Fast Approximate Minimum Spanning Tree Algorithm Based on K-Means

Caiming Zhong[1,2,3], Mikko Malinen[2], Duoqian Miao[1], and Pasi Fränti[2]

[1] Department of Computer Science and Technology, Tongji University, Shanghai 201804, PR China
[2] Department of Computer Science, University of Eastern Finland, P.O. Box 111, FIN-80101 Joensuu, Finland
[3] College of Science and Technology, Ningbo University, Ningbo 315211, PR China

Abstract. We present a fast approximate Minimum spanning tree(MST) framework on the complete graph of a dataset with N points, and any exact MST algorithm can be incorporated into the framework and speeded up. It employs a divide-and-conquer scheme to produce an approximate MST with theoretical time complexity of $O(N^{1.5})$, if the incorporated exact MST algorithm has the running time of $O(N^2)$. Experimental results show that the proposed approximate MST algorithm is computational efficient, and the accuracy is close to the true MST.

Keywords: Minimum spanning tree, divide-and-conquer, K-means.

1 Introduction

Given an undirected and weighted graph, the problem of MST is to find a spanning tree such that the sum of weights is minimized. Since MST can roughly estimate the intrinsic structure of a dataset, it has been broadly applied in image segmentation [1], cluster analysis [9], classification [4], manifold learning [8]. However, traditional MST algorithms such as Prim's and Kruskal's algorithm have running time of $O(N^2)$ [3], and for a large dataset a fast MST algorithm is needed.

Recent work to find an approximate MST can be found in [6][7], and the both work apply MSTs to clustering. Wang et al. [7] employ divide-and-conquer scheme to detect the long edges of the MST at an early stage for clustering. Initially, data points are randomly stored in a list, and each data point is connected to its predecessor (or successor), and a spanning tree is achieved. To optimize the spanning tree, the dataset is divided into a collection of subsets with a divisive hierarchical clustering algorithm. The distance between any pair of data points within a subset can be computed by a brute force nearest neighbor search, and with the distances, the spanning tree is updated.

Lai et al. [6] proposed an approximate MST algorithm based on Hilbert curve for clustering. It is a two-phase algorithm: the first phase is to construct an approximate MST of a given dataset with Hilbert curve, and the second phase is to partition the dataset into subsets by measuring the densities of points along

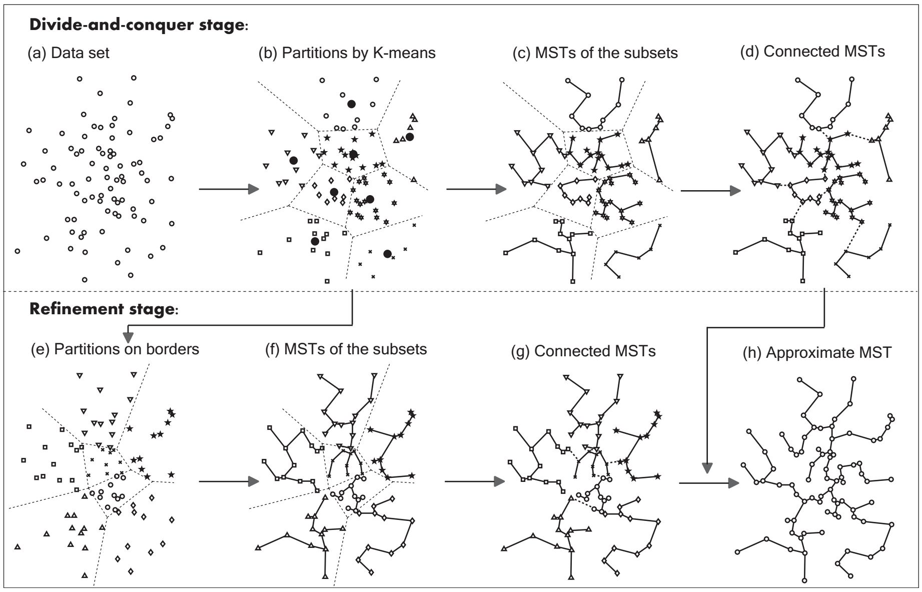

Fig. 1. The scheme of the proposed fast MST algorithm. (a) A given dataset. (b) The dataset is partitioned into $\sqrt{N}$ subsets by K-means. (c) An exact MST algorithm is applied to each subset. (d) MSTs of the subsets are connected. (e) The dataset is partitioned again so that the neighboring data points in different subsets are partitioned into identical partitions. (f) Exact MST 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.

the approximate MST with a specified density threshold. However, the accuracy of MST depends on the order of Hilbert Curve and the number of neighbors of a visited point in the linear list.

2 Proposed Method

2.1 Overview of the Proposed Framework

To improve the efficiency of constructing an MST is to reduce the unnecessary comparisons. For example, in Kruskal's algorithm, it is not necessary to sort all $N(N-1)/2$ edges of a complete graph but to find $(1+\alpha)N$ edges with least weights, where $(N-3)/2 \gg \alpha \geq -1/N$. We employ a divide-and-conquer technique to achieve the improvement. The overview of the proposed method is illustrated in Fig. 1.

2.2 Partition Dataset with K-Means

In general, a data point in an MST is connected to its nearest neighbors, which implies that the connections have a locality property. In the divide step, it is therefore expected that the subsets preserve this locality. Since K-means can partition local neighboring data points into the same group, we employ K-means to partition the dataset.

The Number of Clusters *K*. In our method, the number of clusters K is set to $\sqrt{N}$. There are two reasons for this determination. One is that the maximum number of clusters in some clustering algorithms is often set to $\sqrt{N}$ as a rule of thumb [2]. That means if a dataset is partitioned into $\sqrt{N}$ subsets, each subset will consist of data points coming from an identical genuine cluster, which satisfies the requirement of the locality property when constructing an MST. 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.

Divide and Conquer Algorithm. After the dataset is 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 algorithm. The algorithm of K-means based 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

```
1. Set the number of subsets $K = \sqrt{N}$.
2. Apply $K$-means to $X$ to achieve $K$ subsets $S = \{S_1, \ldots, S_K\}$.
3. Apply an exact MST algorithm to $S_i$, and its MST $MST(S_i)$ is obtained.
```

2.3 Combine MSTs of the *K* Subsets

An intuitive solution to combine MSTs is brute force: for the MST of a cluster, the shortest edge between it and MSTs of other clusters is computed. But this solution is time consuming, and therefore a fast MST-based effective combination is presented.

MST-Based Combination. The neighboring subsets are determined first because the MSTSs of those far away from each other will not be connected. This can be achieved by MST of the centers of the subsets, see Fig. 2. To connect a pair of neighboring subsets efficiently, the nearest point of one subset to the center of the other is selected. For example, a and b are the nearest points to opposite centers respectively, and they are connected.

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

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

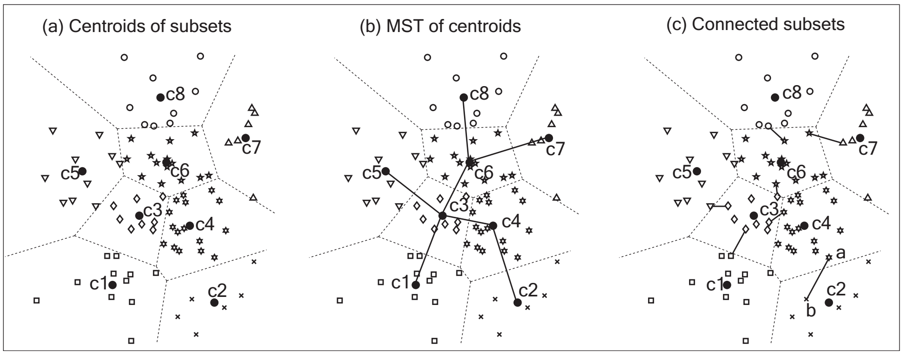

Fig. 2. The combine step of MSTs of the proposed algorithm. In (a), centers of the partitions (c1, ..., c8) are calculated. In (b), a MST of the centers, MST_{cen}, is constructed with an exact MST algorithm. In (c), each pair of subsets whose centers are neighbors with respect to MST_{cen} in (b) is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

1. Compute the center c_i of subset S_i, $1 \leq i \leq K$.
2. Construct an MST, MST_{cen}, of $c_1, \cdots, c_K$ by an exact MST algorithm.
3. For each pair of subsets (S_i, S_j) whose centers are connected by an edge of MST_{cen}, discover the edge by **DCE** that connects $MST(S_i)$ and $MST(S_j)$.
4. Combine discovered edges with $MST(S_1), \cdots, MST(S_K)$ to achieve MST_1.

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 $MST(S_j)$;

1. Find data point $a \in S_i$ so that the distance between a and c_j is minimized.
2. Find data point $b \in S_j$ so that the distance between b and c_i is minimized.
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 not enough. 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. Based on this observation, the refinement stage is designed.

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}. In most cases, these midpoints lie near the boundaries, and are therefore employed as the initial cluster centers. The dataset is then partitioned by K-means, in which only one iteration is performed for the purpose of focusing on the boundaries. The process is illustrated in Fig. 3.

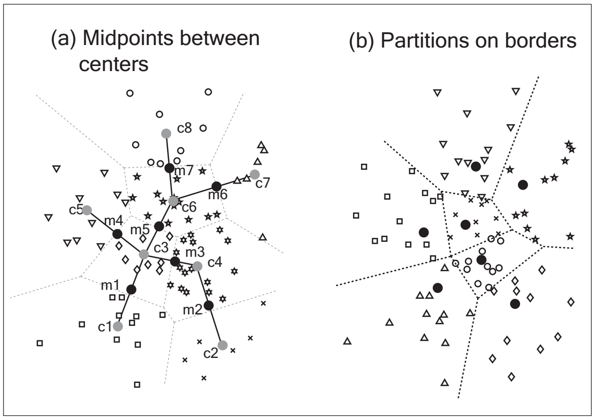

Fig. 3. Boundary-based partition. In (a), the black solid points, $m_1, \cdots, m_7$, are the midpoints of the edges of MST_{cen}. In (b), each data point is assigned to its nearest midpoint, and the dataset is partitioned by the midpoints. The corresponding Voronoi graph is with respect to the midpoints.

Build Secondary Approximate MST. After the dataset has been re-partitioned, the conquer and combine steps similar to those in first stage are used to produce the secondary approximate MST. The algorithm is summarized as follows:

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

1. Compute the midpoint m_i of an edge $e_i \in MST_{cen}$, where $1 \leq i \leq K-1$.
2. Partition dataset X into $K-1$ subsets, $S'_1, \cdots, S'_{K-1}$, by assigning each point to its nearest point from $m_1, \cdots, m_{K-1}$.
3. Build MSTs, $MST(S'_1), \cdots, MST(S'_{K-1})$, with an exact MST algorithm.
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 on this graph to achieve the final approximate MST of X.

3 Complexity and Accuracy Analysis

3.1 Complexity Analysis

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

$$T_{FMST} = T_{DAC} + T_{CA} + T_{SAM} + T_{COM} \tag{1}$$

where T_{DAC}, T_{CA} and T_{SAM} are the time complexities of the algorithms **DAC**, **CA** and **SAM** respectively, 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 into K subsets and constructing the MSTs of the subsets with an exact MST algorithm. Since $K = \sqrt{N}$, we have $T_{DAC} = O(N^{1.5})$. In **CA**, computing the mean points of the subsets and constructing MST of the K mean points take only $O(N)$ time. For each connected subset pair, determining the connecting edge requires $O(2N \times (K-1)/K)$. The total computational cost of **CA** is therefore $O(N)$.

In **SAM**, Computing $K-1$ midpoints and partitioning the dataset take $O(N\times (K-1))$ time. The running time of Step 3 and 4 is $O((K-1) \times N^2/(K-1)^2) = O(N^2/(K-1))$ and $O(N)$, respectively. 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 complexity of the proposed fast algorithm is $O(N^{1.5})$.

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 four datasets: t4.8k [5], MNIST [10], ConfLongDemo [11] and MiniBooNE [11]. Experiments were conducted on a PC with an Intel Core2 2.4GHz CPU and 4GB memory running Windows 7.

4.1 Running Time

From each dataset, subsets with different size 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.

The running time of FMST and Prim's algorithm on the four datasets is illustrated in the first row of Fig. 4. 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. 4, however, deteriorates because of the high dimensionality. Although the complexity analysis indicates that the time complexity of proposed FMST is $O(N^{1.5})$, the actual running time may be different because K-means can not produce clusters being of equal size. We analyze 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 the results are shown in Table 1.

4.2 Accuracy

Suppose E_{appr} is the set of the correct edges in an approximate MST, the edge error rate ER_{edge} is defined as: $ER_{edge} = \frac{N-|E_{appr}|-1}{N-1}$. The second measure is

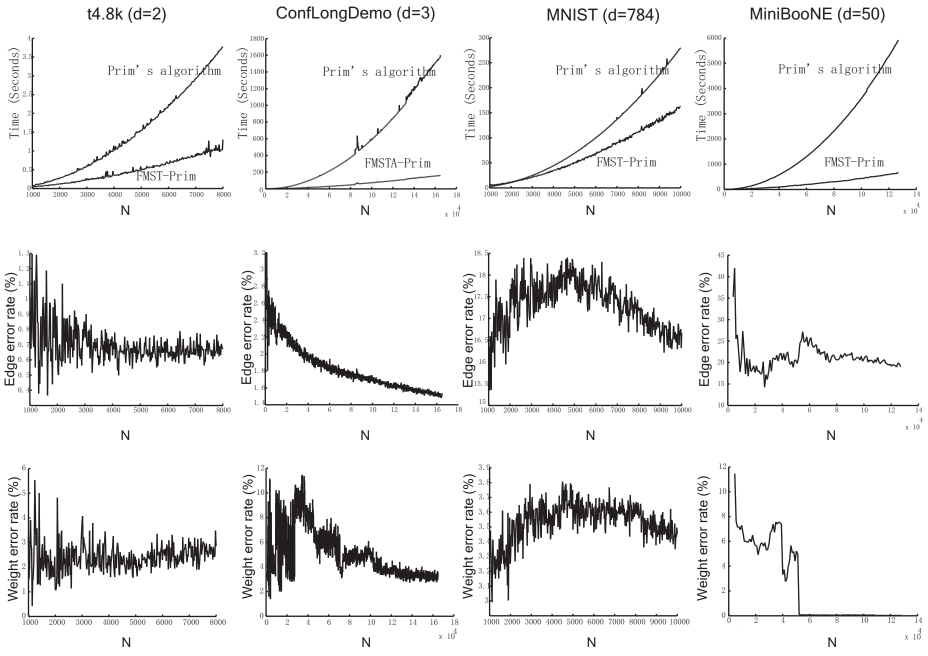

Fig. 4. The results of the test on the four datasets

Table 1. The exponent bs obtained by fitting $T = aN^b$

	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

defined as the differ of the sum of the weights in FMST and the exact MST, which is called weight error rate: $ER_{weight} = \frac{W_{appr} - W_{exact}}{W_{exact}}$, where W_{exact} and W_{appr} are the sum of 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. 4. We can see that both the edge error rate and the weight error rate decrease with the increase of the data size. For datasets with high dimension, the edge error rates are bigger, for example, the maximum edge error rates of MNIST are approximate to 18.5%, while those of t4.8k and ConfLongDemo less than 3.2%. In contrast, the weight error rates decrease when the dimensionality increases. This is one aspect of the curse of dimensionality, *distance concentration*, which means that Euclidean distances between all pairs of points in high dimensional data are tend to be similar.

5 Conclusion

In this paper, we have proposed a fast approximate MST algorithm with a divide and conquer scheme. The time complexity of the proposed algorithm is theoretically $O(N^{1.5})$. Furthermore, any MST algorithm can be incorporated into to the proposed framework to make it more efficient.

References

1. An, L., Xiang, Q.S., Chavez, S.: A fast implementation of the minimum spanning tree method for phase unwrapping. IEEE Trans. Medical Imaging 19, 805–808 (2000)
2. Bezdek, J.C., Pal, N.R.: Some new indexes of cluster validity. IEEE Trans. Systems, Man and Cybernetics, Part B 28, 301–315 (1998)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press (2001)
4. Juszczak, P., Tax, D.M.J., Pękalska, E., Duin, R.P.W.: Minimum spanning tree based one-class classifier. Neurocomputing 72, 1859–1869 (2009)
5. Karypis, G., Han, E.H., Kumar, V.: CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. IEEE Trans. Comput. 32, 68–75 (1999)
6. Lai, C., Rafa, T., Nelson, D.E.: Approximate minimum spanning tree clustering in high-dimensional space. Intelligent Data Analysis 13, 575–597 (2009)
7. Wang, X., Wang, X., Wilkes, D.M.: A divide-and-conquer approach for minimum spanning tree-based clustering. IEEE Trans., Knowledge and Data Engineering 21, 945–958 (2009)
8. Yang, L.: Building k edge-disjoint spanning trees of minimum total length for isometric data embedding. IEEE Trans. Pattern Analysis and Machine Intelligence 27, 1680–1683 (2005)
9. Zhong, C., Miao, D., Wang, R.: A graph-theoretical clustering method based on two rounds of minimum spanning trees. Pattern Recognition 43, 752–766 (2010)
10. `http://yann.lecun.com/exdb/mnist`
11. `http://archive.ics.uci.edu/ml/`