and exiting a quadtree cell in matching pairs) to form a tour. Our GGMST variant tracks *connected* subsets of portals (partitions) to encode how GGMST edges unite boundary portals to form connected components, i.e., forests within each cell of the quadtree constructed over the clusters. In TSP, every point must appear in the tour, however in our case the problem specifies exactly one point must be chosen per cluster. We encode these portal "activations" at quadtree leaf cells via a dynamic program and avoid re-choosing points at higher levels of the quadtree.
2. We also give an exact FPT algorithm that runs in $O\big(\rho^2 N^2 \big(2^{O(k^4)}\big)\big)$, where κ is the treewidth of the hypergraph induced by the non-empty clusters of the integer grid in $\mathbb{R}^d$,

for any constant dimnesion d (Thm. 12). In telecom networks, especially large-scale ones, each local subnetwork (e.g., LAN, building, or regional cluster) designates exactly one node (its gateway or hub) to connect to external links. This is precisely the setting where each cluster (local subnetwork) has to select one hub. Real networks often have "hierarchical" or "near-tree" structures for reasons of reliability, scalability, or cost. For instance, metro fiber networks or networks built on geographic constraints (where expensive cross-links are limited) may indeed exhibit sparse topological connections, effectively bounding the treewidth.

3. We show that the optimal GGMST in $\mathbb{R}^2$ can be computed in time $O(N\rho^2 280^k)$ where the instance $(\mathcal{G},\mathcal{N},P)$ lies in a grid of size $k \times l$ and ρ is the maximum number of points inside each cluster (Thm. 14). As a byproduct of this result, we can obtain the following PTAS. Using a similar approximation argument that appeared in [5], one can obtain a $(1+\epsilon)$-approximation for instance $(\mathcal{G},\mathcal{N},P)$ in time $O(N\rho^2 280^{\frac{1}{\epsilon}})$ (Corollary 15) which significantly improves the $(1+\epsilon)$-approximation algorithm with running time $O\left(\frac{l}{\epsilon^2}\rho^{\frac{6}{\epsilon}}2^{(\frac{34}{\epsilon^2})}\right)$ in [5]. Note that our approximation algorithm in the plane can outperform compared to the one that works in higher dimensions if the quality of approximation is of interest since the base is a constant number compared to $1/\epsilon$.
4. In the case that for any non-empty cluster, either the row or column it belongs to, consists of at most $\gamma < N$ many non-empty clusters, we give another FPT algorithm in $\mathbb{R}^2$ that runs in $O\big(N\rho^2 560^{\gamma}\big)$ which is much faster than our FPT algorithm in $\mathbb{R}^d$ (Thm. 19).

2 Preliminaries

Given two points $x, y \in \mathbb{R}^d$, the Euclidean distance between them is denoted by $\|x-y\|$, which represents the standard ℓ_2 norm, capturing the natural geometric separation in the space. Computing minimum spanning trees requires setting a fixed point as the root of a tree. We denote a subtree rooted at a point x by T_x. This notation emphasizes that T_x consists of all nodes and edges descending from x, encapsulating the hierarchical structure induced by the root. The weight of the tree $W(T)$ represents the total edge lengths of T. When dealing with substructures, we use $W(T_x)$ to specify the weight of the subtree rooted at x. These definitions serve as the foundation for analyzing spatial structures when optimization the weights overall possible GGMSTs.

▶ **Definition 1** (Hypergraph (HG)). *A "Hypergraph" of an integer grid is a graph whose vertices are the non-empty clusters in the grid and two vertices share an edge iff the clusters are adjacent.*

Let $\mathcal{I} = (\mathcal{G},\mathcal{N},P)$ be an instance of the GGMST where $\mathcal{G}$ is an HG, $\mathcal{N}$ is the set of clusters with $|\mathcal{N}| = N$, and P is the set of points spread across the clusters in $\mathcal{N}$, with $|P| = n$. An HG $\mathcal{G}$ in $\mathcal{I}$ can attain a spanning tree whose nodes span all the clusters in $\mathcal{I}$. The said spanning tree is defined as follows:

▶ **Definition 2** (Backbone Spanning Tree (BBST)). *A "Backbone Spanning Tree* (BBST)*" w.r.t. some integer grid clusters is a spanning tree if*

1. *Each vertex corresponds to a cluster,*
2. *All vertices span the entire non-empty clusters,*
3. *And two vertices share an edge if their corresponding clusters are adjacent to each other.*

▶ **Definition 3.** *Let $\mathcal{I} = (\mathcal{G},\mathcal{N},P)$ be an instance of the GGMST problem. For a given* BBST *T, let* OptMST(T) *denote the optimal tree corresponding to the* BBST *T.*

Note that every backbone spanning tree can induce a GGMST whose adjacency of nodes follows the adjacency of nodes in the BBST. Let $\mathcal{B}$ be the set of all possible BBSTs. Then:

$$T_{opt} := \min_{T \in \mathcal{B}} \mathsf{OptMST}(T).$$

In other words, a branching tree of the BBST T describes the adjacency of the vertices of a GGMST T^* where T^* selects the points of every cluster marked by T and connects them with respect to the adjacency of the clusters in T.

3 Approximation algorithm for GGMST

As mentioned earlier, the GGMST problem is NP-hard in general, but some underlying geometric structural properties can be utilized to achieve approximation schemes. Our goal is to provide a PTAS, for every $\epsilon > 0$, albeit potentially with a high dependence on $1/\epsilon$.

3.1 Algorithmic sketch

We begin with discretizing each cluster into a finite set of candidate points independent of n. By choosing a sufficiently fine grid inside each cluster, we can guarantee that picking the nearest candidate point to the optimal solution's chosen point will not increase the weight (W) of optimal GGMST by more than $\epsilon \cdot W(T_{\text{opt}})$. Next, we construct a *WSPD-based* $(1+\epsilon)$-*spanner* to reduce the potential $O(n^2)$ edge set down to $O(n)$ edges without significantly increasing GGMST cost.

We then build a *quadtree* on the clusters, placing $O(1/\epsilon)$ *portals* on each cell boundary to constrain how GGMST edges may cross those boundaries. Applying a bottom-up *dynamic programming* (DP) merges partial GGMST solutions from child cells by matching their portal-connectivity patterns similar to Arora's technique for TSP that appeared in [2]. By carefully bounding the "overhead", i.e., inflated weight added to the solution due to the approximation via the portal placement introduced at each scale (from forcing edges to go through portals) and leveraging the spanner's limited set of edges, the algorithm achieves a total GGMST cost within $(1+\epsilon)$ of the true optimum, all in polynomial time for a fixed ϵ.

Let $\delta > 0$, we partition each cluster uniformly into grid cells each of side length δ. This yields $O(\frac{1}{\delta^d})$ candidate points per cluster. Let P be the resulting set of all cluster grid points. The naïve algorithm below is a dynamic programming algorithm that can compute the exact GGMST given a BBST T.

▶ **Lemma 4.** *For a given* BBST T*, one can compute the* $\mathsf{OptMST}(T)$ *in polynomial time.*

Proof. Select an internal vertex of T corresponding to a cluster C as the root. For any leaf cluster $l \in T$, and for any point $p \in l$, set $W(T_p) = 0$. For any internal node $I \in T$, and any point $q \in I$, let $\mathsf{Chd}(I)$ denote the set of children of I in BBST T. Then we have:

$$W(T_q) = \sum_{c \in \mathsf{Chd}(I)} \min_{p \in c} \Big\{ W(T_p) + \|p - q\| \Big\}$$

The algorithm runs in $O(N\rho^2)$ time using a forward and then backward breadth-first search on T since the number of children per node is $O(2^{d+1}) = O(1)$, where $d = O(1)$ and $\rho < n$ is the maximum number of points in a cluster. ◀

▶ **Lemma 5** (Discretization). *Let* $\epsilon > 0$*. We can choose* $\delta = \epsilon/(\sqrt{d}(N-1))$ *sufficiently small so that there exists a* $(1+\epsilon)$*-approximate GGMST solution selecting from these candidate points. Moreover, the number of grid points overall is* $O(N^d/\epsilon^d)$

Proof. Consider an optimal GGMST solution T_{opt} that selects a point q_i in each cluster C_i. By construction of the grid inside C_i, there is a grid point p_i no more than $\delta\sqrt{d}$ away from q_i (since the cluster is unit size and we have a δ-spaced lattice).

Replace each q_i by p_i to obtain a new (approximate) solution T'. The increase in the length of any edge (q_i, q_j) in T_{opt} is at most proportional to δ because shifting points can increase distances by at most $\sqrt{d}\delta$. Thus, the total additional cost incurred by this rounding is at most $\sqrt{d}\delta N$, since the MST has at most $N-1$ edges and each edge length changes by at most $\sqrt{d}\delta$.

In the case that $N \leq 2^d$ and assuming $d = O(1)$, one can enumerate all $N^{N-2} = 2^{(d2^d)} = O(1)$ many BBSTs and compute GGMST with a minimum total weight in polynomial time in an exact manner utilizing the algorithm described in Lemma 4. For all $N > 2^d$, following the packing argument demonstrated in Lemma 1 in [3], we derive that $W(T_{\text{opt}}) > 1$. Hence:

$W(T') \leq W(T_{\text{opt}}) + (N-1) \cdot \sqrt{d}\delta = W(T_{\text{opt}}) + (N-1) \cdot \frac{\epsilon}{N-1} = W(T_{\text{opt}}) + \epsilon \leq W(T_{\text{opt}}) + \epsilon \cdot W(T_{\text{opt}}) = (1+\epsilon) \cdot W(T_{\text{opt}})$. ◀

3.2 The dynamic program

We present the algorithm and analysis in four main steps: (1) quadtree construction and cluster isolation, (2) building a $(1+\epsilon)$-Spanner using WSPD, (3) portal placement and the dynamic programming scheme, and finally, (4) analysis of approximation guarantee and running Time.

The first two steps are relatively straightforward to handle as follows: our first goal is to spatially organize the clusters and points so that we can apply a dynamic program to propagate the minimum weight of a spanning forest from each "zone" to the adjacent ones. We use a *quadtree* to recursively partition the plane until each leaf cell of the quadtree contains at most one cluster (thus fully containing that cluster). By ensuring each cluster resides in precisely one leaf cell, picking a representative grid point from that cluster is a local decision at the leaf. We utilize further to ensure a constant number of points per leaf limits state explosion in the DP state.

Without further techniques, considering an MST among n points might require looking at $O(n^2)$ potential edges. To make the problem tractable, we reduce complexity by constructing a $(1+\epsilon)$-spanner. Given n points in the plane and $\epsilon > 0$, it is known that one can construct a $(1+\epsilon)$-spanner G with $O(n/\epsilon^d)$ edges in $O(n \log n + n/\epsilon^d)$ time [8]. Furthermore: $\text{MST}(G) \leq (1+\epsilon)\text{MST}(P)$, where P is the set of points and $\text{MST}(P)$ is the optimal MST cost on these n points.

3.2.1 Portal placement and the DP scheme

Even with a spanner, we face the challenge of ensuring we pick exactly one point per cluster to form a GGMST. We approach this as follows: In the Arora-style PTAS algorithm, portals are used to discretize how solutions cross cell boundaries. Place $p = O(1/\epsilon)$ portals uniformly along each side of every quadtree cell. Any MST edge crossing the boundary must "snap" to the nearest portal. By introducing portals: i) we limit the complexity of boundary interactions in the DP states and ii) we ensure that any additional cost from snapping edges to portals is at most $\epsilon \cdot W(T_{\text{opt}})$ after appropriate scaling takes effect.

▶ **Lemma 6.** *Let T_{opt} be an GGMST of cost $W(T_{opt})$ in a bounding box of side-length M. Suppose at quadtree level i, each cell has side length $\frac{M}{2^i}$. Let X_i be the number of GGMST edges that cross cell boundaries at level i. If every crossing edge is at least $\alpha \frac{M}{2^i}$ in length,*

then:

$$X_i \leq \frac{\beta\, W(T_{opt})}{\alpha\,(M/2^i)}$$

for some constant $\alpha > 0$ and $\beta > 0$.

▶ **Lemma 7** (Portal Approximation). *There is a way to choose $O(1/\epsilon)$ portals per cell boundary so that rerouting GGMST edges through these portals increases the GGMST cost by at most $\epsilon \cdot W(T_{opt})$.*

Proof. Following the geometric scaling argument from Arora's PTAS, at each level of the quadtree, the cost added by snapping edges to portals is a small fraction of the GGMST's cost *at that scale*. Summing these fractions over all levels (cells at different scales) yields at most $\epsilon \cdot W(T_{\text{opt}})$ in total overhead.

Let the bounding box of the entire cluster set have side length M. The quadtree is built by recursively splitting into four equal squares. At level 0 (the root), cells have side length M. At level i, cells have side length $\frac{M}{2^i}$. We place $p = O\big(\frac{1}{\epsilon}\big)$ portals on each cell boundary, so the portal spacing at level i is on the order of $\frac{M}{2^i \cdot p} = O\big(\epsilon\, \frac{M}{2^i}\big)$. The cost analysis unfolds as follows:

Overhead per Crossing at Level i: Consider an GGMST edge e that crosses a boundary at level i. Snapping e from its original crossing point to the nearest portal adds at most $O\big(\epsilon\, \frac{M}{2^i}\big)$ extra length, since the portal spacing is $O\big(\epsilon\, \frac{M}{2^i}\big)$.

Bounding the Number of Crossings: We now focus on all GGMST edges crossing cell boundaries at level i. Each such crossing indicates an edge of length at least $\Theta\big(\frac{M}{2^i}\big)$, since it spans across different cells at that scale. Following Lemma 6, if there were too many such "long" edges, their combined length would exceed $W(T_{\text{opt}})$, contradicting the GGMST's minimality. Hence, we conclude that the *total length* of edges crossing boundaries at level i is bounded by some constant factor of $W(T_{\text{opt}})$. Since each crossing can add up to $O\big(\epsilon\, \frac{M}{2^i}\big)$ extra length, the total *overhead* at this level is at most $(\epsilon \cdot c) \cdot W(T_{\text{opt}})$ for some universal constant $c > 0$.

Summation Over All Levels: The quadtree may have $O(\log N)$ levels (or $O(\log M)$). If we allocated a full $\epsilon\, W(T_{\text{opt}})$ overhead *per level*, we could end up with $O(\log N) \cdot \epsilon\, W(T_{\text{opt}})$. To avoid this, we use a *geometric distribution* of overhead:

$$\text{Overhead}_i \;\leq\; \frac{\epsilon}{2^i}\, W(T_{\text{opt}}) \quad \text{at each level } i.$$

By placing portals more densely at coarser levels, the detour at scale i is forced to be only $\frac{\epsilon}{2^i}\, W(T_{\text{opt}})$. Summing these over $i = 0$ to $\log N$ gives

$$\text{Total Overhead} \;=\; \sum_{i=0}^{\log N} \text{Overhead}_i \;\leq\; \epsilon\, W(T_{\text{opt}}) \sum_{i=0}^{\log N} \frac{1}{2^i} \;<\; 2\,\epsilon\, W(T_{\text{opt}}).$$

Adjusting constants or using $\epsilon' = \frac{\epsilon}{2}$ in the construction ensures the total overhead does not exceed $\epsilon \cdot W(T_{\text{opt}})$. This completes the proof. ◀

3.2.2 DP state definition and encoding connectivity

Consider a cell C in the quadtree. The DP state $D(C, S, B)$ might encode:

1. C: The cell for which a GGMST is being computed.

2. S: Which clusters fully contained in C have chosen their representative point. This controls the GGMST to be a forest and then connects to other forests at a higher-level cell. Since each leaf cell has at most one cluster, and these sets merge as we go up, S is manageable.
3. B: A description of how GGMST connections that leave C through its boundary portals are arranged. This is a pattern representing how portals are connected internally.

Portal Partitions: To represent how portals connect inside C, we consider a partition of the set of portals on C's boundary into connected components. Each connected component represents a partial GGMST structure inside C that connects some subset of these boundary portals. The number of partitions of a set of m elements is given by the mth Bell number B_m, which grows super-exponentially in k. For $m = p = O(1/\epsilon)$, we know that Bell number B_m is:

$$B_m = \Theta\big(\big(m^m/(e^m \cdot m^{3/2})\big)\big).$$

Hence $B_m \leq m^m = O(1/\epsilon)^{O(1/\epsilon)}$. This encoding is crucial. By storing a "minimal structure" that indicates which portals are connected, we can later merge these configurations at higher-level cells.

DP Recurrence: For a leaf cell C, we know exactly which points it contains (at most a constant number). We try all possible ways of picking one point for the cluster (if there is one) and then determine how this affects portal connectivity. In particular, for each choice, we compute the partial GGMST cost inside C (often trivial since it is just one cluster plus some boundary edges to portals). Then, we encode the resulting portal connectivity in a partition B.

$$D(C, S, B) = \underbrace{\text{PortalsCost}(p,\ B)}_{\text{how } p \text{ connects to boundary portals}}$$

where $\text{PortalsCost}(p, B)$ is the cost of hooking up p to the boundary portals in a way that yields the partition B. In other words, how does p connect to (some or all) of the portals in B? This ensures that the internal MST structure is consistent with B. For an internal cell C with children C_1, C_2, C_3, C_4, we combine their solutions:

$$D(C, S, B) = \min_{\substack{S_1 \cup S_2 \cup S_3 \cup S_4 = S \\ B_1, B_2, B_3, B_4}} \left(\sum_{i=1}^{4} D(C_i, S_i, B_i) \ + \ \textsc{MergeCost}(B_1, B_2, B_3, B_4, B) \right),$$

where MergeCost involves adding edges from the spanner G between portals to ensure the combined configuration at C respects partition B. The spanner's $O(n)$ edges ensure checking merges is efficient, see Figure 1.

3.2.3 Analysis: approximation and running time

The approximation factors set to $\epsilon/3$ for various steps such as cluster discretization, spanner construction, and portal partitioning combine to yield a $(1+\epsilon)$-approximation overall. We now give a detailed view of the running time analysis, including how the complexity of handling portals is controlled. The runtime analysis is as follows:

1. We discretize each of N cell into $O(N^d/\epsilon^d)$ cells, hence it takes $O(N^{d+1}/\epsilon^d)$ time.
2. We build a $(1+\epsilon/3)$-spanner G in time of $O(n \log n + n/\epsilon^d)$.

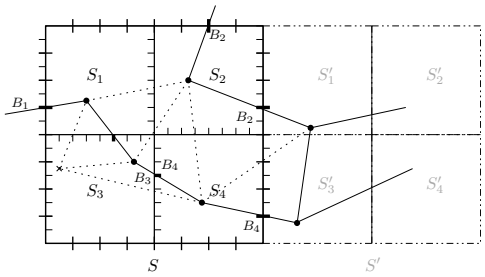

Figure 1 Connected components in the cell $S = S_1 \cup S_2 \cup S_3 \cup S_4$ have formed through a detour through the adjacent cell S' reckoned by MergeCost. The dotted lines display the edges of the underlying spanner. A partition of active (bold) portals on the boundary can be stored in B.

3. We build a quadtree on N cells in time $O(N \log N)$. The quadtree has $O(N)$ cells since we stop subdividing when each cell has at most a constant number of points and at most one cluster. Each leaf cell is thus simple to handle. Suppose each leaf has $l = O\left(\frac{N^d}{\epsilon^d}\right)$ candidate points. For each candidate grid point, we check edges to up to $p = O\left(\frac{1}{\epsilon}\right)$ portals. That's $O(l \cdot k) = O\left(\frac{N^d}{\epsilon^d} \cdot \frac{1}{\epsilon}\right)$. Summation across all leaves, since we have $O(N)$ leaves, the total is $N \times O\left(\frac{N^d}{\epsilon^d} \cdot \frac{1}{\epsilon}\right) = O\left(\frac{N^{d+1}}{\epsilon^{d+1}}\right)$.
4. At internal cells, we use a dynamic programming (DP) approach. The DP states consider:
   - Which of the four subcells are represented in the cell (chosen at lower levels).
   - How the GGMST inside the subcells connects to one another outside via a set of boundary portals. The challenge is that at each cell, we must consider different ways to connect portals. The number of ways to partition a set of m elements into connected subsets is at most the mth Bell number $B_m = (1/\epsilon)^{O(1/\epsilon)}$. Since the number of the cells in the quadtree is $O(N)$, the total would be $O(N(1/\epsilon)^{O(1/\epsilon)})$.

Overall, the runtime is $O(N^{d+1}/\epsilon^d) + O(n \log n + n/\epsilon^d) + O(N \log N) + O((N/\epsilon)^{d+1}) + O(N \cdot (1/\epsilon)^{O(1/\epsilon)})$, which is: $O\big(n \log n + n/\epsilon^d + (N/\epsilon)^{d+1} + N \cdot (1/\epsilon)^{O(1/\epsilon)}\big)$.

▶ **Theorem 8.** *Given an instance $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ with $|\mathcal{N}| = N$ and $|P| = n$, where clusters are axis-aligned unit hypersquares in $\mathbb{R}^d$, for any $\epsilon > 0$, one can compute a $(1+\epsilon)-GGMST$ of $\mathcal{I}$ in time $O^*\big((1/\epsilon)^{O(1/\epsilon)}N + N^{d+1} + n \log n\big)$, where O^* hides factors polynomial in $1/\epsilon$.*

4 An FPT algorithm for exact GGMST

Our algorithm breaks down into two levels by mixing a coarse dynamic program to enumerate all BBSTs like T and a fine-grained dynamic programming algorithm to obtain OptMST(T) (using the algorithm in Lemma 4). However, the number of backbone spanning trees can grow exponentially with the size of the graph, making enumeration computationally challenging. Given an instance $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ with $|\mathcal{N}| = N$ and $|P| = n$, we present an algorithm that efficiently enumerates all spanning trees of our $\mathcal{G} = (V, E)$ with bounded treewidth κ, exploiting the structure provided by a nice tree decomposition, where V is the set of clusters and E is the set of edges representing the adjacency between the hypernode in V. Computing the exact treewidth of a graph is NP-complete [1]. However, Korhonen [12] presented an algorithm that computes a tree decomposition of width k' satisfying $\kappa' \leq 2\kappa + 1$, where κ is the optimal treewidth of $\mathcal{G}$.

▶ **Theorem 9** (Korhonen's Algorithm [12])**.** *Given a graph G with treewidth κ, there exists an algorithm that computes a tree decomposition of $\mathcal{G}$ with width $k' \leq 2k+1$ in time $O(2^{O(k)} \cdot N)$.*

In our algorithm, we use the tree decomposition obtained from Korhonen's algorithm and κ' will be used as the treewidth parameter in our analysis.

▶ **Definition 10** (Nice Tree Decomposition). *A tree decomposition is* nice *if:*

1. *It is a rooted tree.*
2. *Each node is of one of the following types:* Leaf*: A leaf node with an empty bag.* Introduce*: A node with one child t' such that $X_t = X_{t'} \cup \{v\}$ for some $v \notin X_{t'}$.* Forget*: A node with one child t' such that $X_t = X_{t'} \setminus \{v\}$ for some $v \in X_{t'}$.* Join*: A node with two children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.*

Every tree decomposition can be transformed into a nice tree decomposition with at most $O(N)$ nodes [11].

Our goal is to enumerate all combinatorially different spanning trees of $\mathcal{G}$. We achieve this by performing dynamic programming over the nice tree decomposition of $\mathcal{G}$. The key idea is to represent partial solutions at each node of the decomposition and combine them according to specific rules that ensure correctness and avoid redundancy.

4.1 DP formulation

Our algorithm consists of several components that update the state of connectivity of vertices in each bag and the acyclicity of the spanning tree induced by the vertices in the bag.

State Representation: At each node t of the nice tree decomposition, we maintain a set of DP states. Each state represents a set of partial spanning trees consistent with the subgraph induced by the vertices seen so far.

▶ **Definition 11** (DP State). *A DP state at node t is a tuple (σ, δ) where:*

1. *σ is a partition of the vertices in X_t, representing the connected components among them in the partial spanning tree.*
2. *$\delta \subseteq E_t$ is the set of edges included in the partial spanning tree, where E_t is the set of edges between vertices in X_t.*

Note that each state must satisfy the following invariants: (1) *Acyclicity:* The edges in δ do not form cycles, and (2) *Consistency:* The partition σ accurately reflects the connectivity induced by δ. Let DP_t denote the set of DP states at node t. We define recursive formulas for computing DP_t based on the type of node. We have the following breakdowns for our DP recurrences:

- *Leaf Nodes:* $\mathrm{DP}_t = \{(\emptyset, \emptyset)\}$.
- *Introduce Nodes:* Let t be an introduce node introducing vertex v, with child t'. The DP states are computed as:

$$\mathrm{DP}_t = \bigcup_{(\sigma', \delta') \in \mathrm{DP}_{t'}} \bigcup_{\delta_v \subseteq E_v} \{(\sigma_v, \delta_v \cup \delta') \mid \mathrm{Valid}(\sigma_v, \delta_v \cup \delta')\}$$

  where: E_v is the set of edges between v and vertices in $X_{t'}$, δ_v is a subset of E_v (possible ways v can connect to $X_{t'}$), σ_v is the updated partition obtained by adding v to σ' and merging parts if v is connected to vertices in δ_v, and $\mathrm{Valid}(\sigma_v, \delta_v \cup \delta')$ ensures acyclicity and consistency.
- *Forget Nodes:* Let t be a forget node forgetting vertex v, with child t'. The DP states are computed as:

$$\mathrm{DP}_t = \{(\sigma|_{X_t}, \delta') \mid (\sigma', \delta') \in \mathrm{DP}_{t'}\}$$

  where $\sigma|_{X_t}$ is the restriction of the partition σ' to X_t.

- *Join Nodes:* Let t be a join node with children t_1 and t_2. The DP states are computed as:

$$\mathrm{DP}_t = \{(\sigma, \delta_1 \cup \delta_2) \mid (\sigma, \delta_1) \in \mathrm{DP}_{t_1}, (\sigma, \delta_2) \in \mathrm{DP}_{t_2}, \mathrm{Valid}(\sigma, \delta_1 \cup \delta_2)\}$$

The function $\mathrm{Valid}(\sigma, \delta)$ returns true if: The edges in δ do not form cycles. The partition σ correctly represents the connected components induced by δ. See Appendix 7 for further explanation and detailed pseudocode describing the recursive formulas above.

4.2 Runtime and space analysis

Let $N = |V|$ be the number of vertices in $\mathcal{G}$ and κ' be the width of the tree decomposition obtained from Korhonen's algorithm. At each node t, the size of the bag is $|X_t| \leq \kappa' + 1$. The number of possible partitions σ of X_t is given by the Bell number $B_{k'+1}$ and the number of possible edge subsets δ among vertices in X_t is $2^{\binom{\kappa'+1}{2}}$. Since for any $m > 0$, the mth Bell number is: $B_m = \left((m/e)^m m^{3/2}\right)$, the total number of states at each node, and hence the required space is:

$$S(N, \kappa) = B_{\kappa'+1} \cdot 2^{\binom{\kappa'+1}{2}} = O\Big(2^{O(\kappa^2)}\Big(O\big(\kappa/e\big)^{2\kappa+2} O(\kappa)^{3/2}\Big)\Big) = O\left(2^{O(k^2)}\right),$$

Since $\kappa' \leq 2\kappa + 1$. Regarding time complexity, we need to explore each:

- *Introduce Nodes*: For each state in $\mathrm{DP}_{t'}$, we consider $2^{|E_v|}$ subsets δ_v. Note that $|E_v| \leq \kappa'$, thus we have $2^{\kappa'}$ possibilities. The runtime per introduce node: $O(S \cdot 2^{\kappa'}) = O(S \cdot 2^{O(\kappa)})$.
- *Forget Nodes*: For each state in $\mathrm{DP}_{t'}$, we perform operations in $O(1)$ time. Hence The runtime per forget node: $O(S)$.
- *Join Nodes*: For each pair of states from the two children, we check for compatibility and merge. Number of state pairs is $O(S^2)$.

To obtain the total runtime: The number of nodes in the nice tree decomposition is $O(N)$. Hence, the total time complexity is:

$$T(N, \kappa) = O\big(N(S^2 + S \cdot 2^{O(\kappa)})\big) = O\left(N\left(2^{O(\kappa^4)}\right)\right)$$

▶ **Theorem 12.** *Given an instance $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ with $|\mathcal{N}| = N$ and $|P| = n$, where clusters are axis-aligned unit hypersquares in $\mathbb{R}^d$, one can compute the exact GGMST of $\mathcal{I}$ in time $O\left(\rho^2 N^2 \left(2^{O(k^4)}\right)\right)$, where ρ is the maximum number of points in across all clusters in $\mathcal{N}$.*

Proof. As shown above, given an instance $\mathcal{I}$, with κ as the treewidth of $\mathcal{G}$, one can enumerate all possible BBSTs in time $T(N, \kappa) = O\left(N\left(2^{O(\kappa^4)}\right)\right)$. Once all BBSTs like T are obtained, we can explicitly pick points from the selected clusters that result in the optimal GGMST, i.e., $\mathsf{OptMST}(T)$ using Lemma 4. During enumeration of all BBSTs, we fix each $T \in \mathsf{BBST}$ and utilize the algorithm described in Lemma 4 that runs in $O(N\rho^2)$. Therefore, the total runtime would be: $T(N, n, \kappa) = O\left(\rho^2 N^2 \left(2^{O(k^4)}\right)\right)$. ◀

5 Approximation and FPT algorithms for GGMST in the plane

Let $(\mathcal{G}, \mathcal{N}, P)$ be an example of the GGMST problem with N non-empty cells where the non-empty cells are connected. Let ρ be the maximum number of points inside each cluster.

▶ **Lemma 13.** *The number of backbone forests in a grid $k \times 3$ (with some non-empty cells that) is at most 280^k.*

Proof. Let $F(k)$ denote the number of such forests. Let C_1, C_2, C_3 denote the clusters in row $k-1$ and C_4, C_5, C_6 be the clusters in row k. Let H be a graph with vertices $C_1, \cdots, C_6$ and edges $(C_1, C_4), (C_1, C_5), (C_2, C_4), (C_2, C_5), (C_2, C_6), (C_3, C_5), (C_3, C_6), (C_4, C_5), (C_5, C_6)$. Here, the set of edges of H is the adjacency of each of the clusters $C_1, \cdots, C_6$ without considering the adjacency between C_1, C_2 and C_2, C_3 cells. $F(k) \leq h \times F(k-1)$ where h is the number of forests that are sub-graphs of G. It is not difficult to design a computer simulation and verify that $h = 280$. Therefore, $F(k) \leq 280^k$. ◀

▶ **Theorem 14.** *Let $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ be an instance of GGMST with $|\mathcal{N}| = N$ and $|P| = n$ and with k fixed rows. The optimal GGMST of $\mathcal{I}$ can be computed in $O(N\rho^2 280^k)$ time.*

Proof. Let $\mathcal{F}(i, j)$, where $i \leq j$, denote the set of possible forests restricted to the columns from i to j, covering all k rows of $\mathcal{G}$. By Lemma 13, there exist at most 280^k trees from $\mathcal{F}(1, 3)$ that can serve as the backbone of $\mathcal{G}$. For each such tree T, we apply the dynamic programming approach described in Lemma 4 to obtain $\mathsf{OptMST}(T)$. This requires $O(N_3\rho^2 280^k)$ time to determine the optimal forest, where N_3 represents the number of non-empty clusters up to column 3. We store the backbone forest restricted to columns c_2 and c_3 along with the weight of the minimum spanning forest, computed using the dynamic programming approach from Lemma 4. Notice that no BBST connect two clusters which are two columns (rows) apart from each other. Thus, we only need to look at two consecutive columns and enumerate possible forests as backbones.

Next, we apply a sliding window technique, moving from left to right with a window of size $k \times 2$. At step $i-1$, we assume that the minimum spanning forest whose backbone belongs to $\mathcal{F}(i-2, i-1)$ has already been computed. To expand to column i, we must consider all possible forests in $\mathcal{F}(i-1, i)$ that overlap with $\mathcal{F}(i-2, i-1)$. This requires evaluating forests in $\mathcal{F}(i-2, i-1, i)$ to identify the valid overlaps. The selection of actual points from each cluster is again determined using the dynamic programming approach from Lemma 4, which runs in $O(N_i\rho^2)$ time for each given forest, where N_i represents the number of non-empty cells across the three consecutive columns c_{i-2}, c_{i-1}, c_i. Since there are at most 280^k such forests, the runtime at step i is $O(N_i\rho^2 280^k)$. Thus, upon reaching the last column, the overall running time is $O(N\rho^2 280^k)$. Notice that at the end of the last row, we need to make sure we end up with a tree, eliminating the forests that are not tree after all. ◀

▶ **Corollary 15.** *There is $(1+\epsilon)$-approximation algorithm for GGMST of $\mathcal{I}$ in $\mathbb{R}^2$ that runs in $O(N\rho^2 280^{\frac{1}{\epsilon}})$ time.*

Proof. We apply the approximation algorithm from [5] to obtain an $(1+\epsilon)$-approximation. Their approach includes a subroutine that computes OptMST within a grid of size $k \times l$, running in time $O(l\rho^{6k} 2^{34k^2} k^2)$. However, as shown in Theorem 14, this step can be performed in $O(\rho^2 N 280^k)$ time. By carefully analyzing their PTAS, we conclude that a GMST T can be computed such that its weight is at most $(1+\epsilon) \cdot \mathsf{OptMST}$ while maintaining an overall runtime of $O(N\rho^2 280^{\frac{1}{\epsilon}})$. ◀

▶ **Definition 16** (γ-bounded)**.** *Let $(\mathcal{G}, \mathcal{N}, P)$ be an instance of the GGMST. We say $(\mathcal{G}, \mathcal{N}, P)$ is γ-bounded if for every non-empty cluster (i, j), (in row i and column j), the number of non-empty clusters in row i is at most γ or the number of non-empty clusters in column j is at most γ.*

By a simple adaptation of the argument of the Lemma 13 we have the following lemma.

▶ **Lemma 17.** *The number of backbone forests for instance $(\mathcal{G}, \mathcal{N}, P)$ which lies in a $l \times 3$ grids and for each column $1, 2, 3$ have at most γ non-empty cluster is at most 280^γ.*

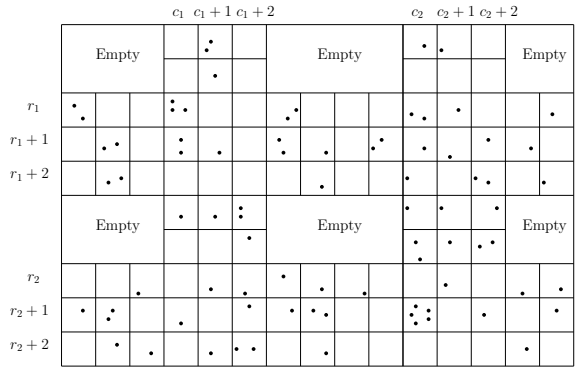

Figure 2 Example of the γ-bounded instance. Note that there could be non-empty cluster in row c_1 as well as in row r_1, etc. The empty text means the clusters without any point in them.

By a simple adaptation of the argument of the Theorem 14 we have the following lemma.

▶ **Lemma 18.** *Let $(\mathcal{G},\mathcal{N},P)$ be an instance of the GGMST problem so that the number of non-empty clusters in every column is at most γ. Then optimal GGMST can be calculated with running time $O(N\rho^2 280^\gamma)$.*

We obtain the following theorem whose proof is delivered in Appendix.

▶ **Theorem 19.** *Let $\mathcal{I}=(\mathcal{G},\mathcal{N},P)$ be an instance of GGMST in $\mathbb{R}^2$ that is γ-bounded with $|\mathcal{N}|=N$ and $|P|=n$. The GGMST of $\mathcal{I}$ can be computed in $O(N\rho^2 560^\gamma)$ time.*

6 Concluding remarks

In this paper we proposed some exact and approximation algorithms improving the runtime on the latest results on the GGMST problem. Our algorithms work in any dimension d and make use of some known techniques, the one by Arora appeared in [2]. Our main contribution is indeed the $(1+\epsilon)$-approximation for GGMST in $\mathbb{R}^d$. Arora's original PTAS for TSP relies on portal pairings—ensuring entry and exit in matching pairs—to construct a tour, whereas our GMST variant instead tracks connected subsets (partitions) of portals to capture how GMST edges merge boundary portals into forests, achieving an approximation error of $\epsilon \cdot W(T_{opt})$ via a geometric series and Bell number analysis. Unlike TSP, where every point must appear in the tour, our formulation requires exactly one point per cluster; these activations are encoded at the leaf cells of our dynamic program to prevent redundancy at higher levels. Furthermore, since edges can only connect adjacent unit squares, the complexity associated with long-range edges is reduced. While Arora's method typically places $O\big(\frac{\log n}{\epsilon}\big)$ portals per cell side—resulting in runtime factors such as $n^{\mathcal{O}(1/\epsilon)}$—our approach utilizes only $O\big(\frac{1}{\epsilon}\big)$ portals, distributing overhead hierarchically across the quadtree levels, thereby avoiding the extra $\log n$ factor in portal spacing. As with Arora's technique, the overall runtime depends super-polynomially on $\frac{1}{\epsilon}$, confirming that our method is a PTAS: it is polynomial in n for any fixed ϵ, but not fully polynomial in $\frac{1}{\epsilon}$.

Our other algorithms are also improving on some known results by [5] in $\mathbb{R}^2$ and provide fixed parameter tractablility for the case where the GMST instances behave naturally such as having bounded treewidth and/or γ-bounded as we introduced in this paper.

References

1. Steen Arnborg, Derek G. Corneil, and Andrew Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987.
2. Sanjeev Arora. Polynomial-time approximation schemes for euclidean tsp and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.
3. B. Bhattacharya, A. Ćustić, A. Rafiey, A. Rafiey, and V. Sokol. Approximation algorithms for generalized mst and tsp in grid clusters. In Z. Lu, D. Kim, W. Wu, W. Li, and D. Z. Du, editors, *Combinatorial Optimization and Applications*, volume 9486 of *Lecture Notes in Computer Science*, pages 117–129. Springer, Cham, 2015. doi:10.1007/978-3-319-26626-8_9.
4. Binay K. Bhattacharya, Ante Custic, Akbar Rafiey, Arash Rafiey, and Vladyslav Sokol. Approximation algorithms for generalized MST and TSP in grid clusters. In *Combinatorial Optimization and Applications - 9th International Conference, COCOA 2015, Houston, TX, USA*, volume 9486 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2015. doi:10.1007/978-3-319-26626-8_9.
5. C. Feremans, A. Grigoriev, and R. Sitters. The geometric generalized minimum spanning tree problem with grid clustering. *4OR*, 4:319–329, 2006.
6. M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
7. Bruce L. Golden, S. Raghavan, and Daliborka Stanojevic. Heuristic search for the generalized minimum spanning tree problem. *INFORMS Journal on Computing*, 17(3):290–304, 2005.
8. Sariel Har-peled. *Geometric Approximation Algorithms*. American Mathematical Society, USA, 2011.
9. He Jiang and Yudong Chen. An efficient algorithm for generalized minimum spanning tree problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 217–224, 2010.
10. H. A. Kachooei, M. Davoodi, and D. Tayebi. On the generalized minimum spanning tree in the euclidean plane. In *Proceedings of the 1st Iranian Conference on Computational Geometry (ICCG)*, pages 19–23, 2018.
11. Ton Kloks. *Treewidth: computations and approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994.
12. Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 2021.
13. Young-Soo Myung, Chang ho Lee, and Dong wan Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995.
14. Petrica C. Pop, Georg Still, and Walter Kern. An approximation algorithm for the generalized minimum spanning tree problem with bounded cluster size. In *Proceedings of the Algorithms and Complexity in Discrete Mathematics (ACiD)*, pages 115–121, 2005.
15. Petrica Claudiu Pop. The generalized minimum spanning tree problem. 2002.

7 Appendix: Omitted proofs and pseudocodes

▶ **Lemma 6.** *Let T_{opt} be an GGMST of cost $W(T_{opt})$ in a bounding box of side-length M. Suppose at quadtree level i, each cell has side length $\frac{M}{2^i}$. Let X_i be the number of GGMST edges that cross cell boundaries at level i. If every crossing edge is at least $\alpha \frac{M}{2^i}$ in length, then:*

$$X_i \leq \frac{\beta\, W(T_{opt})}{\alpha\,(M/2^i)}$$

for some constant $\alpha > 0$ and $\beta > 0$.

Proof. Any GGMST edge e crossing a cell boundary at level i has length at least $\alpha \frac{M}{2^i}$, because it spans distinct cells of side $\frac{M}{2^i}$, for some $\alpha > 0$. Suppose there are X_i such crossing edges. Then these X_i edges alone have total length

$$\sum_{e \text{ crosses at level } i} \|e\| \;\geq\; X_i\Big(\alpha \tfrac{M}{2^i}\Big).$$

Applying a standard geometric pigeonhole argument, the GGMST cannot afford a set of edges whose combined length exceeds $\beta \cdot W(T_{\text{opt}})$, for some constant $\beta > 0$. Concretely, if the total length of all crossing edges at level i were allowed to surpass $\beta \cdot W(T_{\text{opt}})$, one could reconstruct a substructure heavier than $W(T_{\text{opt}})$, contradicting T_{opt}'s minimality.

Consider the MST edges crossing cell boundaries at level i, each having length at least $\alpha \frac{M}{2^i}$. The *pigeons* here are precisely those boundary-crossing edges themselves, while the *holes* represent "slots" in the MST's total cost $W(T_{\text{opt}})$. In other words, if each crossing edge (pigeon) demands at least an $\alpha \frac{M}{2^i}$ share of the MST length budget (a hole), and the MST cannot allocate more than $W(T_{\text{opt}})$ total, then having "too many" such edges (pigeons) would exceed the MST's entire weight (holes). Hence:

$$\sum_{e \text{ crosses at level } i} \|e\| \;\leq\; \beta W(T_{\text{opt}}).$$

Combining both inequalities yields:

$$X_i\big(\alpha \tfrac{M}{2^i}\big) \leq \beta\, W(T_{\text{opt}}),$$

which rearranges to

$$X_i \leq \frac{\beta\, W(T_{\text{opt}})}{\alpha\left(\frac{M}{2^i}\right)} = \frac{2^i\, \beta\, W(T_{\text{opt}})}{\alpha\, M}.$$

If X_i were any larger, the total cost of those crossing edges alone would exceed $W(T_{\text{opt}})$, which is a contradiction. ◀

▶ **Theorem 19.** *Let $\mathcal{I} = (\mathcal{G}, \mathcal{N}, P)$ be an instance of GGMST in $\mathbb{R}^2$ that is γ-bounded with $|\mathcal{N}| = N$ and $|P| = n$. The GGMST of $\mathcal{I}$ can be computed in $O(N\rho^2 560^\gamma)$ time.*

Proof. Consider that $\mathcal{G}$ is embedded within a grid of dimensions $\alpha \times \beta$. We partition this grid as follows : Let $c_1, c_1+1, \ldots, c_1+\ell_1$ denote a sequence of consecutive columns where each column contains at most γ non-empty clusters. Similarly, let $c_2, c_2+1, \ldots, c_2+\ell_2$ be the next sequence of consecutive columns satisfying where each of them has at most γ non-empty clusters. We continue this partitioning until reaching the last column β (refer to Figure 2 for visualization). Likewise, define $r_1, r_1+1, \ldots, r_1+s_1$ as the first set of consecutive rows

where each row contains at most γ non-empty clusters. Define $r_2, r_2+1, \ldots, r_2+s_2$ as the next set of consecutive rows where each also contains at most γ non-empty clusters. We continue this partitioning until reaching the last row α.

Using Lemma 18, we can compute the minimum spanning forest for sub-grid restricted to columns from c_1 to $c_1+\ell_1$ and all the α rows. Next, we compute the optimal minimum forest within the rows $r_1, r_1+1, \ldots, r_1+s_1$ up to column c_1+1, denoted as $\mathsf{OptMST}(F_{r_1})$. Observe that the number of possible forests in these rows and the corresponding column c_1 is bounded by 280^γ. Since the dynamic programming approach processes the rows sequentially, we ensure that the constructed forest is compatible with the existing forest on column c_1. This requires $O(N'\rho^2 280^\gamma \times 2^\gamma)$ time, where N' represents the number of non-empty clusters in the rows from r_1 to r_1+s_1. Observe that there are at most 2^γ possible forests on column c_1. A similar procedure is applied to the subsequent set of rows, $r_2, r_2+1, \ldots, r_2+s_2$, and the next block of columns $c_2, c_2+1, \ldots, c_2+s_2$. To track all possible forests restricted to any row and up to column c_1+1, we spend $O(N_1\rho^2 280^\gamma \times 2^\gamma)$ time, where N_1 represents the number of non-empty clusters in this restricted section. Finally, as we process the columns $c_1+1, c_1+2, \ldots, c_1+\ell_1$, we maintain a sliding window of size $\alpha \times 2$. Note that for each of these forests there is another super-forest $\mathcal{S}$ that is a potential part of the OptMST. We compute the $\mathsf{OptMST}(\mathcal{S})$ using the dynamic programming approach in Lemma 4. The approach proceeds iteratively:

1. Process the grid portion between columns $c_1+\ell_1$ and c_2 row by row until reaching c_2.
2. Continue processing column by column.
3. Repeat 1 and 2 until we reach to the last row and last column in $\mathcal{G}$.

Thus, the overall runtime complexity of the algorithm is bounded by $O(N\rho^2 560^\gamma)$, as required. ◀

Algorithm Pseudocodes in Section 4

Algorithm 1 EnumerateSpanningTrees($\mathcal{G}$)

Compute an approximate treewidth k' and a tree decomposition $(T, \{X_t\})$ of $\mathcal{G}$ using Korhonen's algorithm.
Convert $(T, \{X_t\})$ into a nice tree decomposition.
for each node t in post-order traversal of T **do**
    if t is a leaf node **then**
        $\mathrm{DP}_t \leftarrow \{(\emptyset, \emptyset)\}$
    else if t is an introduce node introducing v **then**
        $\mathrm{DP}_t \leftarrow$ result of **IntroduceNode**(t, v)
    else if t is a forget node forgetting v **then**
        $\mathrm{DP}_t \leftarrow$ result of **ForgetNode**(t, v)
    else if t is a join node **then**
        $\mathrm{DP}_t \leftarrow$ result of **JoinNode**(t)
At the root node r, output the spanning trees represented by DP_r.

Algorithm 2 IntroduceNode(t, v)

Initialize $\mathrm{DP}_t \leftarrow \emptyset$
for each state (σ', δ') in $\mathrm{DP}_{t'}$ **do**
    for each subset $\delta_v \subseteq E_v$ **do**
        $\delta \leftarrow \delta' \cup \delta_v$
        $\sigma \leftarrow$ result of **UpdatePartition**(σ', v, δ_v)
        if Valid(σ, δ) **then**
            Add (σ, δ) to DP_t
return DP_t

Algorithm 3 ForgetNode(t, v)

Initialize $\mathrm{DP}_t \leftarrow \emptyset$
for each state (σ', δ') in $\mathrm{DP}_{t'}$ **do**
    $\sigma \leftarrow \sigma'$ with v removed
    Add (σ, δ') to DP_t
return DP_t

Algorithm 4 JoinNode(t)

Initialize $\mathrm{DP}_t \leftarrow \emptyset$
for each state (σ_1, δ_1) in DP_{t_1} **do**
    for each state (σ_2, δ_2) in DP_{t_2} **do**
        if $\sigma_1 = \sigma_2$ **then**
            $\delta \leftarrow \delta_1 \cup \delta_2$
            if Valid(σ_1, δ) **then**
                Add (σ_1, δ) to DP_t
return DP_t

Algorithm 5 UpdatePartition(σ', v, δ_v)

Add v as a new part in σ'
for each edge $(v, u) \in \delta_v$ **do**
    Merge the parts containing v and u in σ'
return σ'

Algorithm 6 Valid(σ, δ)

Build a graph $G_\delta = (X_t, \delta)$
if G_δ contains a cycle **then return False**
if The connected components of G_δ do not match σ **then return False**
return True