On Embedding an Outer-Planar Graph in a Point Set

Prosenjit Bose[1,2]

[1] School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, Canada, K1S 5B6. E-mail: jit@scs.carleton.ca.

[2] Research supported by NSERC Grant OGP0183877 and a FIR Grant.

Abstract. Given an n-vertex outer-planar graph G and a set P of n points in the plane, we present an $O(n \log^3 n)$ time and $O(n)$ space algorithm to compute a straight-line embedding of G in P, improving upon the algorithm in [GMPP91, CU96] that requires $O(n^2)$ time. Our algorithm is near-optimal as there is an $\Omega(n \log n)$ lower bound for the problem [BMS95]. We present a simpler $O(nd)$ time and $O(n)$ space algorithm to compute a straight-line embedding of G in P where $\log n \leq d \leq 2n$ is the length of the longest vertex disjoint path in the dual of G. Therefore, the time complexity of the simpler algorithm varies between $O(n \log n)$ and $O(n^2)$ depending on the value of d. More efficient algorithms are presented for certain restricted cases. If the dual of G is a path, then an optimal $\Theta(n \log n)$ time algorithm is presented. If the given point set is in convex position then we show that $O(n)$ time suffices.

1 Introduction

The problem of deciding whether a certain combinatorial structure can be embedded in the plane, as well as computing an embedding of that structure has been a recurrent theme in many fields but particularly in graph drawing. From a graph drawing perspective (see [DETT94] for a survey of graph drawing), the traditional questions ask whether a graph can be embedded in the plane such that some criterion is satisfied e.g., that the area of the resulting embedding is small [CP95, KLTT93], that the symmetry present in the graph is revealed in the embedding [MA88], or that the graph is isomorphic to a proximity graph [EW94, MS91, BDLL95, BLL96] of the points in which the vertices are embedded.

The embedding problem that we address has a slightly different perspective: both the point set and the graph are given as input. We want to determine if the input graph can be straight-line embedded in the input point set. We say that an n-node graph $G = (V, E)$ can be *straight-line embedded* onto a set of n points P, if there exists a one-to-one mapping $\phi\colon V \rightarrow P$ from the nodes of G to the points of P such that edges of G intersect only at nodes. That is, edges $(\phi(u_1), \phi(v_1)) \cap (\phi(u_2), \phi(v_2)) = \emptyset$, for all $u_1 v_1 \neq u_2 v_2 \in E$.

The definition of a straight-line embedding implies that G must be a planar graph in order for a straight-line embedding of G onto P to exist. However, even if G is planar, there exist point sets that do not admit a straight-line embedding of G. See Figure 1 for such an example. This raises an interesting open question: Given a planar graph G and a point set P, can G be straight-line embedded in P? We believe that this problem is *NP*-complete. Although the question when G is a planar graph remains unanswered, progress has been made when G is restricted to a subclass of planar graphs.

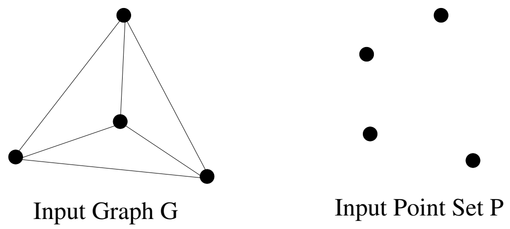

Fig. 1. A graph that cannot be straight-line embedded in a point set.

When the class of graphs is restricted to trees, Perles at the 1990 DIMACS workshop on arrangements posed the following question: Given n points P in general position and an n-node tree T rooted at node ν, can T be straight-line embedded in P with ν at a specified point $p \in P$? Perles showed that this was always possible if p was on the *convex hull of* P, which is the smallest convex set containing the points P. Pach and Törőcsik [PT93] showed that it could if p was not the *deepest point* of P, obtained by repeatedly discarding points on the convex hull. Subsequently, Ikebe et. al [IPTT94] showed that there was always such an embedding using a quadratic time algorithm. In fact, all three algorithms use quadratic time. Finally, Bose et. al [BMS95] proved an $\Omega(n \log n)$ lower bound for the problem and provided a matching $O(n \log n)$ time embedding algorithm.

With the embedding problem being resolved when the input graphs are restricted to trees and unresolved when the input graphs are planar, a natural question to ask is what is the largest subclass of planar graphs that admits a straight-line embedding on any point set. Gritzmann et al. [GMPP91] first showed that the class of outer-planar graphs is the largest class of graphs that admits an embedding in any point set and provided an embedding algorithm that runs in $O(n^2)$ time (Castañeda and Urrutia [CU96] later rediscovered this theorem).

In this paper, we present an $O(n \log^3 n)$ time and $O(n)$ space algorithm to compute a straight-line embedding of an n-vertex outer-planar graph G in a set P of n points in the plane. Since a tree is an outer-planar graph, the $\Omega(n \log n)$ lower bound for trees [BMS95] also holds in this case, thereby implying that our algorithm is optimal to within a polylogarithmic factor. We present a simpler $O(nd)$ time and $O(n)$ space algorithm to compute a straight-line embedding of G in P where $\log n \leq d \leq 2n$ is the length of the longest vertex disjoint path in the dual of G. Therefore, the time complexity of the simpler algorithm varies between $O(n \log n)$ and $O(n^2)$. More efficient algorithms are presented for certain restricted cases. If the dual of G is a path, then an optimal $\Theta(n \log n)$ time algorithm is presented. If the given point set is in convex position then we show that $O(n)$ time suffices.

2 Notation and Preliminaries

We begin by defining some of the graph theoretic and geometric terminology used in this paper. For more details see [BM76] and [PS85].

A *graph* $G = (V, E)$ consists of a finite non empty set $V(G)$ of *vertices*, and a set $E(G)$ of unordered pairs of vertices known as *edges*. An edge $e \in E(G)$ consisting of vertices u and v is denoted by $e = uv$; u and v are called the *endpoints* of e and are said to be *adjacent* vertices or *neighbors*.

A *drawing* of a graph $G = (V, E)$ is a function which maps the vertices of G to points in the plane and edges of G to curves in the plane such that for each edge $e = uv$, the endpoints of the curve corresponding to e are the points in the plane corresponding to u and v. A drawing of G is called a *planar* drawing if no curve intersects itself or any other curve, except possibly at its endpoints. A graph is said to be planar if it admits a planar drawing. A *straight-line* drawing of a graph G is a drawing in which each edge corresponds to the line segment between its endpoints. All planar graphs admit straight-line planar drawings [Far48].

An outer-planar graph is a planar graph where every vertex is on the external face. A maximal outer-planar graph is an outer-planar graph that is no longer outer-planar with the addition of a single edge. Each internal face of a maximal outer-planar graph is a triangle. Note that an algorithm that can embed a maximal outer-planar graph can embed any outer-planar graph G simply by adding extra edges to G making it maximal, embedding the maximal graph and then removing the extra edges. Therefore, in the remainder of the paper, all outer-planar graphs are considered maximal.

Let G be a maximal outer-planar graph. Let $Ext(G)$ represent the external face of G. We adopt the convention that the vertices of a maximal outer-planar graph G are labelled $\{v_0, v_1, \ldots, v_{n-1}\}$ as they appear on $Ext(G)$ (i.e. v_i is adjacent to $v_{i+1}, i = 0, \ldots, n-1$ addition taken modulo n). An edge $e \in Ext(G)$ is an *external* edge of G.

The dual G^* of a maximal outer-planar graph $G = (V, E)$ is defined as follows. Each triangle or face (excluding the outer face) of G is a vertex of G^*. Two vertices of G^* are adjacent if the corresponding faces in G have an edge in common. Since G is maximal outer-planar, G^* is a tree with maximum vertex degree 3. See Figure 2 for an illustration.

All planar point sets are assumed to be in general position, i.e. no three points are collinear. Let P be a set of n points in the plane. Given $a, b \in P$, the open and closed line segments defined by a and b are denoted by (a, b) and $[a, b]$,

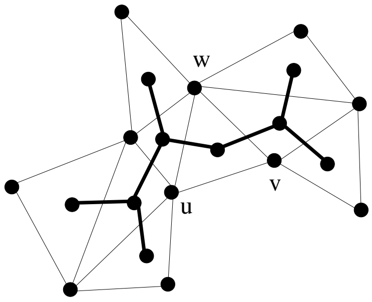

Fig. 2. A maximal outer-planar graph and its dual.

respectively. Given three points $a, b, c \in P$, by $\angle(a, b, c)$ we mean the clockwise angle between $[b, a]$ and $[b, c]$ (see Figure 3).

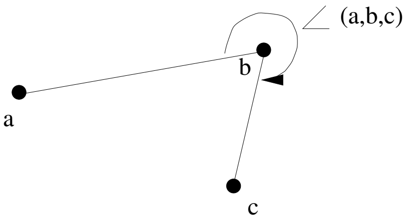

Fig. 3. Illustration of the angle (a, b, c).

3 Embedding Algorithm Outline

We begin by outlining a few of the ideas presented in [GMPP91] and [CU96]. A key concept in their embedding algorithms is the (r, s)-triangle. There are two types of (r, s)-triangles (defined below), one defined on a maximal outer-planar graph and the other defined on a point set.

In the discussion to follow, G is an n vertex maximal outer-planar graph and P is a set of n points in the plane.

Definition 1. Let u, v, w be three mutually adjacent vertices of G. Triangle $\triangle(u, v, w)$ is an (r, s)-triangle of G provided that uv is an external edge of G and the two components of $G \setminus \{u, v, w\}$ have r and s vertices, respectively, such that $r + s = n - 3$.

In Figure 2, $\triangle(u, v, w)$ is a $(5, 3)$-triangle of the graph.

Definition 2. Let r and s be two non-negative integers with $r + s = n - 3$. Let a and b be two consecutive vertices on the convex hull of P and $c \in P$. Triangle $\triangle(a, b, c)$ is an (r, s)-triangle of P provided the following holds:

1. No point of P lies in $\triangle(a, b, c)$.
2. There is a line l_c through c that intersects the interior of $\triangle(a, b, c)$ such that there are r points of $P \setminus \{a, b, c\}$ on one side of l_c and s points of $P \setminus \{a, b, c\}$ on the other side of l_c. These sets are denoted as P_r and P_s, respectively.

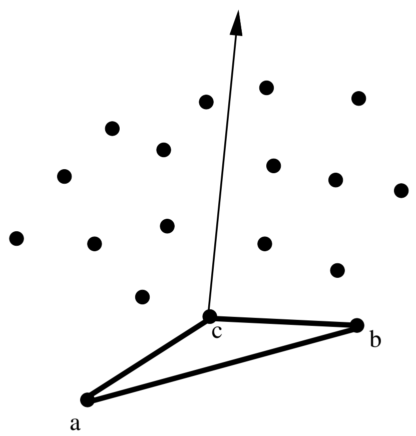

Fig. 4. An $(8, 7)$-triangle.

In Figure 4, $\triangle(a, b, c)$ is an $(8, 7)$-triangle of the point set.

The main idea behind the embedding algorithm is to find an (r, s)-triangle in G and map it to an (r, s)-triangle in P. The existence of an (r, s)-triangle in G follows from the fact that the dual of G is a binary tree. The proof of the existence of an (r, s)-triangle in P forms the basis of an embedding algorithm. A proof of the following lemma appears in [GMPP91] and [CU96]. We provide a similar but alternate proof in section 4.

Lemma 3. *[GMPP91] For any $r, s \geq 0$ such that $r + s = n - 3$ and any two consecutive vertices a, b on the convex hull of P, there always exists a point $c \in P$ such that $\triangle(a, b, c)$ is an (r, s)-triangle of P.*

Lemma 4. *[GMPP91] Let G be an n-node maximal outer-planar graph and P be a set of n points in the plane. Let a and b be two consecutive vertices on the convex hull of P. Let $e = v_i v_{i+1}$ be an external edge of G. There exists a straight-line embedding of G on P with the added constraint that v_i maps to a and v_{i+1} maps to b.*

Proof. We proceed by induction on the number of vertices of G. The result holds trivially if G has three vertices.

Inductive Hypothesis $(k < n, n > 3)$**:** Let G be a k-node maximal outer-planar graph and P be a set of k points in the plane. Let a and b be two consecutive vertices on the convex hull of P. Let $e = v_i v_{i+1}$ be an external edge of G. There exists a straight-line embedding of G on P with the added constraint that v_i maps to a and v_{i+1} maps to b.

Inductive Step $(k = n)$**:** Let G be an n-node maximal outer-planar graph and P be a set of n points in the plane. Let a and b be two consecutive vertices on the convex hull of P. Let $e = v_i v_{i+1}$ be an external edge of G.

Since G is maximal outer-planar, there is a unique vertex v_k adjacent to both v_i and v_{i+1}. The node in the dual G^* representing the triangle $\triangle(v_i, v_{i+1}, v_k)$ has degree at most two since edge $v_i v_{i+1}$ is an external edge. This implies that the removal of $\triangle(v_i, v_{i+1}, v_k)$ decomposes G into two components with cardinalities r and s respectively with $r + s = n - 3$. Therefore, $\triangle(v_i, v_{i+1}, v_k)$ is an (r, s)-triangle of G.

By lemma 3, there is a triangle $\triangle(a, b, c)$ that is an (r, s)-triangle of P. Let l_c be the line through c as defined in definition 2. Map v_i to a, v_{i+1} to b and v_k to c. By construction, the edge $[ac]$ is on the convex hull of P_r and the edge $[bc]$ is on the convex hull of P_s.

Let H_1 and H_2 be the subgraphs of G induced by $\{v_k, \ldots, v_i\}$ and $\{v_{i+1}, \ldots, v_k\}$. Both H_1 and H_2 are maximal outer-planar, and edge $v_i v_k$ is an external edge of H_1 and edge $v_{i+1} v_k$ is an external edge of H_2. Since both H_1 and H_2 have less than n vertices, by the inductive hypothesis, H_1 can be embedded in P_r with edge $v_i v_k$ mapping to $[ac]$ and H_2 can be embedded in P_s with edge $v_{i+1} v_k$ mapping to $[bc]$. The result follows.

As is often the case with inductive proofs, the proof of Lemma 4 directly implies an algorithm to embed an outer-planar graph on a point set. The main steps of the algorithm are outlined in Figure 5. The maximal outer-planar graph to be embedded is G. All index manipulation is done modulo n. The time taken by this algorithm depends on the time taken to perform steps 1-6. In essence, the time can be expressed recursively as $T(n) = T(n-k) + T(k) + \psi(n)$, with $1 \le k \le n-1$ and where $\psi(n)$ represents the time taken to perform steps 1-6.

The adjacency information of the graph G can be stored in a standard data structure such as the doubly-connected edge list (DCEL) [PS85]. However, in the algorithm, there is no need to modify the adjacency information, but merely record the indicies of the vertices in the input graph in the recursive calls. All adjacency queries, such as those made in step 1 of the algorithm are made on the DCEL of G. Since each edge in G is adjacent to two triangles, the vertex v_k in step 1 can be found in constant time by identifying the unique vertex whose index k falls in the range delimited by I_s and I_e.

Step 2 can also be computed in constant time since the cardinalities of the two sets can be computed from indicies of the three vertices forming the triangle. Step 5 is a constant time operation. Finally, step 6 is also a constant time operation, given the indicies of the three vertices forming the triangle.

$Embed(I_s, I_e, v_i, v_{i+1}, P, a, b)$
I_s and I_e are the start and end indicies of the vertices on the external face of the graph.
The edge $v_i v_{i+1}$ is an external edge of the outer-planar graph.
P is a point set with points a, b on its convex hull.

1. Find the unique vertex v_k in G (where k lies in the interval defined by I_s and I_e) adjacent to v_i and v_{i+1}.
2. Since $\triangle(v_i, v_{i+1}, v_k)$ is an (r,s)-triangle of G, compute the cardinalities r and s.
3. Find $c \in P$, such that triangle $\triangle(a,b,c)$ that is an (r,s)-triangle of P.
4. Compute P_r and P_s, the sets on either side of the line l_c, respectively.
5. Map v_i to a, v_{i+1} to b, and v_k to c.
6. Let H_1 and H_2 be the subgraphs of G induced by $\{v_k, \ldots, v_i\}$ and $\{v_{i+1}, \ldots, v_k\}$. The start and end indices for H_1 are k and i, respectively and for H_2 are $i+1$ and k, respectively.
7. If the number of vertices in $H_1 \geq 3$ then $Embed(k, i, v_k, v_i, P_r, a, c)$.
8. If the number of vertices in $H_2 \geq 3$ then $Embed(i+1, k, v_{i+1}, v_k, P_s, b, c)$.

Fig. 5. Outline of algorithm to embed G in P.

Therefore, the main difficulty comes from steps 3 and 4: computing an (r,s)-triangle in a point set. The complexity of the whole algorithm depends on these two steps since the other four steps are constant time operations. In the next section, we present a method for computing an (r,s)-triangle in a point set in $O(n)$ time with no preprocessing which will form the basis of our embedding algorithms.

4 Simple Embedding Algorithm

In this section, we present a simple method for finding an (r,s)-triangle in a point set and show how it is used in the simple embedding algorithm.

Let P be an n point set with a and b two adjacent vertices on the convex hull of P. In the discussion to follow, for any line l through a and not b, the open half-plane containing b shall be referred to as the right half-plane of l; similarly, for any line l through b and not a, the open half-plane containing a shall be referred to as the left half-plane of l.

Lemma 5. *For any $r, s \geq 0$ such that $r + s = n - 3$ and any two consecutive vertices a, b on the convex hull of P, there always exists a point $c \in P$ such that $\triangle(a,b,c)$ is an (r,s)-triangle of P.*

Proof. Let l_1 be a line through a with $s+1$ points of P (excluding b) in the right half-plane of l_1. Let $L(a)$ represent these $s+1$ points. Let l_2 be a line through b and a point c from $L(a)$ such that the left half-plane of l_2 contains no points of $L(a)$ (refer to Figure 6). Let l_3 be the line through a and c. Triangle $\triangle(a,b,c)$ is an (r,s)-triangle of P. Since there are at least s points of P in the right half-plane of l_2 and at most s points in the right half-plane of l_3 there must be a line l_c through c intersecting the interior of $\triangle(a, b, c)$ with r points to one side and s points to the other (excluding a, b, c).

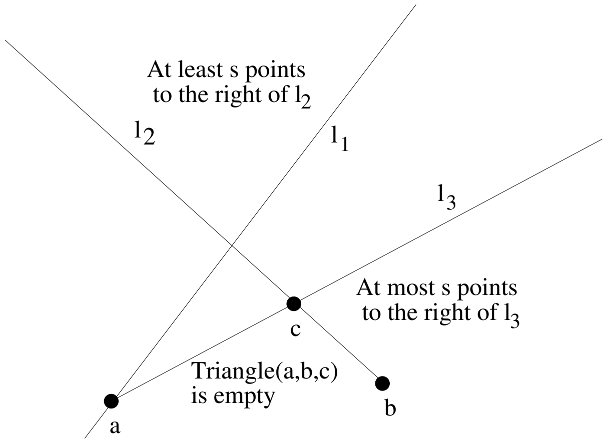

Fig. 6. Computing an (r, s)-triangle.

From the proof of lemma 5, the steps involved in finding point $c \in P$ and line l_c such that $\triangle(a, b, c)$ is an (r, s)-triangle of P are summerized below:

1. Find the line l_1 through a.
2. Compute the set $L(a)$.
3. Find the line l_2 through b, identifying point c.
4. Find the line l_c.

Recall that selecting the i^{th} smallest element in an unsorted list of n elements can be achieved in $O(n)$ time (see [CLR90]). Let $x \in P$ be the point such that $\angle(x, a, b)$ is the $(s+2)^{nd}$ smallest. The point x can be found in $O(n)$ time using linear selection. The line l_1 through a and x has $s+1$ elements (excluding b) in the right half-plane. The set $L(a)$ can be constructed in $O(n)$ time once l_1 is found. Given $L(a)$, notice that l_2 is simply the through b and the point $y \in L(a)$ such that $\angle(a, b, y)$ is the smallest over all points in $L(a)$. Therefore, l_2 can be computed in $O(n)$ time. Finally, l_c can be found in $O(n)$ time by computing the point $z \in P$ such that $\angle(z, c, b)$ is the $(s+1)^{st}$ among all points in P. Therefore, given a set of n points, an (r, s)-triangle can be computed in $O(n)$ time. This immediately gives an $O(n^2)$ time and $O(n)$ space algorithm for embedding an outer-planar graph in a point set since the recurrence for the algorithm *Embed(...)* becomes $T(n) = T(n-k) + T(k) + O(n)$ which solves to $O(n^2)$.

Upon further consideration of the recurrence, we notice that the complexity of the algorithm is actually dependant on the length of the longest path in the dual of G. The algorithm is initiated with an initial invocation of $Embed(0, n-1, v_i, v_{i+1}, P, a, b)$. At each invocation, the algorithm embeds an (r, s)-triangle and makes at most two recursive calls with smaller problem instances. The calling relation forms a binary tree, which we refer to as the *recursion tree* for graph G, denoted as RT_G. An internal node of this tree has at least one child, and is an instance of *Embed(...)* where an (r, s)-triangle is embedded with at least one of r or s being non-zero initiating at least one recursive call. A leaf of the recursion tree is an instance of *Embed(...)* where the size of the graph to be embedded is 3. The root of the tree represents the initial call and the depth of a node in the tree represents its level of recursion. Let d be the length of longest path in the dual tree G^*. The depth of the recursion tree RT_G cannot exceed d since every root to leaf path in RT_G represents a path in G^*.

Lemma 6. *The depth of the recursion tree RT_G does not exceed d, where d is the length of the longest path in the dual tree G^*.*

Since at each level, the graph G is partitioned, the sum of the sizes of all the problems at a particular level of RT_G is $O(n)$. The amount of time spent in one invocation of *Embed(...)* excluding recursive calls is linear in the size of the graph. All of the steps of the algorithm (refer to Figure 5) are constant time except for the two steps involving the computation of an (r, s)-triangle, which we showed is linear in the size of the problem. Therefore, the amount of time spent by the algorithm is $O(n)$ per level of RT_G.

Theorem 7. *Given an n-vertex outer-planar graph G and a set P of n points in the plane, G can be straight-line embedded in P in $O(nd)$ time and $O(n)$ space where d is the length of the longest path in the dual of G.*

5 Near-optimal Embedding Algorithm

Our more efficient algorithm for embedding outer-planar graphs uses segments from the convex hull to avoid intersections between embedded edges. Consequently, we need efficient access to the convex hull of points. Moreover, we need the ability to insert and delete points from the convex hull as we embed (r, s)-triangles. Overmars and van Leeuwen's [OvL81] dynamic convex hull structure permits arbitrary insertion into and deletion from a set of points while maintaining the convex hull of the point set. Each update (insertion or deletion) costs at most $O(\log^2 n)$ time over a sequence of $O(n)$ updates.

If the points of P are placed in a dynamic convex hull maintenance structure that supports insertions and deletions in $O(\log^2 n)$ time then we can find an (r, s)-triangle without resorting to a linear time selection. We review the method for computing an (r, s)-triangle given the maintenance structure. *CM* will refer to the convex hull maintenance structure. We can insert and delete points from

CM in $O(\log^2 n)$ time. Given a point $x \in CM$, we can recover the point adjacent to x on the current convex hull in $O(\log n)$ time.

Without loss of generality, assume that $s \leq r$. Let $x \in P$ be the point such that $\angle(x, a, b)$ is the $(s+2)^{nd}$ smallest. The point x can be found in $O(s \log^2 n)$ time by deleting $s+2$ times the convex hull point adjacent to a starting with b. Store the deleted points in order of deletion into $L(a)$. The line l_1 through a and x has $s+1$ elements (excluding b) in the right half-plane. Given $L(a)$, notice that l_2 is simply the line through b and the point $y \in L(a)$ such that $\angle(a, b, y)$ is the smallest over all points in $L(a)$. Therefore, l_2 can be computed in $O(s)$ time since $L(a)$ has $s+2$ points. Finally, to compute l_c re-insert all the points of $L(a)$ into *CM*. Delete convex hull points adjacent to a starting with b until c is adjacent to a. If there are s points (excluding b) to the right of the line through a and c then l_c is this line. Otherwise, to find l_c continue deleting the convex hull points adjacent to c (different from a) until $s+1$ points have been deleted in total from *CM*. Store all the deleted points in $L(a)$. Notice that *CM* is now a convex hull maintenance structure for P_r. There are $s+1$ points in P_s. In $O(s \log^2 s)$, a *CM* structure can be built for P_s. Therefore, the revised complexity of the algorithm is $T(n) = T(n-k) + T(k) + O(\min(k, n-k) \log^2 n)$ where $1 \leq k \leq n-1$. This recurrence solves to $T(n) = O(n \log^3 n)$. Building the initial *CM* for P cost $O(n \log^2 n)$, therefore, we have the following theorem.

Theorem 8. *Given an n-vertex outer-planar graph G and a set P of n points in the plane, G can be straight-line embedded in P in $O(n \log^3 n)$ time and $O(n)$ space.*

Our algorithm is optimal to within a polylogarithmic factor since an $\Omega(n \log n)$ lower bound for the problem was shown in [BMS95].

6 Restricted Case

If the dual of G is a tree, then notice that G can be embedded simply by computing $(r, 1)$-triangles. This immediately implies that our near-optimal algorithm will run in time $T(n) = T(n-1) + O(\log^2 n)$ which solves to $T(n) = O(n \log^2 n)$. However, when computing $(r, 1)$-triangles, we do not need to re-insert points into the convex hull maintenance structure in order to compute l_c. Since, we do not need to insert points into the convex hull but simply delete them; we opt for a deletion-only convex hull maintenance structure [Cha85, HS92], which provides better amortized time complexities for point deletions than Overmars and van Leeuwen's method.

In [HS92], the point deletion operation removes a point from the convex hull maintenance structure in $O(\log n)$ amortized time (amortized over the sequence of n deletions). Consequently, by using a deletion-only convex hull structure, the running time of the algorithm is summerized by the recurrence $T(n) = T(n-1) + O(\log n)$ which resolves to $T(n)$ is $O(n \log n)$. This is optimal since the lower bound proved in [BMS95] still holds in this restricted case.

Theorem 9. *If the dual of the input graph G is a path, G can be embedded into a point set P in optimal $\Theta(n \log n)$ time.*

If the input point set P is in convex position, then $O(n)$ time and space is sufficient. We assume that the input point set is given in an array A, ordered in clockwise fashion as the points appear on the convex hull of P. Given an (r, s)-triangle of G, finding an (r, s)-triangle in P can be achieved in $O(1)$ time by simply finding the index into array A which splits the array into two sub-arrays of size r and s respectively. Therefore, the recurrence for algorithm *Embed(...)* is $T(n) = T(n-k) + T(k) + O(1)$ which implies that $T(n)$ is $O(n)$.

Theorem 10. *If the input point set P is in convex position then $O(n)$ time and space suffice to straight-line embed G into P.*

7 Conclusions

We presented an $O(n \log^3 n)$ time and $O(n)$ space algorithm to compute a straight-line embedding of an n-vertex outer-planar graph G in a set P of n points in the plane. Since a tree is an outer-planar graph, the $\Omega(n \log n)$ lower bound for trees [BMS95] also holds in this case, thereby implying that our algorithm is optimal to within a polylogarithmic factor. We presented a simpler $O(nd)$ time and $O(n)$ space algorithm to compute a straight-line embedding of G in P where $\log n \leq d \leq 2n$ is the length of the longest vertex disjoint path in the dual of G. Finally, we showed that if the dual of G is a path, then $\Theta(n \log n)$ time and $O(n)$ space are sufficient and if the input point set is in convex position then $O(n)$ time and space suffice.

We conclude with two open problems:

1. Can a $\log^2 n$ factor be shaved off our embedding algorithm, i.e. is there an optimal $O(n \log n)$ time algorithm to embed an outer-planar graph on a point set?
2. Given a planar graph G and a point set P, what is the complexity of deciding if G can be embedded into P?

Acknowledgements: The author wishes to thank Kilani Ghoudi for helpful discussions on this topic, and Janos Pach for pointing out some missing references.

References

[BDLL95] P. Bose, G. Di Battista, W. Lenhart, and G. Liotta. Proximity constraints and representable trees. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 340–351. Springer-Verlag, 1995.

[BLL96] P. Bose, W. Lenhart, and G. Liotta. Characterizing proximity trees. *Algorithmica*, 16:83–110, 1996.

[BM76] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications.* Elsevier Science, New York, NY, 1976.

[BMS95] P. Bose, M. McAllister, and J. Snoeyink. Optimal algorithms to embed trees in a point set. *Journal of Graph Algorithms and Applications*, to appear. Also appears in Proceedings of Graph Drawing GD'95, LNCS 1027, pp. 64–75, 1995.

[Cha85] B. Chazelle. On the convex layers of a planar set. *IEEE Trans. on Inf. Theory*, IT-31:509–517, 1985.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms.* MIT Press, Cambridge, Mass., 1990.

[CP95] P. Crescenzi and A. Piperno. Optimal-area upward drawings of AVL trees. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 307–317. Springer-Verlag, 1995.

[CU96] N. Castaneda and J. Urrutia. Straight line embeddings of planar graphs on point sets. In *Proc. Eighth Canadian Conf. on Comp. Geom.*, pages 312–318, 1996.

[DETT94] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.

[EW94] P. Eades and S. Whitesides. The realization problem for Euclidean minimum spanning trees is NP-hard. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 49–56, 1994.

[Far48] I. Fary. On straight line representation of planar graphs. *Acta Sci. Math. Szeged*, 11:229–233, 1948.

[GMPP91] P. Gritzmann, B. Mohar, J. Pach, and R. Pollack. Embedding a planar triangulation with vertices at specified points (solution to problem e3341. *American Mathematical Monthly*, 98:165–166, 1991.

[HS92] John Hershberger and Subhash Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32:249–267, 1992.

[IPTT94] Y. Ikebe, M. Perles, A. Tamura, and S. Tokunaga. The rooted tree embedding problem into points in the plane. *Discrete & Computational Geometry*, 11:51–63, 1994.

[KLTT93] G. Kant, G. Liotta, R. Tamassia, and I. Tollis. Area requirement of visibility representations of trees. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 192–197, Waterloo, Canada, 1993.

[MA88] J. Manning and M. J. Atallah. Fast detection and display of symmetry in trees. *Congressus Numerantium*, 64:159–169, 1988.

[MS91] C. Monma and S. Suri. Transitions in geometric minimum spanning trees. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 239–249, 1991.

[OvL81] M. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23:166–204, 1981.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

[PT93] János Pach and Jenő Törőcsik. Layout of rooted trees. In W. T. Trotter, editor, *Planar Graphs*, volume 9 of *DIMACS Series*, pages 131–137. American Mathematical Society, 1993.