Progressive Algorithm For Euclidean Minimum Spanning Tree

Amir Mesrikhani [*] Mohammad Farshi [*] Mansoor Davoodi [†]

Abstract

Designing efficient algorithms that process massive data is a challenging task. The progressive algorithms are methods to handle massive data efficiently. In these algorithms, partial solutions are reported to user in some middle steps that approximates the final solution. The user can decide whether to continue the running of the algorithm based on the error of the partial solutions. In this paper, we propose a progressive algorithm for computing Euclidean minimum spanning tree of a set of n points in the plane that consists of $O(\log n)$ steps. The error of the partial solution in step r is $O(1 - \frac{4^r}{n-1}\alpha^{-1})$, where α is the aspect ratio of the point set.

1 Introduction

One method to process massive data efficiently is designing algorithms that solve a problem with a massive input data interactively with users. Progressive algorithms are one of these interactive methods. In these algorithms, partial solutions, whose error are measured by an error function (err), are reported to user in particular steps. The error function err takes a partial solution as an argument and returns non-negative value that represents the error value of the argument. Based on the error value of the partial solution in each step, the user can decide to stop the algorithm or continue toward a partial solution with smaller error value. The convergence speed of the partial solutions to the final solution is determined by a convergence function f_{conv}. The function f_{conv} takes step number r as an argument and returns an upper bound of the error value of the partial solution in step r.

Formally, in 2015, Alewijnse et al. [1] introduced the following defintion for progressive algorithms:

Definition 1 *A progressive algorithm is an algorithm that produces partial solution s_r in step r such that:*

$$err(s_r) \leq f_{conv}(r).$$

[*]Combinatorial and Geometric Algorithms Lab., Department of Mathematical Sciences, Yazd University, Yazd, Iran, `mesrikhani@stu.yazd.ac.ir, mfarshi@yazd.ac.ir (corresponding author)`

[†]Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, Iran, `mdmonfared@iasbs.ac.ir`

Computing Euclidean Minimum Spanning Tree (EMST) of a set of points is a classical problem that has many applications in designing low-cost road network and designing VLSI circuit. In this paper, we aim to design a progressive algorithm to compute EMST of a set of points in the plane.

1.1 Our results

For a set of n points in the plane, we propose a progressive algorithm for computing EMST with $O(\log_4 n)$ steps and the most time-consuming step takes $O(n^2)$ time. In step r, our algorithm produces a partial solution whose error is $O(1-\frac{4^r}{n-1}\alpha^{-1})$. The value α denotes the apsect ratio of input point set which is the ratio of the maximum pairwise distance and the minimum pairwise distance of the input points.

1.2 Related work

The framework of the progressive algorithm was introduced by Alewijnse et al. in 2015 [1]. They studied some fundamental problems in computational geometry like finding convex hull of a set of points and computing k-popular regions for a set of trajectories. Also, progressive algorithms are studied in other contexts and the results could be found in [9, 10]. As aforementioned, a progressive algorithm produces approximation solution in each step . Several approximation algorithms have been proposed for computing Minimum Spanning Tree (MST) [8]. Clarkson et al. [6] studied finding EMST of a set of points in $\mathbb{R}^d$. They proposed $(1 + \varepsilon)$-approximation algorithm that takes $O(n(\log n + (1/\varepsilon)\log \alpha))$ time for $d = 3$. Czmuaj et al. [7] focused on approximating the weight of EMST in sublinear time. Their algorithm approximates the weight in $\mathbb{R}^d$ in $O(\sqrt{n}poly(1/\varepsilon))$ time with high probability, where $poly$ denotes a polynomial function based on $1/\varepsilon$. For a given connected graph of an average degree d, Chazelle et.al [5] presented probabilistic algorithm in $O(d\omega\varepsilon^{-2}\log\frac{d\omega}{\varepsilon})$ time to approximate the weight of MST with error at most ε, where ω is the maximum weight of edges of the input graph. In Hausdorff metric, Alvarez et al. [2] proposed an algorithm in $O(\tau 1/\varepsilon^d \log(1/\varepsilon^d))$ time to approximate the weight of MST with error at most ε, where τ denotes the time needed to compute MST of points in a fixed metric L_p. Some randomized algorithms for approximating MST

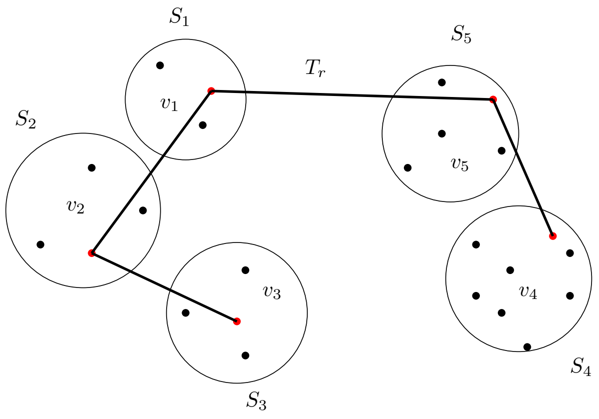

Figure 1: The red points are corresponding points of the super vertices. T_r is the partial solution in step r of the algorithm.

could be found in [4, 8]. Also, some results for approximating MST in the modern parallel models such as MapReduce could be found in [3].

2 Progressive algorithm for computing EMST

In this section, we aim to propose a progressive algorithm for computing EMST of a set P of n points in the plane. Before describing the algorithm, the partial solution and error function should be defined. Let $S = \{S_1, ...S_k\}$ be a partition of set P into the subsets $S_i \neq \emptyset$ such that $S_i \cap S_j = \emptyset$ for any $1 \leq i \neq j \leq k$. For each set $S_i \in S$, we assign a super vertex v_i. The super vertex v_i contains an arbitary point $p_i \in S_i$. Let E_r be a set of edges of a complete graph induced by $V = \{v_1, \ldots, v_k\}$. The weight of any edges $(v_i, v_j) \in E_r$ is defined by the Euclidean distance between p_i and p_j. Consider the graph $G_r = (V, E_r)$, the tree T_r in step r of the algorithm is an EMST of G_r and the weight of T_r denotes the partial solution (see Figure 1).

Let w_r be the weight of partial solution T_r in step r and w_{opt} be the weight of the exact EMST of P. The error function of our progressive algorithm is defined as follows:

$$err(T_r) = 1 - \frac{w_r}{w_{opt}}. \quad (1)$$

The Eq. (1) is a decreasing function. To prove this, we use the following lemma.

Lemma 1 *Let w_r be the weight of a partial solution T_r and w_{opt} be the weight of the exact EMST (T_{exact}) of P. We have $w_r \leq w_{opt}$.*

Proof. Let $V = \{v_1, \ldots, v_k\}$ and $E = \{e_1, \ldots, e_{k-1}\}$ be the set of vertices and edges of T_r respectively. Consider an edge $e_i = (v_t, v_l)$, and the corresponding sets S_t and S_l. We will show that p_t and p_l connect by a path in T_{exact} whose weight is greater or equal to e_i. Two cases may be occured. First, the points of S_t and S_l connect by an edge e_{min} in T_{exact} which is the minimum length edge between the points in S_t and S_l. Let H_1 and H_2 be two paths that connect endpoints of e_{min} to p_t and p_l. By triangle inequality, the weight of e_i is less than or equal to the weight of the path $H_1 \bigcup H_2 \bigcup e_{min}$ (see Figure 2).

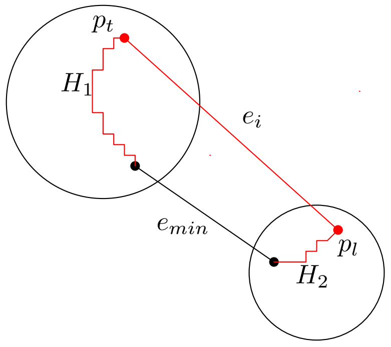

Figure 2: Illustration the first case of the proof of Lemma 1.

Second, there is a path H that connecting S_t and S_l through at least one subset S_k. In this case, by triangle inequality the weight of H is greater than e_i (see Figure 3). So we have $w_r < w_{opt}$. □

The idea of our progressive algorithm is the following: in any step r, we divide the points into 4^r disjoint sets and pick an arbitrary point from each set. Finally, EMST on the selected points is reported to the user as the partial solution. To divide the points into disjoint sets, we use $kd-tree$ approach to partition the plane by finding median vertical and horizontal lines with respect to the x and y coordinates.

So, in the first step of our progressive algorithm, we do as follows. Start by finding median points according to the x and y coordinates. Then draw vertical and horizontal lines passing through median points. These lines divide P into four subsets $S^1 = \{S_1, S_2, S_3, S_4\}$. For each $S_i \in S^1$ $i = 1, \ldots, 4$, assign a super vertex v_i that stores one point from S_i randomly. Construct complete graph induce by vertices v_1, v_2, v_3, v_4 and with edge weight correspond to the Euclidean distance between associate points of super vertices. Compute EMST of this graph (T_1) and report the weight of T_1 as the first partial solution.

In generic step r, we do as follows:

1. For each subset $S_i \in S^{r-1}$ do the following steps:
   (a) Divide S_i into four subsets with resepct to median lines according to x and y coordinates. Add these subsets to S^r.
   (b) Assign a super vertex for each new created subset and store a random point from corresponding subsets in it.

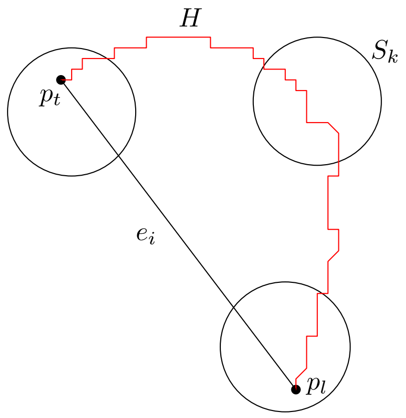

Figure 3: Illustration the second case of the proof of Lemma 1.

2. Let E_{new} be a set of edges that created by connecting each new vertex v to all vertices of T_{r-1}.
3. For each new super vertex v and corresponding edges E_{new}, T_r=*Update-EMST*$(T_{r-1}$,v,$E_{new})$.
4. Report the weight of T_r to the user.

To describe how we can update T_{r-1} to obtain T_r in step 3, the following subproblem could be defined. Let T be an EMST of a graph G that has already been computed and v be a new vertex. Add v to T and connect it to all vertices of T and call the new graph by T' . The problem is designing an efficient algorithm to compute an EMST of T'. To this end, we use the following lemma..

Lemma 2 *Let e be an edge that has minimum weight among all new edges incident to a new vertex v. Then EMST of T' must contain e.*

Proof. Let T'_m denotes the EMST of T'. Assume to the contrary that T'_m does not contain e. So T'_m must contain an edge e' with the weight greater than e. By adding e to T'_m, a cycle that contains both e and e' will be obtained. By deleting e' from T'_m, we get a spanning tree where its weight is less than T'_m. This contradicts to our assumption that T'_m is an EMST. □

The algorithm uses the above lemma as the main strategy. Among all new edges, the algorithm finds the edge with minimum weight and adds it to T. According to above lemma, this edge belongs to EMST of T'. By this addition, T' will be a spanning tree but not necessarily with the minimum weight. Now, each new edge is added one by one to T. Suppose (v_i, v_j) is added to T and $w((v_i, v_j))$ denotes its weight. This graph is a spanning tree, so there is a unique path between v_i and v_j. In this path, the algorithm finds an edge e_{max} with maximum weight. If $w(e_{max}) > w((v_i, v_j))$ then e_{max} is deleted from T and we obtain a spanning tree with smaller weight. Otherwise, (v_i, v_j) is deleted from T' and the current spanning tree is preserved. This process is executed for all new edges. Finally, EMST of T' is computed. To obtain the unique path between v_i and v_j and update T, we can traverse T similar to depth-first traversal of a tree. The Algorithm *Update-EMST* is what we need to do in step 3 of our progressive algorithm.

Algorithm 1: *Update-EMST*$(T$,v,$E)$

```
Output: EMST of $T$
1 Find an edge $(v, v_i)$ with minimum edges among $E$;
2 Add $(v, v_i)$ to $T$;
3 $e_{max} = (v, v_i)$;
4 $DFS(v_i, e_{max}, T, E)$;
5 return $T$;
```

To implement the depth-first traversal of T, we use an array $Tmax[1, \dots, n]$ such that $Tmax[i]$ denotes the edge with the maximum weight among the edges in the unique path between v and v_i. The vertex v is the first argument that $DFS(.)$ is invoked by it.

Algorithm 2: $DFS(v, e_{max}, T, E)$

```
1  Mark $v$ as a visited vertex;
2  for each vertex $v_i$ adjecent to $v$ do
3      if $v_i$ is not marked as visited vetex then
4          if $w(v, v_i) > w(e_{max})$ then
5              $Tmax[i] = (v, v_i)$;
6          else
7              $Tmax[i] = e_{max}$;
8          if a new edge $e_i$ exists in $E$ then
9              if $w(e_i) < Tmax[i]$ then
10                 Add $e_i$ to $T$;
11                 Delete $Tmax[i]$ from $T$;
12                 $Tmax[i] = e_i$;
13     Call $DFS(v_i, Tmax[i], T, E)$;
```

In the following lemma, we analyze the convergence function of our progressive algorithm.

Lemma 3 *Let T_r be a partial solution in step r. Then $err(T_r) \in O(1 - \frac{4^r}{n-1}\alpha^{-1})$, where α is the aspect ratio of the input points.*

Proof. It is clear that, in step r of the algorithm, 4^r super vertices are created. So the partial solution T_r must contain $4^r - 1$ edges. Consider the error function defined in Eq. (1). Let C and D be the minimum pairwise and the maximum pairwise distance of points in the input point set P respectively. Trivially, $w_r \geq$

$(4^r-1)C$ and $w_{opt} \leq D(n-1)$. So, the upper bound of the error value of T_r is:

$$err(T_r) \leq 1 - \frac{(4^r-1)C}{D(n-1)} \leq 1 - \frac{4^r}{n-1}\alpha^{-1}.$$

□

Remark 1: Our progressive algorithm generates the *monotone* partial soltuions. It means that, if T_r and T_{r+1} be two partial solutions in two consecutive steps r and $r+1$, then $w_r \leq w_{r+1}$. Also,

$$f_{conv}(r) \leq f_{conv}(r+1).$$

This property obtain directly from Lemma 1 and Lemma 3.

Remark 2: The idea of our progressive algorithm is approximating the weight of EMST by choosing a subset of P. This idea is similar to a framework called *Coresets* introduced by Agarwal et al. [10]. One possible method is using this approach to pick a small subset of P in each step but Alvarez et al. [2] showed that this framework does not work when we want to approximate the weight of EMST.

Theorem 4 *There exists a progressive algorithm for computing EMST of a set of n points in the plane with $O(\log_4 n)$ steps. The algorithm takes $O(n^2)$ time in the most time-consuming step and the convergence function of the algorithm is $O(1-\frac{4^r}{n-1}\alpha^{-1})$ according to the error function defined in Eq. (1).*

Proof. The convergence function is obtained from Lemma 3. Let $S^r = \{S_1, \dots, S_k\}$ be the decomposition of P in step r, where $k = 4^r$ and $|S_i|$ denotes the cardinality of S_i. By our approach for decomposition, each $S_i \in S^r$, is divided to four equal size subsets. So in step r, $|S_i| \leq \frac{n}{4^r}$. Therefore the number of steps is $O(\log n)$.

The lines 1 and 2 in step r of the progressive algorithm takes $\sum_{i=1}^{k} |S_i| = O(n)$ time, since $S_i \cap S_j = \emptyset$ holds for all $1 \leq i \neq j \leq k$.

For line 3, we know that T_{r-1} has $4^{r-1}-1$ edges. By adding a new vertex v to T_{r-1}, we have 4^{r-1} new edges. So, finding minimum edge takes $O(4^{r-1})$ time. To update T_{r-1} when v is added, we need to depth-first traversal of T_{r-1} that takes $O(4^{r-1})$ time. Therefore, updating T_{r-1} for each new added vertex takes $O(4^{r-1})$ time in total. We have $(4^r - 4^{r-1}) = 3 \times 4^{r-1}$ new vertices in step r. The total time of updating T_{r-1} is:

$$3 \times 4^{r-1} \times O(4^{r-1}) \in O(4^{2r-2}).$$

Since $r = O(\log_4 n)$, so step r never takes more than $O(n^2)$ time. □

3 Conclusion

In this paper, a progressive algorithm is proposed for computing the Euclidean minimum spanning tree of a set of n points in the plane with $O(\log_4 n)$ steps and the most time-consuming step takes $O(n^2)$ time. The upper bound on the error value of the generated partial solution in step r is $O(1-\frac{4^r}{n-1}\alpha^{-1})$, where α is the aspect ratio of points. One interesting future work is designing a progressive algorithm with convergence function that only depends on the size of input especially when α is unbounded. Developing the progressive algorithm for other problems with a massive input data are also interesting.

References

[1] S. P. A. Alewijnse, T. M. Bagautdinov, M. de Berg, Q. W. Bouts, A. P. ten Brink, K. Buchin, and M. A. Westenberg. Progressive geometric algorithms. *Journal of Computational Geometry*, 6(2):72–92, 2015.

[2] V. Alvarez and R. Seidel. Approximating the minimum weight spanning tree of a set of points in the hausdorff metric. *Computational Geometry*, 43(2):94–98, 2010.

[3] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 574–583. ACM, 2014.

[4] P. Berenbrink, B. Krayenhoff, and F. Mallmann-Trenn. Estimating the number of connected components in sublinear time. *Information Processing Letters*, 114(11):639–642, 2014.

[5] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on computing*, 34(6):1370–1379, 2005.

[6] K. L. Clarkson. Fast expected-time and approximation algorithms for geometric minimum spanning trees. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 342–348. ACM, 1984.

[7] A. Czumaj, F. Ergün, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler. Approximating the weight of the euclidean minimum spanning tree in sublinear time. *SIAM Journal on Computing*, 35(1):91–109, 2005.

[8] A. Gupta and J. Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16(1):3–20, 2011.

[9] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Efficient and progressive group steiner tree search. In *Proceedings of the 2016 International Conference on Management of Data*, pages 91–106. ACM, 2016.

[10] A. Mesrikhani and M. Farshi. Progressive sorting in the external memory model. In *48th Annual Iranian Mathematics Conference (AIMC48)*, August 2017.