Information Sciences 295 (2015) 1–17

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A fast minimum spanning tree algorithm based on K-means

Caiming Zhong [a,*], Mikko Malinen [b], Duoqian Miao [c], Pasi Fränti [b]

[a] College of Science and Technology, Ningbo University, Ningbo 315211, PR China
[b] School of Computing, University of Eastern Finland, P.O. Box 111, FIN-80101 Joensuu, Finland
[c] Department of Computer Science and Technology, Tongji University, Shanghai 201804, PR China

A R T I C L E   I N F O

Article history:
Received 14 June 2014
Received in revised form 25 September 2014
Accepted 3 October 2014
Available online 14 October 2014

Keywords:
Minimum spanning tree
Clustering
Manifold learning
K-means

A B S T R A C T

Minimum spanning trees (MSTs) have long been used in data mining, pattern recognition and machine learning. However, it is difficult to apply traditional MST algorithms to a large dataset since the time complexity of the algorithms is quadratic. In this paper, we present a fast MST (FMST) algorithm on the complete graph of N points. The proposed algorithm employs a divide-and-conquer scheme to produce an approximate MST with theoretical time complexity of $O(N^{1.5})$, which is faster than the conventional MST algorithms with $O(N^2)$. It consists of two stages. In the first stage, called the divide-and-conquer stage, K-means is employed to partition a dataset into $\sqrt{N}$ clusters. Then an exact MST algorithm is applied to each cluster and the produced $\sqrt{N}$ MSTs are connected in terms of a proposed criterion to form an approximate MST. In the second stage, called the refinement stage, the clusters produced in the first stage form $\sqrt{N}-1$ neighboring pairs, and the dataset is repartitioned into $\sqrt{N}-1$ clusters with the purpose of partitioning the neighboring boundaries of a neighboring pair into a cluster. With the $\sqrt{N}-1$ clusters, another approximate MST is constructed. Finally, the two approximate MSTs are combined into a graph and a more accurate MST is generated from it. The proposed algorithm can be regarded as a framework, since any exact MST algorithm can be incorporated into the framework to reduce its running time. Experimental results show that the proposed approximate MST algorithm is computationally efficient, and the approximation is close to the exact MST so that in practical applications the performance does not suffer.

1. Introduction

A minimum spanning tree (MST) is a spanning tree of an undirected and weighted graph such that the sum of the weights is minimized. As it can roughly estimate the intrinsic structure of a dataset, MST has been broadly applied in image segmentation [2,47], cluster analysis [46,51–53], classification [27], manifold learning [48,49], density estimation [30], diversity estimation [33], and some applications of the variant problems of MST [10,36,43]. Since the pioneering algorithm of computing an MST was proposed by Otakar Borůvka in 1926 [6], the studies of the problem have focused on finding the optimal exact MST algorithm, fast and approximate MST algorithms, distributed MST algorithms and parallel MST algorithms.

The studies on constructing an exact MST start with Borůvka's algorithm [6]. This algorithm begins with each vertex of a graph being a tree. Then for each tree it iteratively selects the shortest edge connecting the tree to the rest, and combines the edge into the forest formed by all the trees, until the forest is connected. The computational complexity of this algorithm is

* Corresponding author. Tel.: +86 21 69589867.
E-mail address: zhongcaiming@nbu.edu.cn (C. Zhong).

http://dx.doi.org/10.1016/j.ins.2014.10.012
0020-0255/© 2014 Elsevier Inc. All rights reserved.

$O(E \log V)$, where E is the number of edges, and V is the number of vertices in the graph. Similar algorithms have been invented by Choquet [13], Florek et al. [19] and Sollin [42], respectively.

One of the most typical examples is Prim's algorithm, which was proposed by Jarník [26], Prim [39] and Dijkstra [15]. It first arbitrarily selects a vertex as a tree, and then repeatedly adds the shortest edge that connects a new vertex to the tree, until all the vertices are included. The time complexity of Prim's algorithm is $O(E \log V)$. If Fibonacci heap is employed to implement a min-priority queue to find the shortest edge, the computational time is reduced to $O(E + V \log V)$ [14].

Kruskal's algorithm is another widely used exact MST algorithm [32]. In this algorithm, all the edges are sorted by their weights in non-decreasing order. It starts with each vertex being a tree, and iteratively combines the trees by adding edges in the sorted order excluding those leading to a cycle, until all the trees are combined into one tree. The running time of Kruskal's algorithm is $O(E \log V)$.

Several fast MST algorithms have been proposed. For a sparse graph, Yao [50], and Cheriton and Tarjan [11] proposed algorithms with $O(E \log \log V)$ time. Fredman and Tarjan [20] proposed the Fibonacci heap as a data structure of implementing the priority queue for constructing an exact MST. With the heaps, the computational complexity is reduced to $O(E\beta(E, V))$, where $\beta(E, V) = \min\{i | \log^{(i)} V \leqslant E/V\}$. Gabow et al. [21] incorporated the idea of *Packets* [22] into the Fibonacci heap, and reduced the complexity to $O(E \log \beta(E, V))$.

Recent progress on the exact MST algorithm was made by Chazelle [9]. He discovered a new heap structure, called soft heap, to implement the priority queue, and as a result, the time complexity is reduced to $O(E\alpha(E, V))$, where α is the inverse of the Ackermann function. March et al. [35] proposed a dual-tree on a kd-tree and a dual-tree on a cover-tree for constructing MST, with claimed time complexity as $O(N \log N \alpha(N)) \approx O(N \log N)$.

Distributed MST and parallel MST algorithms have also been studied in the literature. The first algorithm of the distributed MST problem was presented by Gallager et al. [23]. The algorithm supposes that a processor exits at each vertex and knows initially only the weights of the adjacent edges. It runs in $O(V \log V)$ time. Several faster $O(V)$ time distributed MST algorithms have been proposed by Awerbuch [3] and Abdel-Wahab et al. [1], respectively. Peleg and Rubinovich [37] presented a lower bound of time complexity $O(D + \sqrt{V}/\log V)$ for constructing a distributed MST on a network, where $D = \Omega(\log V)$ is the diameter of the network. Moreover, Khan and Pandurangan [29] proposed a distributed approximate MST algorithm on networks and its complexity is $\widetilde{O}(D + L)$, where L is the local shortest path diameter.

Chong et al. [12] presented a parallel algorithm to construct an MST in $O(\log V)$ time by employing a linear number of processors. Pettie and Ramachandran [38] proposed a randomized parallel algorithm to compute a minimum spanning forest, which also runs in logarithmic time. Bader and Cong [4] presented four parallel algorithms, of which three algorithms are variants of Borůvka's. For different graphs, their algorithms can find MSTs four to six times faster using eight processors than the sequential algorithms.

Several approximate MST algorithms have been proposed. The algorithms in [7,44] are composed of two steps. In the first step, a sparse graph is extracted from the complete graph, and then in the second step, an exact MST algorithm is applied to the extracted graph. In these algorithms, different methods for extracting sparse graphs have been employed. For example, Vaidya [44] used a group of grids to partition a dataset into cubical boxes of identical size. For each box, a representative point was determined. Any two representatives of two cubical boxes were connected if the corresponding edge length was between two given thresholds. Within a cubical box, points were connected to the representative. Callahan and Kosaraju [7] applied a well-separated pair decomposition of the dataset to extract a sparse graph.

Recent studies that focused on finding an approximate MST and applying it to clustering can be found in [34,45]. Wang et al. [45] employed a divide-and-conquer scheme to construct an approximate MST. However, their goal was not to find the MST but merely to detect the long edges of the MST at an early stage for clustering. An initial spanning tree is constructed by randomly storing the dataset in a list, in which each data point is connected to its predecessor (or successor). At the same time, the weight of each edge from a data point to its predecessor (or successor) are assigned. To optimize the spanning tree, the dataset is divided into multiple subsets with a divisive hierarchical clustering algorithm (DHCA), and the nearest neighbor of a data point within a subset is found by a brute force search. Accordingly, the spanning tree is updated. The algorithm is performed repeatedly and the spanning tree is optimized further after each run.

Lai et al. [34] proposed an approximate MST algorithm based on Hilbert curve for clustering. It consists of two phases. The first phase is to construct an approximate MST with the Hilbert curve, and the second phase is to partition the dataset into subsets by measuring the densities of the points along the approximate MST with a specified density threshold. The process of constructing an approximate MST is iterative and the number of iterations is $(d + 1)$, where d is the number of dimensions of the dataset. In each iteration, an approximate MST is generated similarly as in Prim's algorithm. The main difference is that Lai's method maintains a min-priority queue by considering the approximate MST produced in the last iteration and the neighbors of the visited points determined by a Hilbert sorted linear list, while Prim's algorithm considers all the neighbors of a visited point. However, the accuracy of Lai's method depends on the order of the Hilbert curve and the number of neighbors of a visited point in the linear list.

In this paper, we propose an approximate and fast MST (FMST) algorithm based on the divide-and-conquer technique, of which the preliminary version of the idea was presented in a conference paper [54]. It consists of two stages: divide-and-conquer and refinement. In the divide-and-conquer stage, the dataset is partitioned by K-means into $\sqrt{N}$ clusters, and the exact MSTs of all the clusters are constructed and merged. In the refinement stage, boundaries of the clusters are considered. It runs in $O(N^{1.5})$ time when Prim's or Kruskal's algorithm is used in its divide-and-conquer stage, and in practical use does not reduce the quality compared to an exact MST.

The rest of this paper is organized as follows. In Section 2, the fast divide-and-conquer MST algorithm is presented. The time complexity of the proposed method is analyzed in Section 3, and experiments on the efficiency and accuracy of the proposed algorithm are given in Section 4. Finally, we conclude this work in Section 5.

2. Proposed method

2.1. Overview of the proposed method

The efficiency of constructing an MST or a K nearest neighbor graph (KNNG) is determined by the number of comparisons of the distances between two data points. In the methods like brute force for KNNG and Kruskal's for MST, many unnecessary comparisons exist. For example, to find the K nearest neighbor of a point, it is not necessary to search the entire dataset but a small local portion; to construct an MST with Kruskal's algorithm in a complete graph, it is not necessary to sort all $N(N-1)/2$ edges but to find $(1+\alpha)N$ edges with least weights, where $(N-3)/2 \gg \alpha \geqslant -1/N$. With this observation in mind, we employ a divide-and-conquer technique to build an MST with improved efficiency.

In general, a divide-and-conquer paradigm consists of three steps according to [14]:

1. Divide step. The problem is divided into a collection of subproblems that are similar to the original problem but smaller in size.
2. Conquer step. The subproblems are solved separately, and corresponding subresults are achieved.
3. Combine step. The subresults are combined to form the final result of the problem.

Following this divide-and-conquer paradigm, we constructed a two-stage fast approximate MST method as follows:

1. Divide-and-conquer stage
   1.1 Divide step. For a given dataset of N data points, K-means is applied to partition the dataset into $\sqrt{N}$ subsets.
   1.2 Conquer step. An exact MST algorithm such as Kruskal's or Prim's algorithm is employed to construct an exact MST for each subset.
   1.3 Combine step. $\sqrt{N}$ MSTs are combined using a connection criterion to form a primary approximate MST.
2. Refinement stage
   2.1 Partitions focused on borders of the clusters produced in the previous stage are constructed.
   2.2 A secondary approximate MST is constructed with the conquer and combine steps in the previous stage.
   2.3 The two approximate MSTs are merged and a new more accurate is obtained by using an exact MST algorithm.

The process is illustrated in Fig. 1. In the first stage, an approximate MST is produced. However, its accuracy is insufficient compared to the corresponding exact MST, because many of the data points that are located on the boundaries of the subsets are connected incorrectly in the MST. This is because an exact MST algorithm is applied only to data points within a subset but not to those crossing the boundaries of the subsets. To compensate for the drawback, a refinement stage is designed.

In the refinement stage, we re-partition the dataset so that the neighboring data points from different subsets will belong to the same partition. After this, the two approximate MSTs are merged, and the number of edges in the combined graph is at most $2(N-1)$. The final MST is built from this graph by an exact MST algorithm. The details of the method will be described in the following subsections.

2.2. Partition dataset with K-means

For two points connected by an edge in an MST, at least one is the nearest neighbor of the other, which implies that the connections have a locality property. Therefore, in the divide step, it is expected that the subsets preserve this locality. As K-means can partition some of local neighboring data points into the same group, we employ K-means to partition the dataset.

K-means requires the number of clusters to be known and the initial center points to be determined, and we will discuss these two problems below.

2.2.1. The number of clusters K

In this study, we set the number of clusters K to $\sqrt{N}$ based on the following two reasons. One is that the maximum number of clusters in some clustering algorithms is often set to $\sqrt{N}$ as a rule of thumb [5,41]. That means if a dataset is partitioned into $\sqrt{N}$ subsets, each subset may consist of data points coming from an identical genuine cluster so that the requirement of the locality property when constructing an MST is met.

The other reason is that the overall time complexity of the proposed approximate MST algorithm is minimized if K is set to $\sqrt{N}$, assuming that the data points are equally divided into the clusters. This choice will be theoretically and experimentally studied in more detail in Sections 3 and 4, respectively.

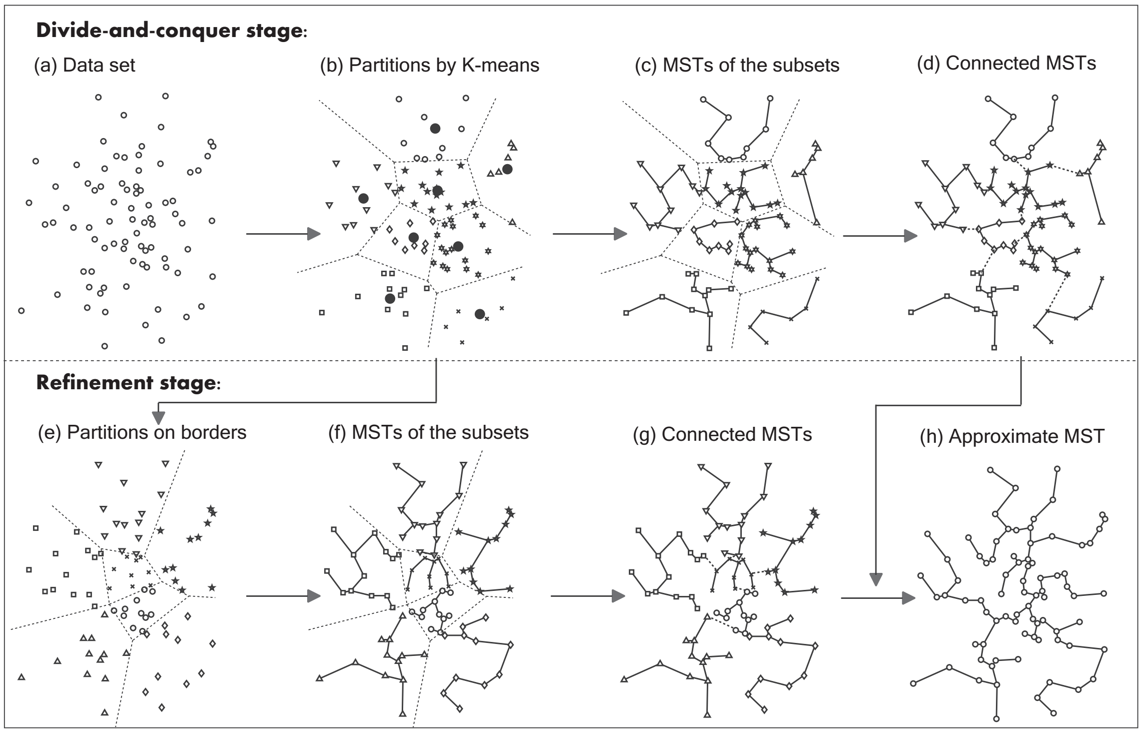

Fig. 1. The scheme of the proposed FMST algorithm. (a) A given dataset. (b) The dataset is partitioned into $\sqrt{N}$ subsets by K-means. The dashed lines form the corresponding Voronoi graph with respect to cluster centers (the big gray circles). (c) An exact MST algorithm is applied to each subset. (d) MSTs of the subsets are connected. (e) The dataset is partitioned again so that the neighboring data points in different subsets of (b) are partitioned into identical partitions. (f) An exact MST algorithm such as Prim's algorithm is used again on the secondary partition. (g) MSTs of the subsets are connected. (h) A more accurate approximate MST is produced by merging the two approximate MSTs in (d) and (g) respectively.

2.2.2. Initialization of K-means

Clustering results of K-means are sensitive to the initial cluster centers. A bad selection of the initial cluster centers may have negative effects on the time complexity and accuracy of the proposed method. However, we still randomly select the initial centers due to the following considerations.

First, although a random selection may lead to a skewed partition, such as a linear partition, the time complexity of the proposed method is still $O(N^{1.5})$, see Theorem 2 in Section 4. Second, in the proposed method, a refinement stage is designed to cope with the data points on the cluster boundaries. This process makes the accuracy relatively stable, and random selection of initial cluster centers is reasonable.

2.2.3. Divide-and-conquer algorithm

After the dataset has been divided into $\sqrt{N}$ subsets by K-means, the MSTs of the subsets are constructed with an exact MST algorithm, such as Prim's or Kruskal's. This corresponds to the conquer step in the divide and conquer scheme, it is trivial and illustrated in Fig. 1(c). The algorithm of K-means based on divide and conquer is described as follows:

```
Divide and Conquer Using K-means (DAC)
Input: Dataset $X$;
Output: MSTs of the subsets partitioned from $X$
Step 1. Set the number of subsets $K = \sqrt{N}$.
Step 2. Apply $K$-means to $X$ to achieve $K$ subsets $S = \{S_1, \ldots, S_K\}$, where the initial centers are randomly selected.
Step 3. Apply an exact MST algorithm to each subset in $S$, and an MST of $S_i$, denoted by $MST(S_i)$, is obtained,
        where $1 \leqslant i \leqslant K$.
```

The next step is to combine the MSTs of the K subsets into a whole MST.

2.3. Combine MSTs of the K subsets

An intuitive solution to combining MSTs is brute force: For the MST of a cluster, the shortest edge between it and the MSTs of other clusters is computed. But this solution is time consuming, and therefore a fast MST-based effective solution is also presented. The two solutions are discussed below.

2.3.1. Brute force solution

Suppose we combine a subset S_l with another subset, where $1 \leqslant l \leqslant K$. Let x_i, x_j be data points and $x_i \in S_l, x_j \in X - S_l$. The edge that connects S_l to another subset can be found by brute force:

$$e = \arg\min_{e_i \in E_l} \rho(e_i) \tag{1}$$

where $E_l = \{e(x_i, x_j) | x_i \in S_l \wedge x_j \in X - S_l\}, e(x_i, x_j)$ is the edge between vertices x_i and x_j, $\rho(e_i)$ is the weight of edge e_i. The whole MST is obtained by iteratively adding e into the MSTs and finding the new connecting edge between the merged subset and the remaining part. This process is similar to single-link clustering [21].

However, the computational cost of the brute force method is high. Suppose that each subset has an equal size of N/K, and K is an even number. The running time T_c of combining the K trees into the whole MST is:

$$\begin{aligned} T_c &= 2 \times \left\{ \frac{N}{K} \times \frac{(K-1) \times N}{K} + \frac{2 \times N}{K} \times \frac{(K-2) \times N}{K} + \cdots + \frac{(K/2) \times N}{K} \times \frac{(K/2) \times N}{K} \right\} = \left(\frac{K^2}{6} + \frac{K}{4} - \frac{1}{6} \right) \times \frac{N^2}{K} \\ &= O(KN^2) = O(N^{2.5}) \end{aligned} \tag{2}$$

Consequently, a more efficient combining method is needed.

2.3.2. MST-based solution

The efficiency of the combining process can be improved in two aspects. First, in each combining iteration only one pair of neighboring subsets is considered in finding the connecting edge. Intuitively, it is not necessary to take into account subsets that are far from each other, because no edge in an exact MST connects the subsets. This consideration will save some computations. Second, to determine the connecting edge of a pair of neighboring subsets, the data points in the two subsets will be scanned only once. The implementation of the two techniques is discussed in detail.

Determine the neighboring subsets. As the aforementioned brute force solution runs in the same way as single-link clustering [24] and all the information required by single-link can be provided by the corresponding MST of the same data, we make use of the MST to determine the neighboring subsets and improve the efficiency of the combination process.

If each subset has one representative, an MST of the representatives of the K subsets can roughly indicate which pairs of subsets could be connected. For simplicity, the mean point, called the center, of a subset is selected as its representative. After an MST of the centers (MST_{cen}) is constructed, each pair of subsets whose centers are connected by an edge of MST_{cen} is combined. Although not all of the neighboring subsets can be discovered by MST_{cen}, the dedicated refinement stage could remedy this drawback to some extent.

The centers of the subsets in Fig. 1(c) are illustrated as the solid points in Fig. 2(a), and MST_{cen} is composed of the dashed edges in Fig. 2(b).

Determine the connecting edges. To combine MSTs of a pair of neighboring subsets, an intuitive way is to find the shortest edge between the two subsets and connect the MSTs by this edge. Under the condition of an average partition, finding the shortest edge between two subsets takes N steps, and therefore, the time complexity of the whole connection process is $O(N^{1.5})$. Although this does not increase the total time complexity of the proposed method, the absolute running time is still somewhat high.

To make the connecting process faster, a novel way to detect the connecting edges is illustrated in Fig. 3. Here, c_2 and c_4 are the centers of the subset S_2 and S_4, respectively. Suppose a is the nearest point to c_4 from S_2, and b is the nearest point to c_2 from S_4. The edge $e(a, b)$ is selected as the connecting edge between S_2 and S_4. The computational cost of this is low. Although the edges found are not always optimal, this can be compensated by the refinement stage.

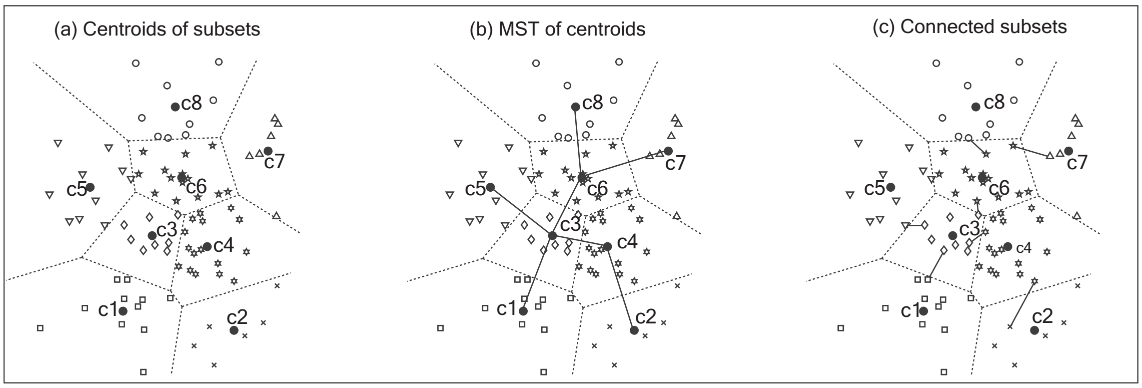

Fig. 2. The combine step of MSTs of the proposed algorithm. In (a), centers of the partitions (c1, ..., c8) are calculated. In (b), a MST of the centers, MST_{cen}, is constructed with an exact MST algorithm. In (c), each pair of subsets whose centers are neighbors with respect to MST_{cen} in (b) is connected.

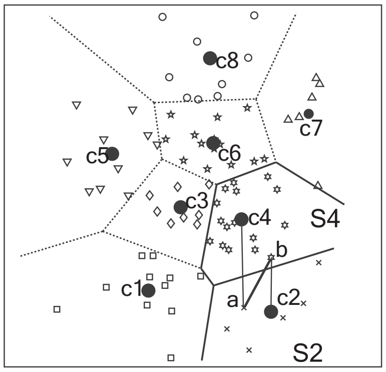

Fig. 3. Detecting the connecting edge between S_4 and S_2.

Consequently, the algorithm for combining the MSTs of the subsets is summarized as follows:

Combine Algorithm (CA)
Input: MSTs of the subsets partitioned from X : $MST(S_1), \ldots, MST(S_K)$.
Output: Approximate MST of X, denoted by MST_1, and MST of the centers of $S_1, \ldots, S_K$, denoted by MST_{cen};

```
Step 1. Compute the center $c_i$ of subset $S_i, 1 \leqslant i \leqslant K$.
Step 2. Construct an MST, $MST_{cen}$, of $c_1, \ldots, c_K$ by an exact MST algorithm.
Step 3. For each pair of subsets $(S_i, S_j)$ that their centers $c_i$ and $c_j$ are connected by an edge $e \in MST_{cen}$, discover the edge
        by DCE (Detect the Connecting Edge) that connects $MST(S_i)$ and $MST(S_j)$.
Step 4. Add all the connecting edges discovered in Step 3 to $MST(S_1), \ldots, MST(S_K)$, and $MST_1$ is achieved.
```

Detect the Connecting Edge (DCE)
Input: A pair of subsets to be connected, (S_i, S_j);
Output: The edge connecting $MST(S_i)$ and $MST(S_j)$;

```
Step 1. Find the data point $a \in S_i$ such that the distance between $a$ and the center of $S_j$ is minimized.
Step 2. Find the data point $b \in S_j$ such that the distance between $b$ and the center of $S_i$ is minimized.
Step 3. Select edge $e(a, b)$ as the connecting edge.
```

2.4. Refine the MST focusing on boundaries

However, the accuracy of the approximate MST achieved so far is far from the exact MST. The reason is that, when the MST of a subset is built, the data points that lie in the boundary of the subset are considered only within the subset, but not across the boundaries. In Fig. 4, subsets S_6 and S_3 have a common boundary, and their MSTs are constructed independently. In the MST of S_3, point a and b are connected to each other. But in the exact MST they are connected to the points in S_6 rather than in S_3. Therefore, data points located on the boundaries are prone to be misconnected. Based on this observation, the refinement stage is designed.

2.4.1. Partition dataset focusing on boundaries

In this step, another complimentary partition is constructed so that the clusters would locate at the boundary areas of the previous K-means partition. We first calculate the midpoints of each edge of MST_{cen}. These midpoints generally lie near the boundaries, and are therefore employed as the initial cluster centers. The dataset is then partitioned by K-means. The partition process of this stage is different from that of the first stage. In this stage, the initial cluster centers are specified and the maximum number of iterations is set to 1 for the purpose of focusing on the boundaries. Since MST_{cen} has $\sqrt{N} - 1$ edges, there will be $\sqrt{N} - 1$ clusters in this stage. The process is illustrated in Fig. 5.

In Fig. 5(a), the midpoints of the edges of MST_{cen} are computed as $m_1, \ldots, m_7$. In Fig. 5(b), the dataset is partitioned with respect to these seven midpoints.

2.4.2. Build secondary approximate MST

After the dataset has been re-partitioned, the conquer and combine steps are similar to those used for producing the primary approximate MST. The algorithm is summarized as follows:

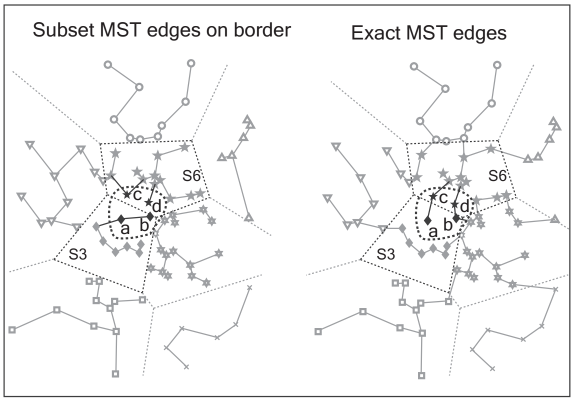

Fig. 4. The data points on the subset boundaries are prone to be misconnected.

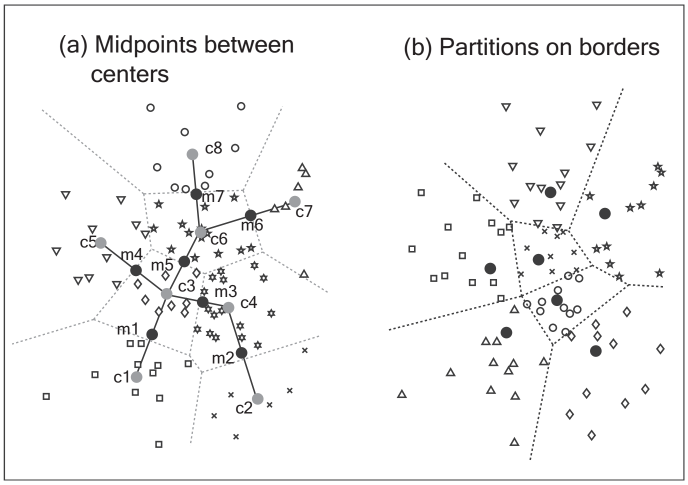

Fig. 5. Boundary-based partition. In (a), the black solid points, $m_1, \ldots, m_7$, are the midpoints of the edges of MST_{cen}. In (b), each data point is assigned to its nearest midpoint, and the dataset is partitioned by the midpoints. The corresponding Voronoi graph is with respect to the midpoints.

Secondary Approximate MST (SAM)
Input: MST of the subset centers MST_{cen}, dataset X;
Output: Approximate MST of X, MST_2;

```
Step 1.  Compute the midpoint $m_i$ of an edge $e_i \in MST_{cen}$, where $1 \leqslant i \leqslant K-1$.
Step 2.  Partition dataset $X$ into $K-1$ subsets, $S'_1, \ldots, S'_{K-1}$, by assigning each point to its nearest point from $m_1, \ldots, m_{K-1}$.
Step 3.  Build MSTs, $MST(S'_1), \ldots, MST(S'_{K-1})$, with an exact MST algorithm.
Step 4.  Combine the $K-1$ MSTs with CA to produce an approximate MST $MST_2$.
```

2.5. Combine two rounds of approximate MSTs

So far we have two approximate MSTs on dataset X, MST_1 and MST_2. To produce the final approximate MST, we first merge the two approximate MSTs to produce a graph, which has no more than $2(N-1)$ edges, and then apply an exact MST algorithm to this graph to achieve the final approximate MST of X.

Finally, the overall algorithm of the proposed method is summarized as follows:

Fast MST (FMST)
Input: Dataset X;
Output: Approximate MST of X;

(*continued on next page*)

Step 1. Apply **DAC** to X to produce the K MSTs.
Step 2. Apply **CA** to the K MSTs to produce the first approximate MST, MST_1, and the MST of the subset centers, MST_{cen}.
Step 3. Apply **SAM** to MST_{cen} and X to generate the secondary approximate MST, MST_2.
Step 4. Merge MST_1 and MST_2 into a graph G.
Step 5. Apply an exact MST algorithm to G, and the final approximate MST is achieved.

3. Complexity and accuracy analysis

3.1. Complexity analysis

The overall time complexity of the proposed algorithm **FMST**, T_{FMST}, can be evaluated as:

$$T_{FMST} = T_{DAC} + T_{CA} + T_{SAM} + T_{COM} \tag{3}$$

where T_{DAC}, T_{CA} and T_{SAM} are the time complexities of the algorithms **DAC**, **CA** and **SAM**, respectively, and T_{COM} is the running time of an exact MST algorithm on the combination of MST_1 and MST_2.

DAC consists of two operations: partitioning the dataset X with K-means and constructing the MSTs of the subsets with an exact MST algorithm. Now we consider the time complexity of DAC by the following theorems.

Theorem 1. *Suppose a dataset with N points is equally partitioned into K subsets by K-means, and an MST of each subset is produced by an exact algorithm. If the total running time for partitioning the dataset and constructing MSTs of the K subsets is T, then* $\arg\min_K T = \sqrt{N}$.

Proof. Suppose the dataset is partitioned into K clusters equally so that the number of data points in each cluster equals N/K. The time complexity of partitioning the dataset and constructing the MSTs of K subsets are $T_1 = NKId$ and $T_2 = K(N/K)^2$, respectively, where I is the number of iterations of K-means and d is the dimension of the dataset. The total complexity is $T = T_1 + T_2 = NKId + N^2/K$. To find the optimal K corresponding to the minimum T, we solve $\partial T/\partial K = NId - N^2/K^2 = 0$ which results in $K = \sqrt{N/Id}$. Therefore, $K = \sqrt{N}$ and $T = O(N^{1.5})$ under the assumption that $I \ll N$ and $d \ll N$. Because convergence of K-means is not necessary in our method, we set I to 20 in all of our experiments. For very high dimensional datasets, $d \ll N$ may not hold, but for modern large datasets it may hold. The situation for high dimensional datasets is discussed in Section 4.5. □

Although the above theorem holds under the ideal condition of average partition, it can be supported by more evidence when the condition is not satisfied, for example, linear partition and multinomial partition.

Theorem 2. *Suppose a dataset is linearly partitioned into K subsets. If* $K = \sqrt{N}$, *then the time complexity is* $O(N^{1.5})$.

Proof. Let $n_1, n_2, \ldots, n_K$ be the numbers of data points of the K clusters. The K numbers form an arithmetic series, namely, $n_i - n_{i-1} = c$, where $n_1 = 0$ and c is a constant. The arithmetic series sums up to $sum = K * n_K/2 = N$, and thus, we have $n_K = 2N/K$ and $c = 2N/[K(K-1)]$. The time complexity of constructing MSTs of the subsets is then:

$$\begin{aligned} T_2 &= n_1^2 + n_2^2 + \cdots + n_{K-1}^2 = c^2 + (2c)^2 + \cdots + [(K-1)c]^2 = c^2 \times \frac{(K-1)K(2K-1)}{6} \\ &= \left[\frac{2N}{(K-1)K}\right]^2 \times \frac{(K-1)K(2K-1)}{6} = \frac{2}{3} \times \frac{(2K-1)N^2}{K(K-1)} \end{aligned} \tag{4}$$

If $K = \sqrt{N}$, then $T_2 = \frac{4}{3}N^{1.5} + \frac{2}{3}\frac{N^{1.5}}{N^{0.5}-1} = O(N^{1.5})$. Therefore, $T = T_1 + T_2 = O(N^{1.5})$ holds. □

Theorem 3. *Suppose a dataset is partitioned into K subsets, and the sizes of the K subsets follow a multinomial distribution. If* $K = \sqrt{N}$, *then the time complexity is* $O(N^{1.5})$.

Proof. Let $n_1, n_2, \ldots, n_K$ be the numbers of data points of the K clusters. Suppose the data points are randomly assigned into the K clusters, and $n_1, n_2, \ldots, n_K \sim Multinomial(N, \frac{1}{K}, \ldots, \frac{1}{K})$. We have $Ex(n_i) = N/K$ and $Var(n_i) = (N/K) * (1 - 1/K)$. Since $Ex(n_i^2) = [Ex(n_i)]^2 + Var(n_i) = N^2/K^2 + N * (K-1)/K^2$, the expected complexity of constructing MSTs is $T_2 = \sum_{i=1}^{K} n_i^2 = K * Ex(n_i^2) = N^2/K + N * (K-1)/K$, if $K = \sqrt{N}$, then $T_2 = O(N^{1.5})$. Therefore $T = T_1 + T_2 = O(N^{1.5})$ holds. □

According to the above theorems, we have $T_{DAC} = O(N^{1.5})$.

In **CA**, the time complexity of computing the mean points of the subsets is $O(N)$, as one scan of the dataset is enough. Constructing MST of the K mean points by an exact MST algorithm takes only $O(N)$ time. In Step 3, the number of subset pairs is $K-1$, and for each pair, determining the connecting edge by **DCE** requires one scan on the two subsets, respectively. Thus, the time complexity of Step 3 is $O(2N \times (K-1)/K)$, which equals $O(N)$. The total computational cost of **CA** is therefore $O(N)$.

In **SAM**, Step 1 computes $K-1$ midpoints, which takes $O(N^{0.5})$ time. Step 2 takes $O(N \times (K-1))$ to partition the dataset. The running time of Step 3 is $O((K-1) \times N^2/(K-1)^2) = O(N^2/(K-1))$. Step 4 is to call **CA** and has the time complexity of $O(N)$. Therefore, the time complexity of **SAM** is $O(N^{1.5})$.

The number of edges in the graph that is formed by combining MST_1 and MST_2 is at most $2(N-1)$. The time complexity of applying an exact MST algorithm to this graph is only $O(2(N-1)\log N)$. Thus, $T_{COM} = O(N \log N)$.

To sum up, the time cost of the proposed algorithm is $(c_1 N^{1.5} + c_2 N \log N + c_3 N + N^{0.5}) = O(N^1.5)$. The hidden constants are not remarkable; according to our experiments we estimate them as $c1 = 3 + d * I, c2 = 2, c3 = 5$. The space complexity of the algorithm is the same as that of K-means and Prim, which are $O(N)$ if a Fibonacci heap is used within Prim's algorithm.

3.2. Accuracy analysis

Most inaccuracies originate from points that are in the boundary regions of the partitions of K-means. The secondary partition is generated in order to capture these problematic points into the same clusters. Inaccuracies after the refinement stage can, therefore, originate only if two points should be connected by the exact MST, but are partitioned into different clusters both in the primary and in the secondary partition, and neither of the two conquer stages will be able to connect these points. In Fig. 6, few such pair of points are shown that belong to different clusters in both partitions. For example, point a and b belong to different clusters of the first partition, but are in the same cluster of the second.

Since partitions generated by K-means form a Voronoi graph [16], the analysis of the inaccuracy can be related to the degree by which the secondary Voronoi edges overlap that of the Voronoi edges of the primary partition. Let $|E|$ denote the number of edges of a Voronoi graph, in two-dimensional space, $|E|$ is bounded by $K-1 \leqslant |E| \leq 3K-6$, where K is the number of clusters (the Voronoi regions). In a higher dimensional case it is more difficult to analyze.

A favorable case is demonstrated in Fig. 7. The first row is a dataset which consists of 400 points and is randomly distributed. In the second row, the dataset is partitioned into six clusters by K-means, and a collinear Voronoi graph is achieved. In the third row, the secondary partition has five clusters, each of which completely cover one boundary region in the second row. An exact MST is produced in the last row.

4. Experiments

In this section, experimental results are presented to illustrate the efficiency and the accuracy of the proposed fast approximate MST algorithm. The accuracy of FMST is tested with both synthetic datasets and real applications. As a framework, the proposed algorithm can be incorporated with any exact or even approximate MST algorithm, of which the running time is definitely reduced. Here we only take into account Kruskal's and Prim's algorithms because of their popularity. As in Kruskal's algorithm, all the edges need to be sorted into nondecreasing order, it is difficult to apply the algorithm to large datasets. Furthermore, as Prim's algorithm may employ a Fibonacci heap to reduce the running time, we therefore use it rather than Kruskal's algorithm in our experiments as the exact MST algorithm.

Experiments were conducted on a PC with an Intel Core2 2.4 GHz CPU and 4 GB memory running Windows 7. The algorithm for testing the running time is implemented in C++, while the other tests are performed in Matlab (R2009b).

4.1. Running time

4.1.1. Running time on different datasets

We first perform experiments on four typical datasets with different sizes and dimensions to test the running time. The four datasets are described as Table 1.

Dataset t4.8k[1] is designed to test the CHAMELEON clustering algorithm in [28]. MNIST[2] is a dataset of ten handwriting digits and contains 60,000 training patterns and 10,000 test patterns of 784 dimensions, we use just the test set. The last two sets are from the UCI machine learning repository.[3] ConfLongDemo has eight attributes, of which only three numerical attributes are used here.

From each dataset, subsets with different sizes are randomly selected to test the running time as a function of data size. The subset sizes of the first two datasets gradually increase with step 20, the third with step 100 and the last with step 1000.

In general, the running time for constructing an MST of a dataset depends on the size of the dataset but not on the underlying structure of the dataset. In our FMST method, K-means is employed to partition a dataset, and the size of the subsets depends on the initialization of K-means and the distributions of the datasets, which leads to different time costs. We therefore perform FMST ten times on each dataset to alleviate the effects of the random initialization of K-means.

[1] http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download.
[2] http://yann.lecun.com/exdb/mnist.
[3] http://archive.ics.uci.edu/ml/.

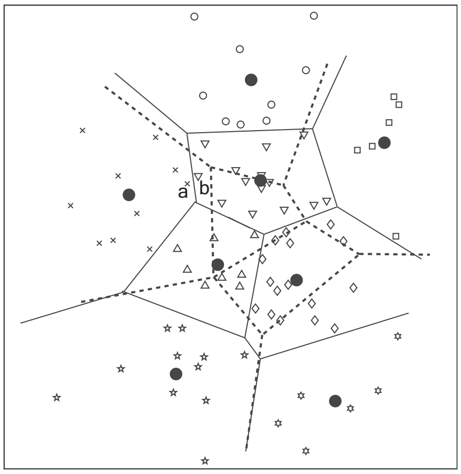

Fig. 6. Merge of two Voronoi graphs. Voronoi graph in solid line is corresponding to the first partition, and that in dashed line corresponding to the secondary partition. Only the first partition is illustrated.

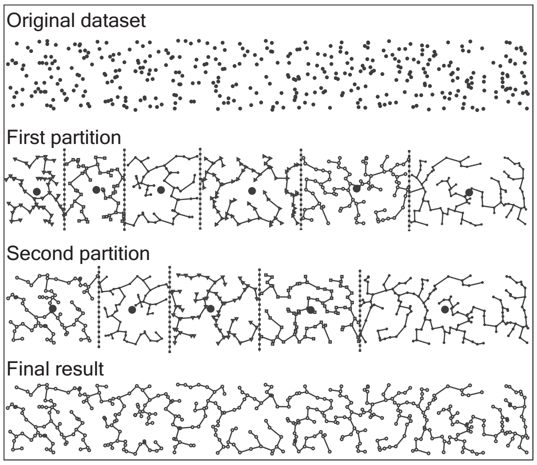

Fig. 7. The collinear Voronoi graph case.

Table 1
The description of four datasets.

	t4.8k	MNIST	ConfLongDemo	MiniBooNE
Data size	8000	10,000	164,860	130,065
Dimension	2	784	3	50

The running time of FMST and Prim's algorithm on the four datasets is illustrated in the first row of Fig. 8. From the results, we can see that FMST is computationally more efficient than Prim's algorithm, especially for the large datasets ConfLongDemo and MiniBooNE. The efficiency for MiniBooNE shown in the rightmost of the second and third row in Fig. 8, however, deteriorates because of the high dimensionality.

Although the complexity analysis indicates that the time complexity of the proposed FMST is $O(N^{1.5})$, the actual running time can be different. We analyzed the actual processing time by fitting an exponential function $T = aN^b$, where T is the running time and N is the number of data points. The results are shown in Table 2.

4.1.2. Running time with different Ks

We have discussed the number of clusters K and set it to $\sqrt{N}$ in Section 2.2.1, and have also presented some supporting theorems in Section 3. In practical applications, however, the value is slightly small. Some experiments were performed on

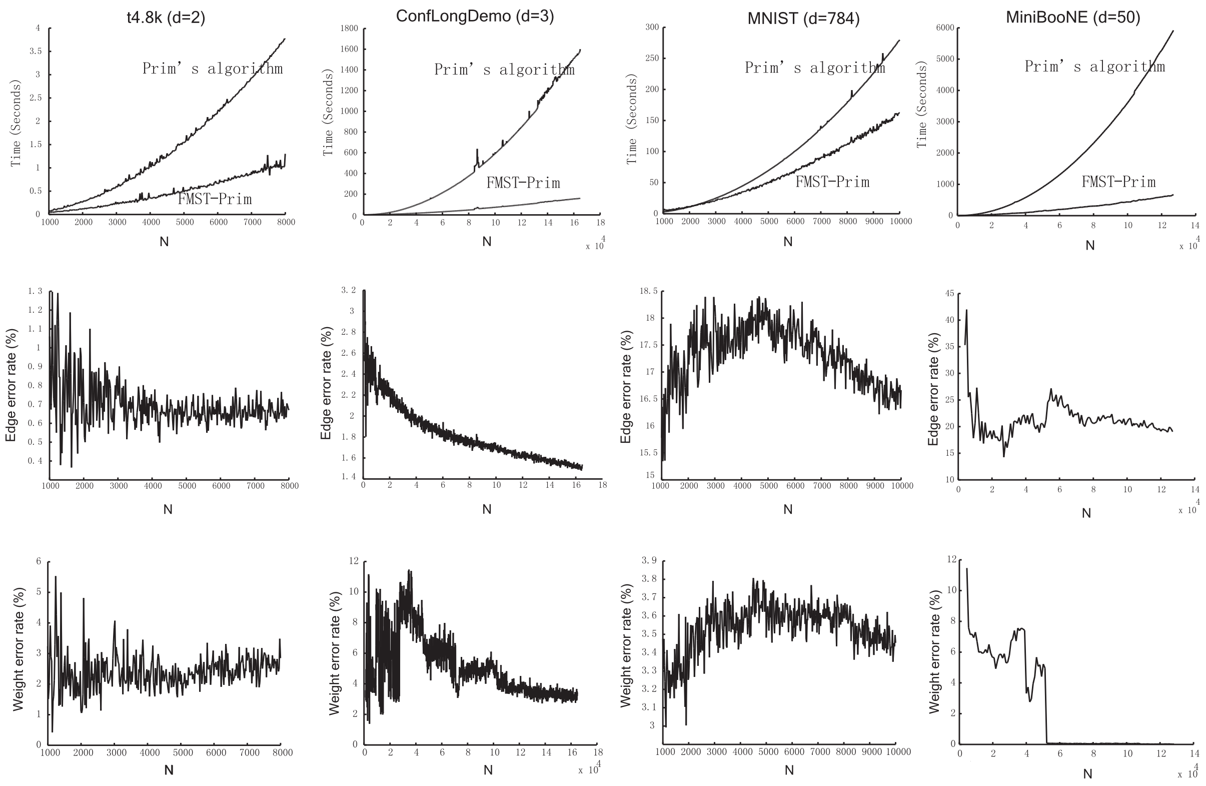

Fig. 8. The results of the test on the four datasets. FMST-Prime denotes the proposed method based on Prim's algorithm. The first row shows the running time of t4.8k, ConfLongDemo, MNIST and MiniBooNE, respectively. The second row shows corresponding edge error rates. The third row shows corresponding weight error rates.

dataset t4.8k and ConfLongDemo to study the effect of different Ks on running time. The experimental results are illustrated in Fig. 9, from which we find that if K is set to 38 for t4.8k and 120 for ConfLongDemo, the running time will be minimum. But according to the previous analysis, K would be set to $\sqrt{N}$, namely 89 and 406 for the two datasets, respectively. Therefore, K is practically set to $\frac{\sqrt{N}}{C}$, where $C > 1$. For dataset t4.8k and ConfLongDemo, C is approximately 3. The phenomenon is explained as follows.

From the analysis of the time complexity in Section 3, we can see that the main computational cost comes from K-means, in which a large K leads to a high cost. If partitions produced by K-means have the same size, when K is set to $\sqrt{N}$, the time complexity is minimized. However, the partitions practically have unbalanced sizes. From the viewpoint of divide-and-conquer, the proposed method with a large K will have a small time cost for constructing the meta-MSTs, but the unbalanced partitions can reduce this gain, and the large K only increases the time cost of K-means. Therefore, before K is increased to $\sqrt{N}$, the minimum time cost can be achieved.

4.2. Accuracy on synthetic datasets

4.2.1. Measures by edge error rate and weight error rate

The accuracy is another important aspect of FMST. Two accuracy rates are defined: edge error rate ER_{edge} and weight error rate ER_{weight}. Before ER_{edge} is defined, we present the notation of an equivalent edge of an MST, because the MST may not be unique. The equivalence property is described as:

Equivalence Property. Let T and T' be the two different MSTs of a dataset. For any edge $e \in (T \setminus T')$, there must exist another edge $e' \in (T' \setminus T)$ such that $(T' \setminus \{e'\}) \cup \{e\}$ is also an MST. We call e and e' a pair of equivalent edges.

Proof. The equivalency property can be operationally restated as: Let T and T' be the two different MSTs of a dataset, for any edge $e \in (T \setminus T')$, there must exist another edge $e' \in (T' \setminus T)$ such that $w(e) = w(e')$ and e connects T'_1 and T'_2, where T'_1 and T'_2 are the two subtrees generated by removing e' from T', $w(e)$ is the weight of e.

Let G be the cycle formed by $\{e\} \cup T'$, we have:

$$\forall e' \in (G \setminus \{e\} \setminus (T \cap T')), w(e) \geqslant w(e') \qquad (5)$$

Otherwise, an edge in $G \setminus \{e\} \setminus (T \cap T')$ should be replaced by e when constructing T'.

Table 2
The exponent *b*s obtained by fitting $T = aN^b$. FMST denotes the proposed method.

	b			
	t4.8k	MNIST	ConfLongDemo	MiniBooNE
FMST	1.57	1.62	1.54	1.44
Prim's alg.	1.88	2.01	1.99	2.00

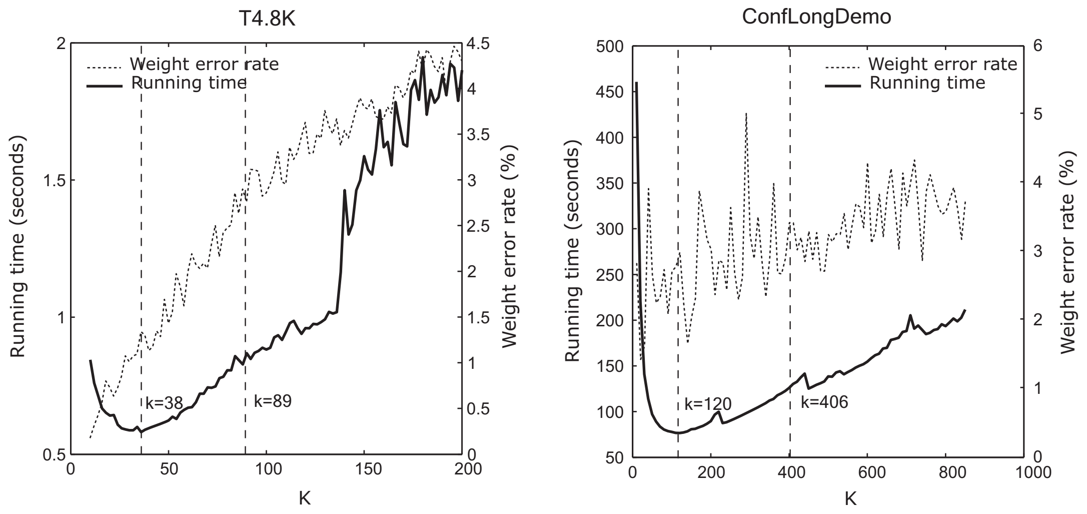

Fig. 9. Performances (running time and weight error rate) as a function of *K*. The left shows the running time and weight error rate of FMST on t4.8k, and the right on ConfLongDemo.

Furthermore, the following claim holds: there must exist at least one edge $e' \in (G \setminus \{e\} \setminus (T \cap T'))$, such that the cycle formed by $\{e'\} \cup T$ contains e. We prove this claim by contradiction.

Assuming that all the cycles G'_j formed by $\left\{e'_j\right\} \cup T$ do not contain e, where $e'_j \in (G \setminus \{e\} \setminus (T \cap T'))$, $1 \leqslant j \leqslant |G \setminus \{e\} \setminus (T \cap T')|$, let $G_{union} = (G'_1 \setminus \{e'_1\}) \cup \cdots \cup (G'_l \setminus \{e'_l\})$, where $l = |G \setminus \{e\} \setminus (T \cap T')|$. G can be expressed as $\{e\} \cup \{e'_1\} \cup \cdots \cup \{e'_l\} \cup G_{delta}$, where $G_{delta} \subset (T \cap T')$. As G is a cycle, $G_{union} \cup \{e\} \cup G_{delta}$ must also be a cycle, this is contradictory because $G_{union} \subset T, G_{delta} \subset T$ and $e \in T$. Therefore the claim is correct.

As a result, there must exist at least one edge $e' \in (G \setminus \{e\} \setminus (T \cap T'))$ such that $w(e') \geqslant w(e)$.

Combining this result with (5), we have the following: for $e \in (T \setminus T')$, there must exist an edge $e' \in (T' \setminus T)$ such that $w(e) = w(e')$. Furthermore, as e and e' are in the same cycle $G, (T' \setminus \{e'\}) \cup \{e\}$ is still an MST. □

According to the equivalency property, we define a criterion to determine whether an edge belongs to an MST:

Let T be an MST and e be an edge of a graph. If there exists an edge $e' \in T$ such that $|e| = |e'|$ and e connects T_1 and T_2, where T_1 and T_2 are the two subtrees achieved by removing e' from T, then e is a *correct edge*, i.e., belongs to an MST.

Suppose E_{appr} is the set of the correct edges in an approximate MST, the edge error rate ER_{edge} is defined as:

$$ER_{edge} = \frac{N - |E_{appr}| - 1}{N - 1} \tag{6}$$

The second measure is defined as the difference of the sum of the weights in FMST and the exact MST, which is called the weight error rate ER_{weight}:

$$ER_{weight} = \frac{W_{appr} - W_{exact}}{W_{exact}} \tag{7}$$

where W_{exact} and W_{appr} are the sum of the weights of the exact MST and FMST, respectively.

The edge error rates and weight error rates of the four datasets are shown in the third row of Fig. 8. We can see that both the edge error rate and the weight error rate decrease with the increase in data size. For datasets with high dimensions, the edge error rates are greater, for example, the maximum edge error rates of MNIST are approximately 18.5%, while those of t4.8k and ConfLongDemo are less than 3.2%. In contrast, the weight error rates decrease when the dimensionality increases. For instance, the weight error rates of MNIST are less than 3.9%. This is the phenomenon of the curse of dimensionality. The high dimensional case will be discussed further in Section 4.5.

4.2.2. Accuracy with different Ks

Globally, the edge and weight error rates increase with K. This is because the greater the K, the greater the number of split boundaries, from which the error edges come. But when K is small, the error rates increase slowly with K. In Fig. 9, we can see that the weight error rates are still low when K is set to approximate $\frac{\sqrt{N}}{3}$.

4.2.3. Comparison to other approaches

We first compare the proposed FMST with the approach in [34]. The approach in [34] is designed to detect the clusters efficiently by removing the longer edges of the MST, and an approximate MST is generated in the first stage.

The accuracy of the approximate MST produced in [34] is relevant to a parameter: the number of the nearest neighbors of a data point. This parameter is used to update the priority queue when an algorithm like Prim's is employed to construct an MST. In general, the larger the number, the more accurate the approximate MST. However, this parameter is also relevant to the computational cost of the approximate MST, which is $O(dN(b + k + k\log N))$, where k is the number of nearest neighbors and b is the number bits of a Hilbert number. Here we only focus on the accuracy of the method, and the number of nearest neighbors is set to $N * 0.05, N * 0.10, N * 0.15$, respectively. The accuracy is tested on t4k.8k, and the result is shown in Fig. 10. From the result, the edge error rates are more than 22%, and much higher than that of FMST, even if the number of nearest neighbors is set to $N * 0.15$, which leads to a loss in the computational efficiency of the method.

We then compare FMST with two other methods: MST using cover-tree by March et al. [35] and the divide-and-conquer approach by Wang et al. [45] on the following datasets: MNIST, ConfLongDemo, MiniBooNE and ConfLongDemo $\times$ 6. To compare the performances on a large data set, ConfLongDemo $\times$ 6 is generated. It has 989,160 data points, and is achieved as follows: Move two copies of ConfLongDemo to the right of the dataset along the first coordinate axis, and then copy the whole data and move the copy to the right along the second coordinate axis.

The results measured by running time (RT) and weight error rate in Table 3 confirm that Wang's approach is faster due to the recursive dividing of the data, but suffers from lower quality results, especially with the ConfLongDemo dataset, this is because the approach focuses on finding the longest edges of an MST in the early stage for efficient clustering but does not focus on constructing a high quality approximate MST. The method by March et al. is different and produces exact MSTs. It works very fast on lower dimensional datasets, but inefficiently on high dimensional data such as MNIST and MiniBooNE. FMST is slower than Wang's approach on all of the tested datasets, but has better quality. In [35], kd-tree and similar structures are used, which are known to work well with low-dimensional data. The proposed method is slower than March's method for lower dimensional datasets, but faster for the higher dimensional.

4.3. Accuracy on clustering

In this subsection, the accuracy of FMST is tested on a clustering application. Path-based clustering employs the minimax distance metric to measure the dissimilarities of data points [17,18]. For a pair of data points x_i, x_j, the minimax distance D_{ij} is defined as:

$$D_{ij} = \min_{\mathcal{P}_{ij}^k} \left\{ \max_{(x_p, x_{p+1}) \in \mathcal{P}_{ij}^k} d(x_p, x_{p+1}) \right\} \tag{8}$$

where $\mathcal{P}_{ij}^k$ denotes all possible paths between x_i and x_j and k is an index to enumerate the paths, and $d(x_p, x_{p+1})$ is the Euclidean distance between x_p and x_{p+1}.

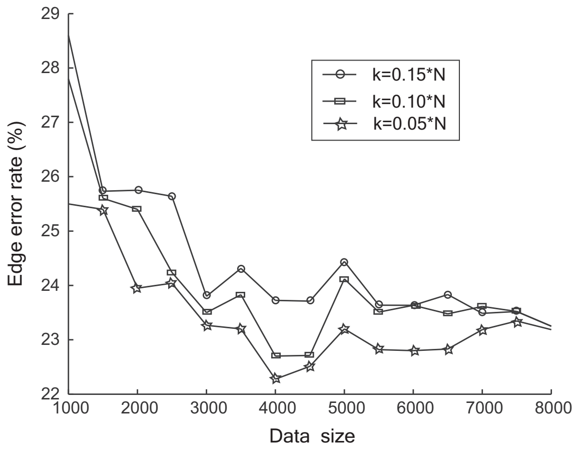

Fig. 10. The edge error rate of Lai's method on t4.8k.

Table 3
The proposed method FMST is compared to MST-Wang [45] and MST-March [35] methods.

Methods	MNIST		MiniBooNE		ConfLongDemo		ConfLong Demo × 6	
	RT(S)	ER_{weight} (%)	RT(S)	ER_{weight} (%)	RT(S)	ER_{weight} (%)	RT(S)	ER_{weight} (%)
FMST	164	3.3	781	0.3	174	0.5	16,201	0.2
MST-Wang	26	43.4	64	40.5	51	38.2	5262	46.8
MST-March	1135	0	2181	0	18	0	133	0

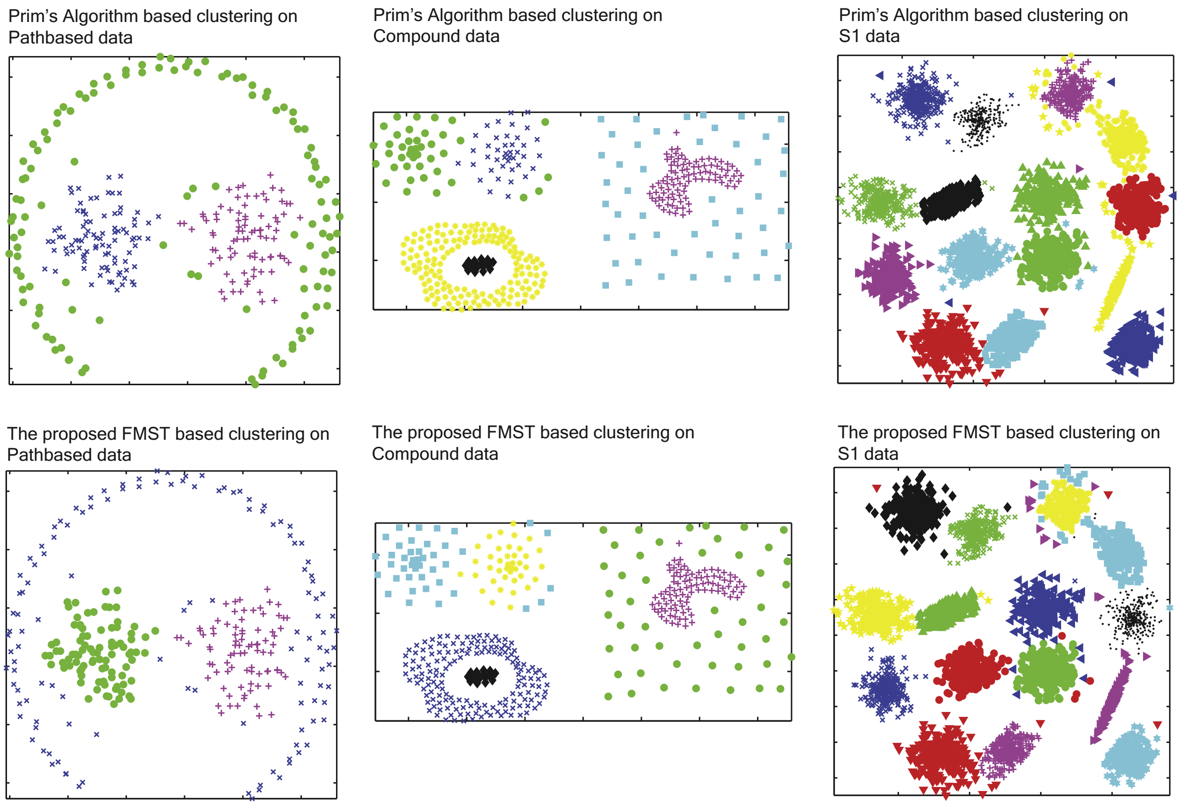

Fig. 11. Prim's algorithm and the proposed FMST based clustering results.

Table 4
The quantitative measures of clustering results. FMST denotes the proposed method.

Datasets	FMST				Prim's algorithm			
	Rand	AR	Jac	FM	Rand	AR	Jac	FM
Pathbased	0.937	0.859	0.829	0.906	0.942	0.870	0.841	0.913
Compound	0.993	0.982	0.973	0.986	0.994	0.984	0.977	0.988
S1	0.995	0.964	0.936	0.967	0.995	0.964	0.935	0.966

The minimax distance can be computed by an all-pair shortest path algorithm, such as the Floyd Warshall algorithm. However, this algorithm runs in time $O(N^3)$. An MST can be used to compute the minimax distance more efficiently in [31]. To make the path-based clustering robust to outliers, Chang and Yeung [8] improved the minimax distance and incorporated it into spectral clustering. We tested the FMST within this method on three synthetic datasets (Pathbased, Compound and S1).[4]

For computing the minimax distances, Prim's algorithm and FMST are used. In Fig. 11, one can see that the clustering results on three datasets are almost the same. The quantitative measures are given in Table 4, which contains four validity

[4] http://cs.joensuu.fi/sipu/datasets/.

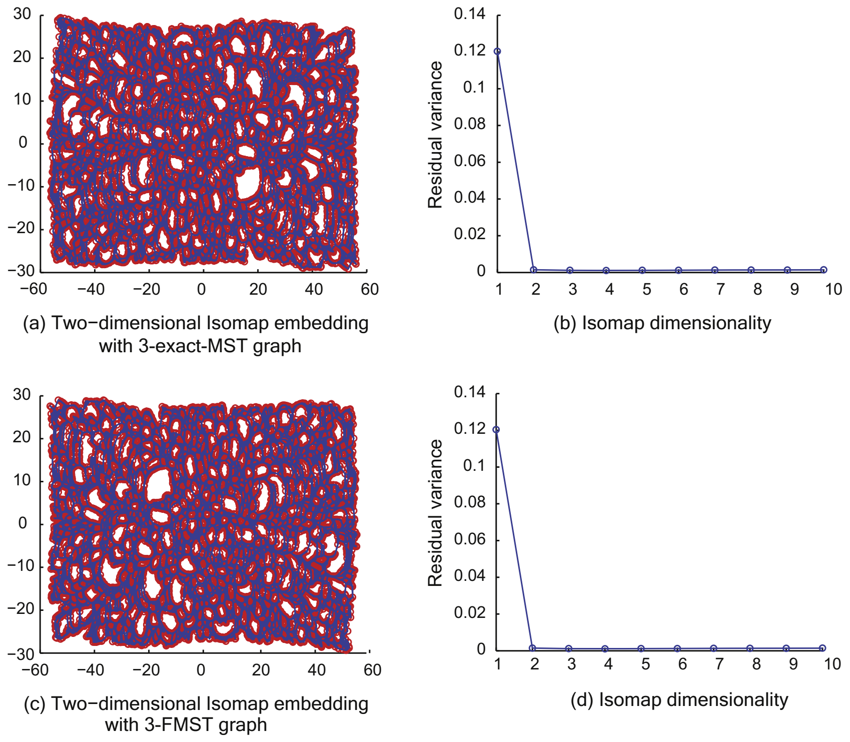

Fig. 12. Two 3-MST graph based ISOMAP results using exact MST (Prim's algorithm) and FMST, respectively. In (a) and (c), the two dimensional embedding is illustrated. (b) and (d) are corresponding resolutions.

indexes and indicates that the results on the first two datasets of Prim's algorithm-based clustering are slightly better than those of the FMST-based clustering.

4.4. Accuracy on manifold learning

MST has been used for manifold learning [48,49]. For a KNN based neighborhood graph, an improperly selected k may lead to a disconnected graph, and degrade the performance of manifold learning. To address this problem, Yang [48] used MSTs to construct a k-edge connected neighborhood graph. We implement the method of [48], with exact MST and FMST respectively, to reduce the dimensionality of a manifold.

The FMST-based and the exact MST-based dimensionality reduction were performed on the dataset Swiss-roll, which has 20,000 data points. In experiments, we selected the first 10,000 data points because of the memory requirement, and set $k = 3$. The accuracy of the FMST-based dimensionality reduction is compared with that of an exact MST-based dimensionality reduction in Fig. 12. The intrinsic dimensionality of Swiss-roll can be detected by the "elbow" of the curves in (b) and (d). Obviously, the MST graph based method and the FMST graph based method have almost identical residual variance, and both indicate the intrinsic dimensionality is 2. Furthermore, Fig. 12(a) and (c) shows that the two methods have similar two-dimensional embedding results.

4.5. Discussion on high dimensional datasets

As described in the experiments, the performances of both computation and accuracy of the proposed method are reduced when applied to high-dimensional datasets. Since the time complexity of FMST is $O(N^{1.5})$ under the condition of $d \ll N$, when the number of dimensions d is becoming large and even approximate to N, the computational cost will degrade to $O(N^{2.5})$. However, it is still more efficient than the corresponding Kruskal's or Prim's algorithms.

The accuracy of FMST is reduced because of the curse of dimensionality, which includes distance concentration phenomenon and the hubness phenomenon [40]. The distance concentration phenomenon is that the distances between all pairs of data points from a high dimensional dataset are almost equal, in other words, the traditional distance measures become ineffective, and the distances computed with the measures become unstable [25]. For constructing an MST in terms of these distances, the results of Kruskal's or Prim's algorithm are meaningless, so is the accuracy of the proposed FMST. Furthermore, the hubness phenomenon in a high-dimensional dataset, which implies some data points may appear in many more KNN lists than other data points, shows that the nearest neighbors also become meaningless. Obviously, hubness affects the construction of an MST in the same way.

The intuitive way to address the above problems caused by the curse of dimensionality is to employ dimensionality reduction methods, such as ISOMAP, LLE, or subspace based methods for a concrete task in machine learning, such as subspace based clustering. Similarly, for constructing an MST of a high dimensional dataset, one may preprocess the dataset with dimensionality reduction or subspace based methods for the purpose of getting more meaningful MSTs.

5. Conclusion

In this paper, we have proposed a fast MST algorithm with a divide-and-conquer scheme. Under the assumption that the dataset is partitioned into equal sized subsets in the divide step, the time complexity of the proposed algorithm is theoretically $O(N^{1.5})$. Although this assumption may not hold practically, the complexity is still approximately $O(N^{1.5})$. The accuracy of the FMST was analyzed experimentally using edge error rate and weight error rate. Furthermore, two practical applications were considered, and the experiments indicate that the proposed FMST can be applied to large datasets.

Acknowledgments

This work was partially supported by the Natural Science Foundation of China (No. 61175054), the Center for International Mobility (CIMO), and sponsored by K.C. Wong Magna Fund in Ningbo University.

References

[1] H. Abdel-Wahab, I. Stoica, F. Sultan, K. Wilson, A simple algorithm for computing minimum spanning trees in the internet, Inform. Sci. 101 (1997) 47–69.
[2] L. An, Q.S. Xiang, S. Chavez, A fast implementation of the minimum spanning tree method for phase unwrapping, IEEE Trans. Med. Imag. 19 (2000) 805–808.
[3] B. Awerbuch, Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems, in: Proceedings of the 19th ACM Symposium on Theory of Computing, 1987.
[4] D.A. Bader, G. Cong, Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs, J. Paral. Distrib. Comput. 66 (2006) 1366–1378.
[5] J.C. Bezdek, N.R. Pal, Some new indexes of cluster validity, IEEE Trans. Syst., Man Cybernet., Part B 28 (1998) 301–315.
[6] O. Borůvka, O jistém problému minimálním (About a Certain Minimal Problem), Práce moravské přírodovědecké společnosti v Brně III (1926) 37–58 (in Czech with German summary).
[7] P.B. Callahan, S.R. Kosaraju, Faster algorithms for some geometric graph problems in higher dimensions, in: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete algorithms, 1993.
[8] H. Chang, D.Y. Yeung, Robust path-based spectral clustering, Patt. Recog. 41 (2008) 191–203.
[9] B. Chazelle, A minimum spanning tree algorithm with inverse-Ackermann type complexity, J. ACM 47 (2000) 1028–1047.
[10] G. Chen et al, The multi-criteria minimum spanning tree problem based genetic algorithm, Inform. Sci. 177 (2007) 5050–5063.
[11] D. Cheriton, R.E. Tarjan, Finding minimum spanning trees, SIAM J. Comput. 5 (1976) 24–742.
[12] K.W. Chong, Y. Han, T.W. Lam, Concurrent threads and optimal parallel minimum spanning trees algorithm, J. ACM 48 (2001) 297–323.
[13] G. Choquet, Etude de certains réseaux de routes, Comptesrendus de l'Acadmie des Sciences 206 (1938) 310 (in French).
[14] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., The MIT Press, 2001.
[15] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1959) 269–271.
[16] Q. Du, V. Faber, M. Gunzburger, Centroidal Voronoi tessellations: applications and algorithms, SIAM Rev. 41 (1999) 637–676.
[17] B. Fischer, J.M. Buhmann, Path-based clustering for grouping of smooth curves and texture segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 25 (2003) 513–518.
[18] B. Fischer, J.M. Buhmann, Bagging for path-based clustering, IEEE Trans. Pattern Anal. Mach. Intell. 25 (2003) 1411–1415.
[19] K. Florek, J. Łkaszewicz, H. Perkal, H. Steinhaus, S. Zubrzycki, Sur la liaison et la division des points d'un ensemble fini, Colloq. Mathemat. 2 (1951) 282–285.
[20] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM 34 (1987) 596–615.
[21] H.N. Gabow, Z. Galil, T.H. Spencer, R.E. Tarjan, Efficient algorithms for finding minimum spanning trees in undirected and directed graphs, Combinatorica 6 (1986) 109–122.
[22] H.N. Gabow, Z. Galil, T.H. Spencer, Efficient implementation of graph algorithms using contraction, J. ACM 36 (1989) 540–572.
[23] R.G. Gallager, P.A. Humblet, P.M. Spira, A distributed algorithm for minimum-weight spanning trees, ACM Trans. Program. Lang. Syst. 5 (1983) 66–77.
[24] J.C. Gower, G.J.S. Ross, Minimum spanning trees and single linkage cluster analysis, J. R. Statist. Soc., Ser. C (Appl. Statist.) 18 (1969) 54–64.
[25] C.M. Hsu, M.S. Chen, On the design and applicability of distance functions in high-dimensional data space, IEEE Trans. Knowl. Data Eng. 21 (2009) 523–536.
[26] V. Jarník, O jistém problému minimálním (About a certain minimal problem), Práce moravské přírodovědecké společnosti v Brně VI (1930) 57–63 (in Czech).
[27] P. Juszczak, D.M.J. Tax, E. Pękalska, R.P.W. Duin, Minimum spanning tree based one-class classifier, Neurocomputing 72 (2009) 1859–1869.
[28] G. Karypis, E.H. Han, V. Kumar, CHAMELEON: a hierarchical clustering algorithm using dynamic modeling, IEEE Trans. Comput. 32 (1999) 68–75.
[29] M. Khan, G. Pandurangan, A fast distributed approximation algorithm for minimum spanning trees, Distrib. Comput. 20 (2008) 391–402.
[30] K. Li, S. Kwong, J. Cao, M. Li, J. Zheng, R. Shen, Achieving balance between proximity and diversity in multi-objective evolutionary algorithm, Inform. Sci. 182 (2012) 220–242.
[31] K.H. Kim, S. Choi, Neighbor search with global geometry: a minimax message passing algorithm, in: Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 401–408.
[32] J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, Proc. Am. Math. Soc. 7 (1956) 48–50.
[33] B. Lacevic, E. Amaldi, Ectropy of diversity measures for populations in Euclidean space, Inform. Sci. 181 (2011) 2316–2339.
[34] C. Lai, T. Rafa, D.E. Nelson, Approximate minimum spanning tree clustering in high-dimensional space, Intell. Data Anal. 13 (2009) 575–597.
[35] W.B. March, P. Ram, A.G. Gray, Fast euclidean minimum spanning tree: algorithm, analysis, and applications, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2010.
[36] T. Öncan, Design of capacitated minimum spanning tree with uncertain cost and demand parameters, Inform. Sci. 177 (2007) 4354–4367.
[37] D. Peleg, V. Rubinovich, A near tight lower bound on the time complexity of distributed minimum spanning tree construction, SIAM J. Comput. 30 (2000) 1427–1442.
[38] S. Pettie, V. Ramachandran, A randomized time-work optimal parallel algorithm for finding a minimum spanning forest, SIAM J. Comput. 31 (2000) 1879–1895.
[39] R.C. Prim, Shortest connection networks and some generalizations, Bell Syst. Tech. J. 36 (1957) 567–574.

[40] M. Radovanovic, A. Nanopoulos, M. Ivanovic, Hubs in space: popular nearest neighbors in high-dimensional data, J. Mach. Learn. Res. 11 (2010) 2487–2531.
[41] M.R. Rezaee, B.P.F. Lelieveldt, J.H.C. Reiber, A new cluster validity index for the fuzzy c-mean, Patt. Recog. Lett. 19 (1998) 237–246.
[42] M. Sollin, Le trace de canalisation, in: C. Berge, A. Ghouilla-Houri (Eds.), Programming, Games, and Transportation Networks, Wiley, New York, 1965 (in French).
[43] S. Sundar, A. Singh, A swarm intelligence approach to the quadratic minimum spanning tree problem, Inform. Sci. 180 (2010) 3182–3191.
[44] P.M. Vaidya, Minimum spanning trees in k-dimensional space, SIAM J. Comput. 17 (1988) 572–582.
[45] X. Wang, X. Wang, D.M. Wilkes, A divide-and-conquer approach for minimum spanning tree-based clustering, IEEE Trans. Knowl. Data Eng. 21 (2009) 945–958.
[46] Y. Xu, V. Olman, D. Xu, Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees, Bioinformatics 18 (2002) 536–545.
[47] Y. Xu, E.C. Uberbacher, 2D image segmentation using minimum spanning trees, Image Vis. Comput. 15 (1997) 47–57.
[48] L. Yang, k-Edge Connected neighborhood graph for geodesic distance estimation and nonlinear data projection, in: Proceedings of the 17th International Conference on Pattern Recognition, ICPR'04, 2004.
[49] L. Yang, Building k edge-disjoint spanning trees of minimum total length for isometric data embedding, IEEE Trans. Patt. Anal. Mach. Intell. 27 (2005) 1680–1683.
[50] A.C. Yao, An $O(|E|\log\log|V|)$ algorithm for finding minimum spanning trees, Inform. Process. Lett. 4 (1975) 21–23.
[51] C.T. Zahn, Graph-theoretical methods for detecting and describing gestalt clusters, IEEE Trans. Comp. C20 (1971) 68–86.
[52] C. Zhong, D. Miao, R. Wang, A graph-theoretical clustering method based on two rounds of minimum spanning trees, Patt. Recog. 43 (2010) 752–766.
[53] C. Zhong, D. Miao, P. Fränti, Minimum spanning tree based split-and-merge: a hierarchical clustering method, Inform. Sci. 181 (2011) 3397–3410.
[54] C. Zhong, M. Malinen, D. Miao, P. Fränti, Fast approximate minimum spanning tree algorithm based on K-means, in: 15th International Conference on Computer Analysis of Images and Patterns, York, UK, 2013.