Improved Algorithms for the Point-Set Embeddability problem for Plane 3-Trees

Tanaeem M Moosa and M. Sohel Rahman

[1] Department of CSE, BUET, Dhaka-1000, Bangladesh
http://www.buet.ac.bd/cse
[2] {tanaeem,msrahman}@cse.buet.ac.bd

Abstract. In the point set embeddability problem, we are given a plane graph G with n vertices and a point set S with n points. Now the goal is to answer the question whether there exists a straight-line drawing of G such that each vertex is represented as a distinct point of S as well as to provide an embedding if one does exist. Recently, in [15], a complete characterization for this problem on a special class of graphs known as the plane 3-trees was presented along with an efficient algorithm to solve the problem. In this paper, we use the same characterization to devise an improved algorithm for the same problem. Much of the efficiency we achieve comes from clever uses of the triangular range search technique. We also study a generalized version of the problem and present improved algorithms for this version of the problem as well.

1 Introduction

A *planar graph* is a graph that can be *embedded* in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints. A planar graph already drawn in the plane without edge intersections is called a *plane graph* or *planar embedding* of the graph. A straight-line drawing Γ of a plane graph G is a graph embedding in which each vertex is drawn as a point and each edge is drawn as straight line segments (as opposed to curves, etc.).

Given a plane graph G with n vertices and a set S of n points in the plane, a point-set embedding of G on S is a straight-line drawing of G such that each vertex is represented as a distinct point of S. The problem of computing a point-set embedding of a graph, also referred to as the point-set embeddability problem in the literature, has been extensively studied both when the mapping of the vertices to the points is chosen by the drawing algorithm and when it is partially or completely given as part of the input. There exists a number of results of the point-set embeddability problem on different graph classes in the literature [4, 10, 12, 17]. A number of variants of the original problem have also been studied in the literature. For example in [1, 8], a variant of the point-set embeddability problem has been studied, where the vertex set of the given graph and the given set of points are divided into a number of partitions and a particular vertex subset is to be mapped to a particular point subset. Other variants have also been studied with great interest [13, 9].

Very recently, Nishat et al. [15] studied the point set embeddability problem on a class of graphs known as the *plane 3-tree*. Plane 3-trees belong to an interesting class of graphs and recently a number of different drawing algorithms have been presented in the literature on plane 3-trees [2, 14, 15]. In this paper, we follow up the work of [15] and improve upon their result from an algorithmic point of view. In [15], Nishat et al. presented an $O(n^2 \log n)$ time algorithm that can decide whether a plane 3-tree G of n vertices admits a point-set embedding on a given set of n points or not and compute a point-set embedding of G if such an embedding exists. In this paper, we show how to improve the running time of the above algorithm. In particular, we take their algorithmic ideas as the building block of our algorithm and with some non trivial modifications we achieve a running time of $O(n^{4/3+\epsilon} \log n)$. The efficiency of our algorithm comes mainly from clever uses of triangular range search and counting queries [18, 7, 6] and bounding the number of such queries. Furthermore, we study a generalized version of the Point-Set Embeddability problem where the point set S has more points than the number of vertices of the input plane 3-tree, i.e., $|S| = k > n$. For this version of the problem, an $O(nk^8)$ time algorithm was presented in [15]. We present an improved algorithm running in $O(nk^4)$ time.

The rest of this paper is organized as follows. Section 2 presents some definitions and preliminary results. Section 3 presents a brief review of the algorithm presented in [15]. In Section 4 we present our main result. Section 5 briefly discusses about the generalized version of the problem and we briefly conclude in Section 6.

2 Preliminaries

In this section we present some preliminary notations, definitions and results that we use in our paper. We follow mainly the definitions and notations of [16]. We start with a formal definition of the *straight-line drawing*.

Definition 1 (Straight-Line Drawing) *Given a plane graph G, a straight-line drawing $\Gamma(G)$ of G is a drawing of G where vertices are drawn as points and edges are drawn as connecting lines.*

The problem we handle in this paper is formally defined as follows.

Problem 1 (Point-Set Embeddability) *Let G be a plane graph of n vertices and S be a set of n points on plane. The point-set embeddability problem wants to find a straight-line drawing of G such that the vertices of G are mapped to the points of S.*

Finding a point-set embedding for an arbitrary plane graph is proved to be NP-Complete [5], even for some restricted subclasses. On the other hand, polynomial time algorithm exists for finding point-set embedding for outerplanar graphs or trees [11, 4]. An interesting research direction in the literature is to investigate this problem on various other restricted graph classes. One such interesting graph class, known as the plane 3-tree, is formally defined below.

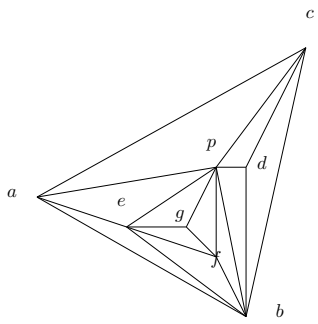


Fig. 1. A plane 3-tree of 17 vertices

Definition 1 (Plane 3-Tree). *A plane 3-tree is a triangulated plane graph $G = (V, E)$ with n vertices such that either $n = 3$, or there exists a vertex x such that the graph induced by $V - \{x\}$ is also a plane 3-tree.*

Figure 1 presents a plane 3-tree with 17 vertices. As has been mentioned above, the very recent work of Nishat et al. [15] proved that finding point-set embedding is polynomially solvable if the input is restricted to a Plane 3-Tree. Since a plane 3-tree is triangulated, its outer face has only 3 vertices, known as the *outer vertices*. The following two interesting properties of a plane 3-tree with $n > 3$ will be required later.

Proposition 1 ([3]) *Let G be a plane 3-tree with $n > 3$ vertices. Then, there is a node x with degree 3 whose deletion will give a plane 3-tree of $n-1$ vertices.*

Proposition 2 ([3]) *Let G be a plane 3-tree with $n > 3$ vertices. Then, there exists exactly 1 vertex (say, p) that is a common neighbor of all 3 outer vertices.*

For a plane 3-tree G, the vertex p (as defined in Proposition 2) is referred to as the *representative vertex* of G. For a plane graph G, and a cycle C in it, we use $G(C)$ to denote the subgraph of G inside C (including C). In what follows, if a cycle C is a triangle involving the vertices x, y and z, we will often use $\triangle xyz$ and $G(\triangle xyz)$ to denote C and $G(C)$. The following interesting lemma was recently proved in [15] and will be useful later in this paper.

Lemma 1 ([15]). *Let G be a plane 3-tree of $n > 3$ vertices and C be any triangle of G. Then, the subgraph $G(C)$ is a plane 3-tree.*

We now define an interesting structure related to a plane 3-tree, known as the *representative tree*.

Definition 2 (Representative Tree). *Let G be a plane 3-tree with n vertices with outer vertices a, b and c and representative vertex p (if $n > 3$). The representative tree T of G is an ordered tree defined as follows:*

- *If $n = 3$, then T is an single vertex.*

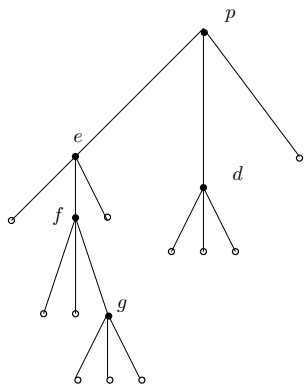


Fig. 2. The representative tree of the plane 3-tree of Figure 1

– *Otherwise, the root of T is p and its subtrees are the representative trees of $G(\triangle apb)$, $G(\triangle bpc)$ and $G(\triangle cpa)$ in that order.*

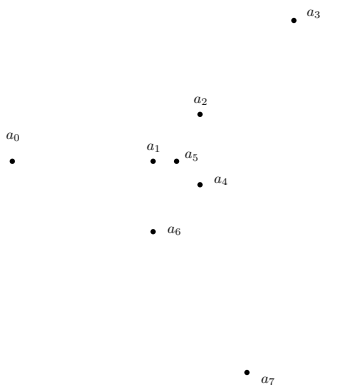
The representative tree of the plane 3-tree of Figure 1 is presented in Figure 2. Note that, the representative tree T has $n' = n - 3$ internal nodes, each internal node having degree 3. Also, note that the outer vertices of G are named as a, b and c respectively in counter-clockwise order around p. Therefore, the representative tree T of a plane 3-tree G is unique as per Definition 2. Now consider a plane 3-tree G and its representative tree T. Assume that G' is a subgraph of G and T' is a subtree of T. Then, G' is referred to as the *corresponding subgraph* of T' if and only if T' is the representative tree of G'. There is an $O(n)$ time algorithm to construct the representative tree from a given plane graph [15].

Given a set of points S, we use the symbol $P_S(\triangle xyz)$ to denote the set of points that are inside the triangle $\triangle xyz$. We use the symbol $N_S(\triangle xyz)$ to denote size of the set $P_S(\triangle xyz)$. We will extensively use the triangular range search and counting queries in our algorithm. Below we formally define these two types of queries.

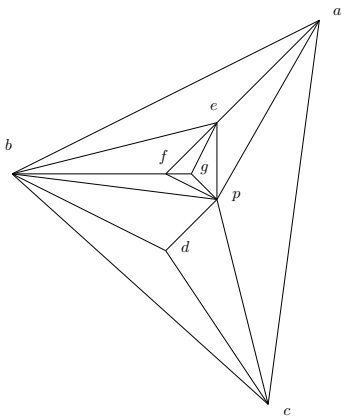
Problem 2 (Triangular Range Search) *Given a set S of points that can be preprocessed, we have to answer queries of the form $SetQuery(S, \triangle abc)$, where the query returns $P_S(\triangle abc)$.*

Problem 3 (Triangular Range Counting) *Given a set S of points that can be preprocessed, we have to answer queries of the form $CountQuery(S, \triangle abc)$, where the query returns $N_S(\triangle abc)$.*

In what follows, we will follow the following convention: If an algorithm has preprocessing time $f(n)$ and query time $g(n)$, we will say its overall running time is $\langle f(n), g(n) \rangle$. We conclude this section with an example illustrating the point set embedding of a plane 3-tree.



(a) An example of the point set S.



(b) An embedding of the plane 3-tree of Figure 1 on the point set of Figure 3(a)

Fig. 3. An example of point set embedding of a plane 3-tree.

Example 1 *In Figure 3(a) we present an example of the point set S having $n = 17$ points. Then, in Figure 3(b), an embedding of the plane 3-tree of Figure 1 is illustrated.*

3 Algorithm of [15]

In this section, we briefly describe the quadratic time algorithm of [15]. To simplify the description, we first assume that the points of S are in general positions, i.e., no three points of S are collinear.

Lemma 2 ([15]). *Let G be a plane 3-tree of n vertices and S be a set of n points. If G admits a point-set embedding on S, then the convex hull of S contains exactly three points in S.*

Lemma 3 ([15]). *Let G be a plane 3-tree of n vertices with a, b and c being the three outer vertices of G, and let p be the representative vertex of G. Let S be a set of n points such that the convex hull of S contains exactly three points. Assume that G has a point-set embedding $\Gamma(G)$ on S for a given mapping of a, b and c to the three points of the convex-hull of S. Then p has a unique valid mapping.*

The algorithm of [15] performs the following steps to find a valid embedding of G in a given point-set S if one exists.

Step: 1 Find the convex hull of the given points. If the convex hull does not have exactly 3 points, return the message that no embedding exists.

Step: 2 For each of the possible 6 mappings of the outer vertices of G to the three points of the convex hull, perform Steps 3 and 4 (recursively).

Step: 3 Assume that at the beginning of this step, we are considering the representative (sub)tree T' and the corresponding graph is G' (obviously a subgraph of G). Let the three outer vertices of G' are a', b' and c' and the representative vertex of it is p'. Note that, initially, $G' = G$, $T' = T$ and the outer vertices and the representative vertex are a, b, c and p respectively. Assume that the number of internal nodes in T' is n'. Note that, number of vertices in the corresponding graph G' is $n'+3$. If $n' = 0$ then embedding is trivially possible and this step returns immediately terminating the recursion. Otherwise, the following step is executed to check whether an embedding is indeed possible.

Step: 4 Let the root of T' be r. Also let the three children of r be r_1, r_2 and r_3 and the number of internal nodes in the subtrees rooted at r_1, r_2 and r_3 be n'_1, n'_2 and n'_3 respectively. Note that $n' = n'_1 + n'_2 + n'_3 + 1$. Let the three outer vertices a', b' and c' of G' are mapped to points x, y and z of S. Now, we find a point u in S such that $N_S(\triangle xuy) = n'_1$, $N_S(\triangle yuz) = n'_2$, and $N_S(\triangle zux) = n'_3$. By Lemma 3, u is unique if it exists. To find u, all the points of S lying within the triangle $\triangle xyz$ are checked. If u can be found, then p is mapped to u and Steps 3 and 4 are executed recursively for all three subtrees of T'; otherwise no embedding is possible.

In what follows, we will refer to this algorithm as the NMR Algorithm. Naive implementation of NMR algorithm runs in $O(n^3)$ time [15]. By sorting all points of S according the polar angle with respect to each point of S and employing some non-trivial observations, this algorithm can be made to run in $O(n^2)$ time [15]. Note that, the $O(n^2)$ algorithm assumes that the points of S are at general positions. If this assumption is removed, NMR algorithm runs in $O(n^2 \log n)$ time.

4 Our Result

In this section, we modify the algorithm of [15] described in Section 3 and achieve a better running time. For the ease of exposition, we assume for the time being that triangular range search has running time $\langle f(|S|), g(|S|) + \ell \rangle$ and triangular range counting has running time $\langle f(|S|), g(|S|) \rangle$, where S is the input set and ℓ is the output size for triangular range search. We will finally use the actual running times during the analsysis of the algorithm. We first present our modified algorithm below followed by a detailed running time analysis.

Step 1: Find the convex hull of the points of S. By Lemma 2 the convex hull should have 3 points, otherwise no embedding exists.

Step 2: Preprocess the points of S for triangular range search and triangular range counting.

Step 3: For each of the possible 6 mappings of the outer vertices of G to the three points of the convex hull, perform Steps 4 to 6 (recursively).

Step 4: We take the same assumptions as we took at Step 3 of the NMR algorithm. Now, if $n' = 0$ then embedding is trivially possible and this step returns immediately terminating the recursion. Otherwise, the following step is executed to check whether an embedding is indeed possible.

Step 5: Now we want to find a point u such that $N_S(\triangle xuy) = n'_1$, $N_S(\triangle yuz) = n'_2$, and $N_S(\triangle zux) = n'_3$. Recall that, by lemma 3 such a point is unique if it exists. Now, without loss of generality we can assume that $n'_2 \leq \min(n'_1, n'_3)$. In order to find u, we first find points v_1 and v_2 on the line yz such that $N_S(\triangle xv_1y) = n'_1$ and $N_S(\triangle xv_2z) = n'_3$. Note carefully that, in line yz, v_1 appears closer to y than v_2; otherwise there will not be n' points inside the triangle $\triangle xyz$. We will use a binary search and triangular range counting queries to find v_1, v_2 as follows. We first choose the mid point w of the line BC. Then we compute $N_S(\triangle xwy)$ using a triangular range counting query. If $N_S(\triangle xwy) = n'_1$ we are done and we assign $v_1 = w$. Otherwise, if $N_S(\triangle xwy) > n'_1$ ($N_S(\triangle xwy) < n'_1$), then we choose the mid point w' of the line Bw (wC). Then we perform similar checks on $\triangle xw'y$. The point v_2 can also found similarly. Clearly, there always exist such points and steps of binary search is bounded by $O(\log N)$, where N is the maximum absolute value of a point of S in any coordinate.

Step 6: With points v_1 and v_2 at our disposal, we now try to find point u. Note that the point u cannot be inside either $\triangle xv_1y$ or $\triangle xv_2z$. This

is because if u is in $\triangle xv_1y$ then $N_S(\triangle xuy) < N_S(\triangle xv_1y) = n'_1$ implying $N_S(\triangle xuy) < n'_1$, a contradiction. A similar argument is possible for $\triangle xv_2z$. So, we must have $u \in P_S(\triangle xv_1v_2)$. Also note that $N_S(\triangle xv_1v_2) = N_S(\triangle xyz) - N_S(\triangle xv_1y) - N_S(\triangle xv_2z) = n' - n'_1 - n'_3 = n'_2 + 1$. Using triangular range search we now find the points $P_S(\triangle xv_1v_2)$. To find u, we now simply check whether any of these points satisfies the requirement $N_S(\triangle xuy) = n'_1$, $N_S(\triangle yuz) = n'_2$, and $N_S(\triangle zux) = n'_3$. If no such point exists, then we return stating that it will be impossible to embed the graph on the points. Otherwise we find a point u, which is mapped to vertex p. Now Steps 4 to 6 are recursively executed for all three subtrees.

4.1 Analysis

Now we analyze our modified algorithm presented above. Step 1 is executed once and can be done in $O(n \log n)$ time. Step 2 is executed once and can be done in $f(|S|)$ time. Steps 4 to 6 are executed recursively. Step 4 basically gives us the terminating condition of the recursion. We focus on Step 5 and Step 6 separately below.

In Step 5, we find out the two points v_1 and v_2 using binary search and triangular range counting queries. Time required for this step is $O(g(|S|) \log N)$. Note carefully that both the parameters $|S|$ and N are constant in terms of recursion. Also, it is easy to see that, overall, Step 5 is executed once for each node in T. Hence, the overall running time of this step is $O(g(|S|)n \log N)$.

Now we focus on Step 6. Time required for triangular range search in Step 6 is $O(g(|S|) + n'_2)$. In this step we also need $O(n'_2)$ triangular range counting queries which add up to $O(g(|S|)n'_2)$ time. Recall that, $n'_2 = \min(n'_1, n'_3)$, i.e., n'_2 is the number internal nodes of the subtree having the least number of internal nodes. Hence, we have $n'_2 \leq n'/3$. Now, the overall running time of Step 6 can be expressed using the following recursive formula: $T(n') = T(n'_1) + T(n'_2) + T(n'_3) + n'_2 g(|S|))$, where $n'_2 \leq \min(n'_1, n'_3)$. Now we have the following theorem:

Theorem 1. *The overall running time of Step 6 is* $O(g(|S|)\ n \log n)$.

Proof. First, assume that $T(n) \leq c(n \log n)g(|S|), c \geq 1$. Then we have,

$$
\begin{aligned}
T(n') &= T(n'_1) + T(n'_2) + T(n'_3) + n'_2 g(|S|) \\
&\leq c(n'_1 \log n'_1)g(|S|) + c(n'_2 \log n'_2)g(|S|) + c(n'_3 \log n'_3)g(|S|) + n'_2 g(|S|) \\
&< c(n'_1 \log n')g(|S|) + c(n'_2 \log \frac{n'}{2})g(|S|) + c(n'_3 \log n')g(|S|) + c \times n'_2 g(|S|) \\
&= c(n'_1 \log n')g(|S|) + c(n'_2(-1 + \log n'))g(|S|) + c(n'_3 \log n')g(|S|) + cn'_2 g(|S|) \\
&= cg(|S|)(-n'_2 + (n'_1 + n'_2 + n'_3) \log n' + n'_2) \\
&= cg(|S|)n' \log n'
\end{aligned}
$$

This completes the proof. □

Based on the above discussion, total time required for this algorithm is $O(n \log n + f(|S|) + ng(|S|) \log N + ng(|S|) \log n) = O(f(|S|) + n\ g(|S|)(\log n + \log N))$. Now we are ready to replace $f(|S|)$ and $g(|S|)$ with some concrete values. To the best of our knowledge, the best result of triangular range search and counting queries is due to Chazelle et al. [7]. In particular, Chazelle et al. proposed a solution for the triangular range search queries in [7] with time complexity $\langle O(m^{1+\epsilon}), O(n^{1+\epsilon}/m^{1/2})\rangle$, where $n < m < n^2$. Using this result the running time of our algorithm becomes $O(m^{1+\epsilon} + (\log n + \log N)n^{2+\epsilon}/m^{1/2})$, which reduces to $O(n^{4/3+\epsilon} + (\log n + \log N)n^{4/3+\epsilon})$ if we choose $m = n^{4/3}$.

Finally, we can safely ignore the $\log N$ component from our running time as follows. Firstly, the $\log N$ component becomes significant only when N is doubly exponential in n or larger, which is not really practical. Secondly, while we talk about the theoretical running time of algorithms, we often ignore the inherent $O(\log N)$ terms assuming that two (large) numbers can be compared in $O(1)$ time. For example, in comparison model, since sorting n (large) numbers having maximum value N requires $\Theta(n \log n)$ comparisons we usually say that sorting requires $\Theta(n \log n)$ time. Essentially, here, we ignore the fact that comparing two numbers actually requires $\Omega(\log N)$ time. Notably, the algorithm of [15] also has an hidden $O \log N$ term since it requires $O(n^2)$ comparisons each of which actually requires $O(\log N)$ time. One final note is that for instances where the $\log N$ term does have significant effect, we can in fact get rid of the term using standard techniques to transform a counting algorithm into a ranking algorithm at the cost of a $\log n$ time increase in the running time. Similar techniques are also applicable for the algorithm of [15]. So, we have the following theorem.

Theorem 2. *The point set Embeddability problem can be solved in $O(n^{4/3+\epsilon} \log n)$ time if the input graph is a plane 3-tree and S does not contain any three points that are collinear.*

4.2 For points not in General positions

So far we have assumed that the points of S are in general positions, i.e., no three points in S are collinear. We now discuss how to get around this assumption. Note that, the algorithm of Nishat et al [15] shows improved performance of $O(n^2)$ when the points of S are in general positions. Now, if we remove our assumption, then we may have more than two points that are collinear. In this case, the only only modification needed in our algorithm is in Step 5. Now, the problem is that the two points v_1 and v_2 could not be found readily as before. More specifically, even if Step 5 returns that v_1 and v_2 do not exist, still u may exist. Now note that, in Step 5, we want to find out v_1 and v_2 to ensure that $N_S(\triangle xv_1v_2) = n'_2 + 1$, where $n'_2 = \min(n'_1, n'_3)$, i.e., $n'_2 \leq n'/3$. Since, we have to check each points of $P_S(\triangle xv_1v_2)$ (to find u), the above bound of $n'_2 \leq n'/3$ provides us with the required efficiency in our algorithm.

To achieve the same efficiency, we now slightly modify Step 5. Suppose we are finding v_1 (v_2). We now try to find v_1 (v_2) such that the $N_S(\triangle xv_1y) > n'_1$ ($N_S(\triangle xv_2z) > n'_3$) and v_1 (v_2) is as near as possible to B (C) on the line BC. Let

us assume that we need $\mathcal{I}$ iterations now to find v_1 (v_2). We have the following bound for $\mathcal{I}$.

Lemma 4. *$\mathcal{I}$ is bounded by $O(\log N)$*

Proof. There may not be any point candidate of v_1 (v_2) which has integer coordinates. But as x can be intersection of two lines, each of which goes through two points of S, there may exists a point candidate of v_1 having denominator less than N^2 or there is none. Either way, to find such a point or to be sure no such point exists we only need precision less than $1/N^2$. Therefore, $O(\log N)$ iterations are sufficient. □

Again, the argument presented at the end of Section 4.1 about the component $\log N$ applies here as well. Therefor, the result of Theorem 2 holds even the points of S are not in general positions. So, we restate our stronger and more general result as follows.

Theorem 3. *The point set Embeddability problem can be solved in $O(n^{4/3+\epsilon} \log n)$ time if the input graph is a plane 3-tree.*

5 Generalized Version

A generalized version of the Point Set Embeddability problem is also of interest in the graph drawing research community, where S has more points than the number of vertices of the input graph G. More formally, the generalized point set embeddability problem is defined as follows.

Problem 4 (Generalized Point-Set Embeddability) *Let G be a plane graph of n vertices and S is a set of k points on plane such that $k > n$. The generalized point-set embeddability problem wants to find a straight-line drawing of G such that vertices of G are mapped to some n points of S.*

In this section, we extend our study to solve the Generalized Point-set Embeddability problem for plane 3-trees. This version of the problem was also handled in [15] for plane 3-trees and they presented an algorithm for the problem that runs in $O(nk^8)$ time. Our target again is to improve upon their algorithm. We show how to improve the result to $O(nk^4)$.

We will use dynamic programming (DP) for this purpose. For DP, we define our (sub)problem to be $Embed(r', a', b', c')$ where r' is the root of the (sub)tree T' and a',b' and c' are points in S. Now, $Embed(r', a', b', c')$ returns true if and only if it is possible to embed the corresponding subgraph G' of the subtree T' rooted at r' such that its three outer vertices are mapped to the points a', b' and c'. Now we can start building the DP matrix by computing $Embed(r', a', b', c')$ for the smaller subtrees to see whether the corresponding subgraphs can be embedded for a combination of 3 points of S as outer vertices of the corresponding subgraphs. In the end, the goal is to check whether $Embed(r, a, b, c)$ returns true for any particular points $a, b, c \in S$, where r is the root of the representative

tree T of the input plane 3-tree G. Clearly, if there exists $a, b, c \in S$ such that $Embed(r, a, b, c)$ is true, then G is embeddable in S. We now have the following theorem.

Theorem 4. *The Generalized Point-Set Embeddability problem can be solved in $O(n \times k^4)$ time.*

Proof. If r' is a leaf, then $Embed(r', a', b', c')$ is trivially true for any $a', b', c' \in S$. Now consider the calculation of $Embed(r', a', b', c')$ when r' is not a leaf. Then, assume that the children of r' are r'_1, r'_2 and r'_3 in that order. Then $Embed(r', a', b', c')$ is true if and only if we can find a point $u \in P_S(\triangle a'b'c')$ such that $Embed(r'_1, a', b', u)$, $Embed(r'_2, u, b', c')$ and $Embed(r'_3, a', u, c')$ are true; otherwise $Embed(r', a', b', c')$ will be false. Clearly, the time required to calculate $Embed(r', a', b', c')$ is $O(g(|S|) + N_S(\triangle a'b'c'))$. Therefore, in the worst case the time required to calculate an entry of the DP matrix is $O(g(|S|) + k)$. Now, it is easy to realize that there are in total nk^3 entries in the DP matrix. Note that to be able to compute $P_S(\triangle a'b'c')$ we need to spend $O(f(|S|))$ time for a one-time preprocessing. Hence, the total running time is $O(f(|S|) + nk^3 \times (g(|S|) + k)) = O(f(|S|) + nk^3 g(|S|) + nk^4)$. Using the $\langle O(m^{1+\epsilon}), O(n^{1+\epsilon}/m^{1/2})\rangle$ result of Chazelle et al. [7], the running time becomes $O(O(m^{1+\epsilon}) + nk^3 \times n^{1+\epsilon}/m^{1/2} + nk^4)$. Since, $n < m < n^2$, this running time is $O(nk^4)$ in the worst case. □

6 Conclusion

In this paper, we have followed up the work of [15] and presented an algorithm to solve the point-set embeddability problem in $O(n^{4/3+\epsilon} \log n)$ time. This improves the recent $O(n^2 \log n)$ time result of [15]. Whether this algorithm can be improved further is an interesting open problem. In fact, an $o(n^{4/3})$ algorithm could be an interesting avenue to explore, which however, does not seem to be likely with our current technique. Since there are $\Omega(n)$ nodes in the tree, any solution that uses triangular range search to check validity at least once for each node in the tree would require $\Omega(n)$ calls to triangular range query. Lower bound for triangular range search is shown to be $\langle \Omega(m), \Omega(n/m^{1/2})\rangle$ [6], which suggests an $\Omega(n^{4/3})$ lower bound for our algorithm using triangular range search.

We have also studied a generalized version of the point-set embeddability problem where the point set S has more than n points. For this version of the problem we have presented an algorithm that runs in $O(nk^4)$ time, where $k = |S|$. Nishat et al. [15] also handled this version of the problem and presented an $O(nk^8)$ time algorithm. It would be interesting to see whether further improvements in this case are possible. Also, future research may be directed towards solving these problems for various other classes of graphs.

References

1. M. Badent, E. D. Giacomo, and G. Liotta. Drawing colored graphs on colored points. In F. K. H. A. Dehne, J.-R. Sack, and N. Zeh, editors, *WADS*, volume 4619 of *Lecture Notes in Computer Science*, pages 102–113. Springer, 2007.

2. T. Biedl and L. R. Velzquez. Drawing planar 3-trees with given face-areas. In D. Eppstein and E. Gansner, editors, *Graph Drawing*, volume 5849 of *Lecture Notes in Computer Science*, pages 316–322. Springer Berlin / Heidelberg, 2010.
3. T. C. Biedl and L. E. R. Velázquez. Drawing planar 3-trees with given face-areas. In D. Eppstein and E. R. Gansner, editors, *Graph Drawing*, volume 5849 of *Lecture Notes in Computer Science*, pages 316–322. Springer, 2009.
4. P. Bose. On embedding an outer-planar graph in a point set. *Comput. Geom.*, 23(3):303–312, 2002.
5. S. Cabello. Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. *J. Graph Algorithms Appl*, 10(2):353–363, 2006.
6. B. Chazelle. Lower bounds for off-line range searching. *Discrete & Computational Geometry*, 17(1):53–65, 1997.
7. B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Algorithmica*, 8:407–429, 1992.
8. E. D. Giacomo, W. Didimo, G. Liotta, H. Meijer, F. Trotta, and S. K. Wismath. k-colored point-set embeddability of outerplanar graphs. *J. Graph Algorithms Appl.*, 12(1):29–49, 2008.
9. E. D. Giacomo, G. Liotta, and F. Trotta. On embedding a graph on two sets of points. *Int. J. Found. Comput. Sci.*, 17(5):1071–1094, 2006.
10. Y. Ikebe, M. A. Perles, A. Tamura, and S. Tokunaga. The rooted tree embedding problem into points in the plane. *Discrete & Computational Geometry*, 11:51–63, 1994.
11. Y. Ikebe, M. A. Perles, A. Tamura, and S. Tokunaga. The rooted tree embedding problem into points in the plane. *Discrete & Computational Geometry*, 11:51–63, 1994.
12. A. Kaneko and M. Kano. Straight line embeddings of rooted star forests in the plane. *Discrete Applied Mathematics*, 101(1-3):167–175, 2000.
13. A. Kaneko and M. Kano. Semi-balanced partitions of two sets of points and embeddings of rooted forests. *Int. J. Comput. Geometry Appl.*, 15(3):229–238, 2005.
14. D. Mondal, R. I. Nishat, M. Rahman, and J. Alam. Minimum-area drawings of plane 3-trees. In *Proceedings of the 22nd Canadian Conference on Computational Geometry (CCCG2010)*, pages 191–194, 2010.
15. R. I. Nishat, D. Mondal, and M. S. Rahman. Point-set embeddings of plane 3-trees. In *Graph Drawing*, page To Appear, 2010.
16. T. Nishizeki and M. S. Rahman. *Planar graph drawing*. World Scientific Pub Co. Inc., 2004.
17. J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics*, 17(4):717–728, 2001.
18. M. S. Paterson and F. F. Yao. Point retrieval for polygons. *Journal of Algorithms*, 7(3):441 – 447, 1986.