



|              |                                                                                              |
|--------------|----------------------------------------------------------------------------------------------|
| Title        | Fast Approximation Algorithm for Euclidean Minimum Spanning Tree Building in High Dimensions |
| Author(s)    | Kido, Keito; Amagata, Daichi; Hara, Takahiro                                                 |
| Citation     | Lecture Notes in Computer Science. 2025, 15875, p. 432-443                                   |
| Version Type | AM                                                                                           |
| URL          | <a href="https://hdl.handle.net/11094/105169">https://hdl.handle.net/11094/105169</a>        |
| rights       |                                                                                              |
| Note         |                                                                                              |

*The University of Osaka Institutional Knowledge Archive : OUKA*

<https://ir.library.osaka-u.ac.jp/>

The University of Osaka

# Fast Approximation Algorithm for Euclidean Minimum Spanning Tree Building in High Dimensions

Keito Kido<sup>1</sup>, Daichi Amagata<sup>1</sup><sup>[0000-0001-8571-4931]</sup>(✉), and Takahiro Hara<sup>1</sup>

<sup>1</sup> The University of Osaka, Osaka, Japan

<sup>2</sup> {kido.keito, amagata, daichi, hara}@ist.osaka-u.ac.jp

**Abstract.** This paper addresses the problem of building the Euclidean minimum spanning tree (EMST) of a high-dimensional dataset. The EMST problem has a number of applications, including visualization, cosmology, and chemical sciences, to name a few. Recently, due to machine learning-based embedding techniques, many objects are represented as high-dimensional vectors. The above applications, hence, require an efficient EMST algorithm in high dimensions. Borůvka’s algorithm is a representative one, but it incurs  $O(n^2 \log n)$  time for  $n$  points. The state-of-the-art EMST algorithms improve practical time costs, but they assume low-dimensional points. In high dimensions, such techniques are generally reduced to exhaustive searches, suggesting that computing the exact EMST incurs a high computational cost. Fortunately, many applications require fast response time while allowing approximate results. We therefore propose a new approximate EMST algorithm for high-dimensional points. We conduct experiments on real datasets, and the experimental results demonstrate that our algorithm yields an almost correct answer in a much faster time than that of state-of-the-art algorithms.

**Keywords:** EMST · High-dimensional vectors · Algorithm.

## 1 Introduction

This paper addresses the EMST (Euclidean Minimum Spanning Tree) problem, a classic but still important problem with about 100 years of research history [8]. Given a set  $P$  of  $n$  points in  $\mathbb{R}^d$ , the EMST of  $P$  is the spanning tree with the minimum weight in the complete graph of  $P$ , where the weight of a spanning tree is the sum of the distances between connected points. The EMST problem has a number of applications, such as clustering [29], chemical science [26], cosmology [21], visualization [3], among others. Recently, many objects are represented as high-dimensional vectors, due to the proliferation of machine learning-based embedding techniques. The above applications also have this observation, so efficient computing of EMST in high dimensions is required.

**Challenge.** Efficiently computing EMST has been extensively studied, and both theoretical algorithms (e.g., [7]) and practical ones (e.g., [31]) were devised. Most

of the state-of-the-art practical algorithms [19, 27, 31] assume low-dimensional spaces. Some theoretical algorithms also have this assumption [7, 11], whereas others only estimate the weight of the EMST [10] or assume a distributed computing environment, which is different from our assumption.

The state-of-the-art practical algorithms, which assume low dimensions, employ a tree-based index (e.g., *kd-tree*). Some of them improve Borůvka’s algorithm [8], which incurs  $O(n^2 \log n)$  time, with the tree index, whereas the others employ well-separated pair decomposition (WSPD) [9], and the tree index is used to run WSPD efficiently. In high dimensions, because of the curse of dimensionality, tree indices do not function well, and it is hard to find well-separated pairs. Therefore, their approaches are reduced to simple iterations of the closest-pair finding with  $O(n^2 \log n)$  time exhaustive searches.

As with this paper, the state-of-the-art approximation algorithm [23] assumes high dimensions. This algorithm, however, considers only dimensionality reduction of  $P$  and simply uses Borůvka’s algorithm on the dataset with reduced dimensionality. Hence, it still incurs  $O(n^2 \log n)$  time.

**Contribution.** Motivated by the importance of EMST in high dimensions and the efficiency issue in existing works, we propose a new approximate EMST algorithm. The high computational cost of the state-of-the-art algorithms is derived from their dependence on closest-pair computations (in Borůvka’s algorithm). We therefore design our algorithm not to have this limitation, and our algorithm does not rely on Borůvka’s algorithm.

Conceptually, our algorithm builds a *sparse* graph  $G = (P, E)$ , where  $P$  is a set of vertices (i.e., vertices are points) and  $E$  is a set of edges, such that  $e \in E$  connects only close points. To obtain such  $G$ , our algorithm employs an approximate  $k$  nearest neighbor graph and approximate  $k$  nearest neighbor searches. After  $G$  is obtained, we can obtain the minimum spanning tree of  $G$  with Kruskal’s algorithm [16]. This approach does not run any closest pair findings but still makes the necessary edges. Our algorithm therefore yields a spanning tree of  $P$  similar to its exact EMST.

We summarize the main contributions of this paper as follows:

- We propose a new approximation algorithm for the EMST problem. This algorithm returns an approximate EMST in  $\tilde{O}(n)$  time in practice. ( $\tilde{O}(\cdot)$  hides any polylog factors.)
- We conduct extensive experiments on real-world datasets. The experimental results demonstrate that our algorithm returns an almost correct answer in a much faster time than state-of-the-art algorithms. We also confirm how similar the result of our algorithm is to the exact EMST by visualization.

## 2 Preliminary

### 2.1 Problem Definition

Let  $P$  be a set of high-dimensional points  $\in \mathbb{R}^d$ , where  $d = O(1)$ . We use  $\text{dist}(p, p')$  to denote the Euclidean distance between points  $p$  and  $p' \in P$ . The Euclidean minimum spanning tree of  $P$  is defined below.

**Definition 1 (EUCLIDEAN MINIMUM SPANNING TREE).** *Given a set  $P$  of  $d$ -dimensional points, we can have an undirected graph  $G = (P, E)$  where  $E$  is a set of edges between two points in  $P$ . The weight of an edge  $(p, p')$  is  $\text{dist}(p, p')$ . Let  $W(G)$  be the weight of  $G$  and*

$$W(G) = \sum_{(p, p') \in E} \text{dist}(p, p'). \quad (1)$$

*Consider the complete graph  $G_{\text{comp}}$  of  $P$ , i.e., there is an edge between every pair of points in  $P$ . The Euclidean minimum spanning tree (EMST)  $T^*$  of  $P$  is the spanning sub-graph of  $G_{\text{comp}}$  with the minimum weight.*

Because computing the exact EMST involves many closest-pair computations, it incurs a significant computational cost in high dimensions, which can be seen in Section 2.2. We therefore consider an approximation algorithm for the EMST problem. That is, we return a spanning tree  $T$  of  $P$  which may have  $T \neq T^*$ . It is clear that  $W(T) \geq W(T^*)$ . When  $W(T) \approx W(T^*)$ ,  $T$  is highly accurate. The objective of this paper is to devise an approximate EMST algorithm that yields such a  $T$  quickly.

## 2.2 Borůvka's Algorithm

The most straightforward approach to computing the EMST of  $P$  is first to compute  $G_{\text{comp}}$  and then run Kruskal's algorithm [16]. This approach needs  $O(n^2)$  space, so it is not feasible when  $n$  is large.

Borůvka's algorithm does not have this high space complexity, and this is a building block of the state-of-the-art exact and approximate EMST algorithms [19, 23]. In this algorithm, given  $P$ , we first consider each  $p \in P$  as a graph of a single vertex. For each  $p \in P$ , Borůvka's algorithm computes its nearest neighbor point and makes an edge between  $p$  and its nearest neighbor. In the next iteration, for each disjoint graph, Borůvka's algorithm computes the nearest graph (i.e., the closest pair of points belonging to different graphs) and makes an edge between them. This is repeated until there is no disjoint graph.

Although this algorithm requires only  $O(n)$  space, it incurs  $O(n^2 \log n)$  time, which is exactly the same cost as that of the most straightforward approach. The state-of-the-art algorithms [19, 23] hence suffer from this high cost.

## 2.3 Relevant Techniques

Our algorithm is based on an approximate  $k$  nearest neighbor graph ( $Ak\text{NNG}$ ) and approximate nearest neighbor search (ANNS). Because our algorithm employs the state-of-the-art  $Ak\text{NNG}$  construction algorithm and ANNS one, we introduce them briefly.

**$Ak\text{NNG}$  construction.** Because computing the exact  $k\text{NNG}$  is computationally expensive and incurs  $O(n^2)$ , we use an  $Ak\text{NNG}$ . To construct this, we employ NNDESCENT [14]. For each point in  $P$ , NNDESCENT first adds  $k$  random edges

(i.e., neighbors). Next, for each point  $p \in P$ , it traverses  $p$ 's neighbors and their neighbors, and then it updates  $p$ 's  $k$  neighbors. This is repeated until the updates are converged or a given iteration number is reached.

For fixed parameters, NNDESCENT needs only  $O(n)$  space and  $O(n)$  time [4]. In addition, it returns an  $Ak$ NNG with high accuracy empirically.

**ANNS.** The ANNS problem has been extensively studied, and the current state-of-the-art approach is based on proximity graph [17]. Among existing proximity graph algorithms, we use HNSW [18] because it is the de-facto standard algorithm. HNSW also consumes only  $O(n)$  space and empirically returns ANNS results with high recall. Note that HNSW is not an  $Ak$ NNG-based graph. Thus, we cannot obtain an  $Ak$ NNG from the HNSW structure.

### 3 Related Work

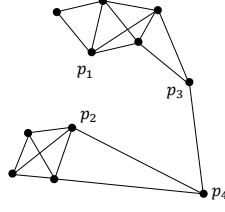
**MST on graphs.** The MST problem is one of the oldest studies in computational geometry. The input of this problem is a graph  $G = (V, E)$ , where  $V$  is a set of vertices. This problem has three famous algorithms: Prim's [25], Kruskal's [16], and Borůvka's. These algorithms all need  $O(|E| \log |V|)$  time to obtain the MST of  $G$ . The algorithm introduced in Section 2.2 is actually an extended version of the original Borůvka algorithm. (The original Borůvka algorithm iteratively selects the edge with the minimum weight for each sub-graph, thus the extended version needs to compute the nearest neighbor for each point in each iteration.) Prim's and Kruskal's algorithms can deal with EMST only when a graph of  $P$  is obtained.

**EMST.** Different from the above problem, EMST requires a set of  $d$ -dimensional points as its input. Section 2.2 introduced the state-of-the-art algorithms based on Borůvka's algorithm. Its GPU implementation was devised in [27], but this is a different setting from ours. We hence review the other EMST works.

Literature [28] shows that the EMST can be obtained in  $O(n \log n)$  time in 2-dimensional spaces. However, its proposed algorithm needs  $O(n^2 \log n)$  time when  $d > 3$ . Yao devised an  $O((n \log n)^{1.8})$  time algorithm for three-dimensional spaces [32].

In [2], a closest-pair-based algorithm is proposed, but it incurs  $O(n^2 \log^d)$  time in high-dimensional spaces. To improve the closest pair findings practically, a WSPD-based algorithm was considered in [22]. A theoretical algorithm using WSPD was analyzed in [7]. The running time of this WSPD algorithm grows exponentially w.r.t.  $d$ , suggesting that the WSPD-based approach is not appropriate in high dimensions. [26] proposed a similar technique to ours. However, it does not guarantee to return a spanning tree, and our algorithm does not have this issue.

Some theoretical studies, such as [10], do *not* compute the EMST of  $P$  but estimate the weight of the EMST. In addition, some works, e.g., [15], address the EMST problem in distributed computing environments. Different from this setting, we assume a single machine.



**Fig. 1.** Approximate  $k$  nearest neighbor graph with an edge  $(p_3, p_4)$  that prevents from obtaining the EMST (if we import the minimum spanning tree from this  $AkNNG$ ). The edge  $(p_1, p_2)$  is necessary to obtain a spanning tree with less weight.

## 4 Proposed Algorithm

**Main idea.** The straightforward algorithm builds the complete graph  $G_{comp}$  of  $P$ , which is time- and space-consuming. To alleviate this space inefficiency, Borůvka’s algorithm incrementally computes the edges of the minimum spanning tree of  $G_{comp}$ . However, Borůvka’s algorithm does not solve the time inefficiency issue. Our new idea for overcoming this challenge is to *build a sparse graph of  $P$  and then add only promising edges to this sparse graph in a batch*.

For each  $p \in P$ , the sparse graph has much fewer edges than the complete graph (e.g.,  $5 \ll n$ ), so it is space-efficient. Moreover, to obtain an approximate yet highly accurate spanning tree, this sparse graph needs to have the edges in the EMST. Because each point in the EMST is connected to its close points, we implement this idea by exploiting an approximate  $k$  nearest neighbor graph ( $AkNNG$ ).

It is important to notice that the  $AkNNG$  can have disjoint sub-graphs and detour paths (even if the  $AkNNG$  contains spanning trees). Figure 1 depicts a detour example in an  $AkNNG$ . Although we can obtain a spanning tree from this  $AkNNG$ , we cannot obtain the EMST. Any spanning trees of this  $AkNNG$  definitely have the edge  $(p_3, p_4)$ , but the EMST does not have this edge, and the upper and bottom parts of this  $AkNNG$  should be connected by  $(p_1, p_2)$ . To avoid such disjoint sub-graphs and detour paths, we add promising edges to the  $AkNNG$  by exploiting approximate nearest neighbor searches.

Our algorithm is orthogonal to any  $AkNNG$  building and ANNS algorithms (so our algorithm can be considered as a framework). Therefore, by carefully selecting efficient  $AkNNG$  building and ANNS techniques, our algorithm can quickly return a spanning tree with high accuracy. To obtain a practically and theoretically fast algorithm (by putting a reasonable assumption), we employ NNDESCENT and HNSW as  $AkNNG$  building and ANNS algorithms, respectively. Although our algorithm may be simple, its novel idea yields high efficiency (and accuracy) in practice.

**The algorithm.** Algorithm 1 details our proposed algorithm. Note that  $\mathcal{H}$ , the HNSW of  $P$ , is pre-built offline.

**Algorithm 1:** APPROXIMATE EMST BUILDING

---

**Input:**  $P, \mathcal{H}$  (HNSW of  $P$ ), and  $k$

```

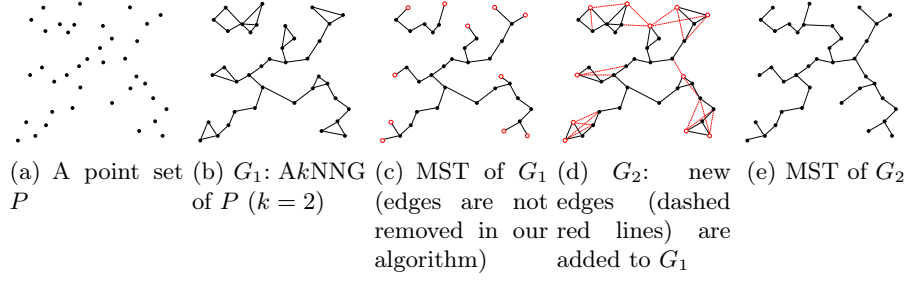
1  $G \leftarrow \text{NNDESCENT}(P, k)$ 
2  $P' \leftarrow \text{KRUSKAL}(G)$   $\triangleright$  obtain a set of points (vertices) with a single degree in
   the MST of each sub-graph
3  $k \leftarrow k \times 2$ 
4 foreach  $p_i \in P'$  do
5    $S \leftarrow \text{ANNS}(k, \mathcal{H})$ 
6   foreach  $p_j \in S$  do
7      $\lfloor$  Add edge  $(p_i, p_j)$  into  $G$  if  $(p_i, p_j)$  is not in  $E$ 
8 while 1 do
9    $f \leftarrow \text{ISDISJOINT}(G)$   $\triangleright$  checked by depth-first-search
10  if  $f$  is true then
11     $\lfloor$  Run lines 2–7
12  else
13     $\lfloor$  break
14  $T \leftarrow \text{KRUSKAL}(G)$   $\triangleright$  return the MST in  $G$ 
15 return  $T$ 

```

---

1. We first run NNDESCENT with a very small  $k$  to obtain a sparse graph  $G$  of  $P$ , i.e., an  $Ak\text{NNG}$ . Note that we use  $G$  as an undirected graph, like [6, 30].
2. Next, we run Kruskal’s algorithm on  $G$  and obtain the MST of  $G$ . Notice that this MST may contain disjoint sub-graphs, and our objective here is not to obtain the MST itself but to obtain a set  $P'$  of points (vertices of  $G$ ) with a single degree in this MST. Our idea here is that such points exist in *boundaries* between different connected components and between different dense spaces (as seen in Figure 1).
3. To remove such disjoint sub-graphs and detours, we run ANNS by using a larger  $k$  and each of these points. Specifically, for each  $p_i \in P'$ , we run  $Ak\text{NNS}$  on  $\mathcal{H}$  by using  $p_i$  as a query point and obtain the  $Ak\text{NNS}$  result  $S$ . We add the edge  $(p_i, p_j)$  to  $G$  for each  $p_j \in S$ . It is important to notice that  $|P'| \ll n$  in practice. That is, different from the existing techniques, our algorithm does not run (A)NNS for “every” point in  $P$ , thanks to the idea in step (2). This approach thus saves computational costs better than the existing state-of-the-art algorithms.
4. After that, we confirm whether  $G$  is a single connected component (i.e., whether it has some disjoint sub-graphs) by using depth-first-search. If so, we go to step (5). Otherwise, to make  $G$  a single connected component, we run steps (2) and (3) by increasing  $k$ . This step (4) is repeated until no disjoint sub-graphs appear.
5. We run Kruskal’s algorithm on  $G$  and return  $T$ , the MST of  $G$ .

*Example 1.* Figure 2 illustrates an example of our algorithm. Given a set  $P$  of points (Figure 2(a)), we build  $G_1$ , an  $Ak\text{NNG}$  of  $P$  ( $k = 2$  in Figure 2(b)). We



**Fig. 2.** Example of our algorithm

compute a set  $P'$  of points with a single degree in the MST of  $G_1$ , i.e., red circles in Figure 2(c). We set  $k = 4$  and run  $AkNNS$  for each point in  $P'$  and add edges to  $G_1$  as shown in Figure 2(d). As the resulting graph  $G_2$  does not have disjoint sub-graphs, we run Kruskal's algorithm to obtain the MST of  $G_2$ . We finally return it (Figure 2(e)).

This example clarifies the following observations: (i) Our algorithm adds edges that avoid disjoint sub-graphs and detours (e.g., compare Figures 2(c) and 2(e)) while minimizing redundant edges. (ii) This graph building is achieved by only a single  $AkNNG$  construction and some additional ANNS (i.e., the number of query points is much less than  $n$ ).

*Remark 1 (PARALLELIZABILITY).* Our algorithm is easy to parallelize: NNDESCENT is parallelizable [14], and the for loop in line 4 is also parallelizable because each ANNS is done independently. Therefore, multi-threading improves the practical efficiency of our algorithm.

**Time analysis.** For fixed parameters, NNDESCENT needs  $O(n)$  time [4, 5]. The time of Kruskal's algorithm on the  $AkNNG$  is  $O(nk \log n) = O(n \log n)$  time [16]. Let  $\mathcal{T}(n)$  be the search time of HNSW on a set of  $n$  points<sup>3</sup>. Because the number of points (vertices) with a single degree can be  $O(n)$ , lines 4–7 need  $O(n) \times \mathcal{T}(n)$  time. Line 9 requires  $O(n)$  time, so each iteration of the while loop needs at most  $O(n) \times (\mathcal{T}(n) + \log n)$  time. As we double  $k$  in each iteration, the number of iterations is  $O(\log n)$ . Given the above facts, our algorithm needs  $O(n \log n) \times \mathcal{T}(n) + O(n \log^2 n)$  time.

In practice, the original HNSW paper [18] claims that  $\mathcal{T}(n) = \text{polylog}(n)$ . Furthermore, we observe that the number of iterations of the while loop is a small constant (e.g., less than three). Our algorithm, therefore, runs in  $\tilde{O}(n)$  time in practice, which is much faster than the existing algorithms and is confirmed in our experiments.

<sup>3</sup> The theoretical upper-bound of  $\mathcal{T}(n)$  has not been given.

## 5 Experiment

We report our experimental results in this section. All experiments were conducted on a Ubuntu 22.04 LTS machine with 28 core Intel Xeon Platinum 8280L CPU@2.7GHz and 742GB RAM.

### 5.1 Setting

**Dataset.** We used FastText [20], GloVe [24], MNIST [13], and SIFT [1]. FastText (GloVe) is a set of 300-dimensional (200-dimensional) 1 (1.2) million word embedding vectors, and MNIST is a set of 784-dimensional 60,000 digit images. SIFT is a set of 128-dimensional 1 million sift image feature vectors.

**Evaluated algorithm.** We evaluated the following EMST algorithms.

- Ours<sup>4</sup>: our algorithm proposed in Section 4. We set  $k = 5$ .
- Borůvka: a state-of-the-art exact EMST algorithm.
- Borůvka + RP [23]: a state-of-the-art approximate EMST algorithm. The reduced dimensionality was 20, as with the original paper.
- DualTree [19]: a state-of-the-art exact EMST algorithm for low-dimensional points. We used MLPACK implementation [12].

All algorithms were implemented in C++ and compiled by g++ 11.4.0 with -O3 optimization.

Ours and Borůvka (+ RP) enable parallel processing, and we used OpenMP for multi-threading. By default, we used 28 threads. DualTree is a sequential algorithm and cannot be parallelized.

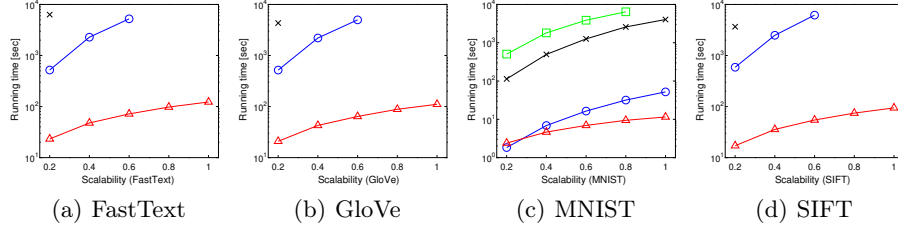
**Criteria.** We measured the time to build the exact or approximate EMST. Note that we set two hours as the time limit. For approximation algorithms, we measured the accuracy, which is  $\frac{W(T)}{W(T^*)}$ , where  $T$  is the spanning tree of  $P$  returned by an approximation algorithm.

### 5.2 Efficiency Result

We first compare our algorithm with the state-of-the-art ones while varying the data size by random sampling. Figure 3 shows the experimental results. Except for the case where a given dataset is very small (i.e., 20% sampling rate in MNIST), our algorithm outperforms the comparison algorithms. DualTree could not terminate within the time limit on the large datasets (FastText, GloVe, and SIFT) even when the sampling rate is 20%. Borůvka has a similar result, and it terminated on the large datasets only when the sampling rate is 20%. Borůvka + RP a bit improves this result, but it still could not terminate on the large datasets when the sampling rate is more than 60%.

On the other hand, our algorithm does not have the above critical issues and returns its result within a reasonable time (i.e., less than 3 minutes). We

<sup>4</sup> [https://github.com/KAKu99/approx\\_emst](https://github.com/KAKu99/approx_emst)



**Fig. 3.** Running time vs. data size. “ $\square$ ” shows **DualTree**, “ $\times$ ” shows **Borůvka**, “ $\circ$ ” shows **Borůvka + RP**, and “ $\triangle$ ” shows **Ours**.

**Table 1.** Decomposed time [sec]

|           | Ours ( $k = 5$ ) |       |       |       | Variant of ours ( $k = 20$ ) |        |       |        |
|-----------|------------------|-------|-------|-------|------------------------------|--------|-------|--------|
| Dataset   | FastText         | Glove | MNIST | SIFT  | FastText                     | Glove  | MNIST | SIFT   |
| NNDESCENT | 73.91            | 58.48 | 10.10 | 63.20 | 351.89                       | 249.57 | 35.20 | 224.85 |
| ANNS      | 23.23            | 22.69 | 0.80  | 7.28  | —                            | —      | —     | —      |
| KRUSKAL   | 9.04             | 9.83  | 0.26  | 8.18  | 8.34                         | 9.83   | 0.32  | 7.96   |

observe that our algorithm is at least 55 times faster than **Borůvka + RP** on the three large datasets. The comparison algorithms have a quadratic scale of  $n$ , but our algorithm scales linearly to  $n$ . (Plots are log-scale.) This comparison result clarifies that our new approach functions well in high dimensions and is much better than the existing techniques.

**Decomposed time.** To show the efficiency of our algorithm, we use a variant of our algorithm as a comparison. This variant runs NNDESCENT with a reasonably large  $k$  that provides an  $Ak$ NNG with no detours and disjoint sub-graphs (so it does not run ANNS). Then, this algorithm returns the MST of the  $Ak$ NNG via Kruskal’s algorithm.

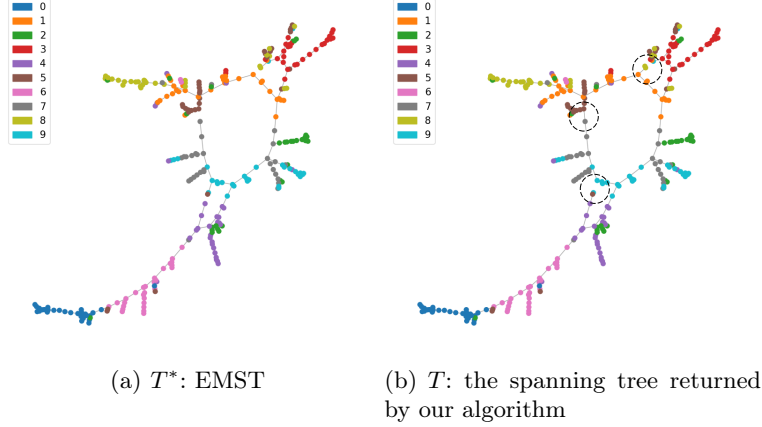
Table 1 shows the time of each component in the algorithms. It can be observed that the variant used a bit larger  $k$  than our algorithm, but its total time is at least twice than that of our algorithm. NNDESCENT with a larger  $k$  is an intuitive and easy approach to obtaining an  $Ak$ NNG whose MST is similar to the exact EMST. However, this approach makes many redundant edges, incurring in vain computations. On the other hand, our algorithm uses a small  $k$  to alleviate the cost of NNDESCENT and employs ANNS to add only necessary edges. The comparison result demonstrates that this idea minimizes the total time needed to compute an approximate EMST.

### 5.3 Accuracy Result

We show the accuracy of each approximation algorithm in Table 2. The exact algorithm did not terminate within the time limit when all points in a given

**Table 2.** Accuracy

| Dataset      | FastText | Glove | MNIST | SIFT |
|--------------|----------|-------|-------|------|
| Ours         | 1.01     | 1.02  | 1.00  | 1.01 |
| Borůvka + RP | 1.36     | 1.32  | 1.28  | 1.31 |

**Fig. 4.** Visualization of spanning trees of 300 random samples of MNIST (best viewed in color)

dataset were used (except MNIST). We therefore used 20% points to measure the accuracy.

It can be seen that our algorithm returns almost perfect results, and its accuracy is better than that of the state-of-the-art approximation algorithm Borůvka + RP. This result again justifies that our algorithm is reasonable. Even when using the exact algorithm (i.e., Borůvka), the dimensionality reduction by random projection incurs a large accuracy loss. In addition, recall that our algorithm is much faster than Borůvka + RP. That is, the performance of our algorithm dominates that of Borůvka + RP.

**Visualization.** Last, to investigate the effectiveness of our algorithm, we visually compare  $T$  (the spanning tree returned by our algorithm) with  $T^*$  (the exact EMST). We used 30 random samples of each class in MNIST (i.e., 300 samples in total) to easily confirm the accuracy of  $T$ . For visualization, we used the spring layout model.

Figure 4 depicts the visualization result. Note that each color represents the corresponding digit. Consistent with the result in Table 2,  $T$  is almost perfect, and  $\frac{W(T)}{W(T^*)} = 1.002$ . Figure 4(b) shows that  $T$  has only slight differences from  $T^*$  (specified by dashed circles). These minor differences do not lose the global

structure of a given dataset, thus it does not bother subsequent applications that want to use EMST.

## 6 Conclusion

The Euclidean minimum spanning tree problem has many important applications, and objects in these applications are usually represented as high-dimensional vectors nowadays. Existing EMST algorithms are optimized for low-dimensional points, so they are not efficient for high-dimensional points. Motivated by these observations, this paper addressed the EMST problem in high dimensions. Because it is practically hard to obtain the exact EMST of many points quickly, we considered returning an approximate EMST. We proposed a fast algorithm that returns a spanning tree of  $P$  whose weight is similar to that of its EMST. We conducted experiments on real-world datasets, and the results demonstrate that our algorithm is much faster than the state-of-the-art EMST algorithms. In addition, we confirmed its high accuracy quantitatively and qualitatively.

**Acknowledgements.** This work was partially supported by AIP Acceleration Research JPMJCR23U2, JST.

## References

1. <http://corpus-texmex.irisa.fr/>
2. Agarwal, P.K., Edelsbrunner, H., Schwarzkopf, O., Welzl, E.: Euclidean minimum spanning trees and bichromatic closest pairs. In: SoCG. pp. 203–210 (1990)
3. Alcaide, D., Aerts, J.: Spanning trees as approximation of data structures. *IEEE Transactions on Visualization and Computer Graphics* **27**(10), 3994–4008 (2020)
4. Amagata, D., Onizuka, M., Hara, T.: Fast and exact outlier detection in metric spaces: A proximity graph-based approach. In: SIGMOD. pp. 36–48 (2021)
5. Amagata, D., Onizuka, M., Hara, T.: Fast, exact, and parallel-friendly outlier detection algorithms with proximity graph in metric spaces. *The VLDB Journal* **31**(4), 797–821 (2022)
6. Arai, Y., Amagata, D., Fujita, S., Hara, T.: Lgtm: A fast and accurate knn search algorithm in high-dimensional spaces. In: DEXA. pp. 220–231 (2021)
7. Arya, S., Mount, D.M.: A fast and simple algorithm for computing approximate euclidean minimum spanning trees. In: SODA. pp. 1220–1233 (2016)
8. Borůvka, O.: O jistém problému minimálním. *Práce Moravské přírodovědecké společnosti* **3**(3), 37–58 (1926)
9. Callahan, P.B., Kosaraju, S.R.: Faster algorithms for some geometric graph problems in higher dimensions. In: SODA. vol. 93, pp. 291–300 (1993)
10. Chen, X., Cohen-Addad, V., Jayaram, R., Levi, A., Waingarten, E.: Streaming euclidean mst to a constant factor. In: STOC. pp. 156–169 (2023)
11. Chen, X., Jayaram, R., Levi, A., Waingarten, E.: New streaming algorithms for high dimensional emd and mst. In: STOC. pp. 222–233 (2022)
12. Curtin, R.R., Cline, J.R., Slagle, N.P., March, W.B., Ram, P., Mehta, N.A., Gray, A.G.: Mlpack: A scalable c++ machine learning library. *The Journal of Machine Learning Research* **14**(1), 801–805 (2013)

13. Deng, L.: The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* **29**(6), 141–142 (2012)
14. Dong, W., Moses, C., Li, K.: Efficient k-nearest neighbor graph construction for generic similarity measures. In: *World Wide Web*. pp. 577–586 (2011)
15. Jayaram, R., Mirrokni, V., Narayanan, S., Zhong, P.: Massively parallel algorithms for high-dimensional euclidean minimum spanning tree. In: *SODA*. pp. 3960–3996 (2024)
16. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society* **7**(1), 48–50 (1956)
17. Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., Lin, X.: Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* **32**(8), 1475–1488 (2019)
18. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **42**(4), 824–836 (2018)
19. March, W.B., Ram, P., Gray, A.G.: Fast euclidean minimum spanning tree: Algorithm, analysis, and applications. In: *SIGKDD*. pp. 603–612 (2010)
20. Mikolov, T., Grave, É., Bojanowski, P., Puhersch, C., Joulin, A.: Advances in pre-training distributed word representations. In: *LREC* (2018)
21. Naidoo, K., Whiteway, L., Massara, E., Gualdi, D., Lahav, O., Viel, M., Gil-Marín, H., Font-Ribera, A.: Beyond two-point statistics: using the minimum spanning tree as a tool for cosmology. *Monthly Notices of the Royal Astronomical Society* **491**(2), 1709–1726 (2020)
22. Narasimhan, G., Zhu, J., Zachariasen, M.: Experiments with computing geometric minimum spanning trees. In: *ALENEX*. pp. 183–196 (2000)
23. Narayanan, S., Silwal, S., Indyk, P., Zamir, O.: Randomized dimensionality reduction for facility location and single-linkage clustering. In: *ICML*. pp. 7948–7957 (2021)
24. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: *EMNLP*. pp. 1532–1543 (2014)
25. Prim, R.C.: Shortest connection networks and some generalizations. *The Bell System Technical Journal* **36**(6), 1389–1401 (1957)
26. Probst, D., Reymond, J.L.: Visualization of very large high-dimensional data sets as minimum spanning trees. *Journal of Cheminformatics* **12**(1), 12 (2020)
27. Prokopenko, A., Sao, P., Lebrun-Grandié, D.: A single-tree algorithm to compute the euclidean minimum spanning tree on gpus. In: *ICPP*. pp. 1–10 (2022)
28. Shamos, M.I., Hoey, D.: Closest-point problems. In: *SFCS*. pp. 151–162 (1975)
29. Sibson, R.: Slink: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal* **16**(1), 30–34 (1973)
30. Uemura, R., Amagata, D., Hara, T.: An efficient framework for approximate nearest neighbor search on high-dimensional multi-metric data. In: *SISAP*. pp. 3–17 (2024)
31. Wang, Y., Yu, S., Gu, Y., Shun, J.: Fast parallel algorithms for euclidean minimum spanning tree and hierarchical spatial clustering. In: *SIGMOD*. pp. 1982–1995 (2021)
32. Yao, A.C.C.: On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing* **11**(4), 721–736 (1982)