Pointed and Colored Binary Encompassing Trees

Michael Hoffmann
Institute for Theoretical Computer Science
ETH Zürich
CH-8092, Switzerland
hoffmann@inf.ethz.ch

Csaba D. Tóth
Department of Mathematics
Massachusetts Institute of Technology
Cambridge, MA 02139, U.S.A.
toth@math.mit.edu

ABSTRACT

For n disjoint line segments in the plane we construct in optimal $O(n \log n)$ time an encompassing tree of maximum degree three such that at every vertex all incident edges lie in a halfplane defined by the incident input segment. In particular, this implies that each vertex is *pointed*. Furthermore, we show that any set of colored disjoint line segments (for each segment one endpoint is colored red and the other endpoint is colored blue) has a *color conforming* encompassing tree of maximum degree three.

Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*computations on discrete structures, geometrical problems and computations*; G.2.2 [Discrete Mathematics]: Graph Theory—*Trees*

General Terms: Algorithms, Theory

Keywords: line segments, spanning tree, bounded degree, pseudo-triangulation

1 Introduction

Spanning trees defined on disjoint objects in the plane are fundamental structures in computational geometry. Complex planar objects are often modeled by their boundary polygons which, in turn, can be represented as a planar straight line graph (PSLG). An *encompassing graph* for a PSLG G is a connected PSLG on the same vertex set that contains all edges of G. Constrained Delaunay triangulations [18] are well-known examples of encompassing graphs. Particularly well-studied are encompassing graphs for disjoint line segments in the plane. In this context, a set of disjoint segments is regarded as a PSLG that is a perfect matching.

Since a triangulation of the free space around n disjoint line segments is an encompassing graph, it is easy to construct an encompassing tree in $O(n \log n)$ time. Research focused on optimizing various parameters (length, degree, etc.) of encompassing trees for n segments. Bose et al. [7, 6] showed that any finite set of disjoint line segments in the plane admits an encompassing tree of maximum degree *three*. They also gave an $O(n \log n)$ time algorithm to construct such an encompassing tree for n input segments. Both the degree bound and the runtime are best possible (the latter in the algebraic computation tree model).

In this paper, we extend the result of Bose et al. in two essentially different directions. We show that an encompassing tree with maximum degree three can be endowed with some additional properties for any input set of disjoint segments. Our first result asserts that one can efficiently compute a *pointed* encompassing tree. A PSLG is *pointed* if and only if for every vertex v all edges incident to v lie in a closed halfplane whose boundary contains v.

THEOREM 1. *Let S be a set of n disjoint line segments in the plane. There exists an encompassing tree of maximum degree three such that for every vertex v all incident edges lie in a halfplane bounded by the line through the segment of S whose endpoint is v. Moreover, such a tree can be constructed in $O(n \log n)$ time and linear space.*

Our second result provides a colorful extension to the result of Bose et al. [7, 6]. A graph is *vertex-colored* if every vertex has a color and no edge is monochromatic. The set of input segments can be considered a vertex-colored matching.

THEOREM 2. *For any set of n disjoint line segments in the plane, each of which has a red and a blue endpoint, we can construct a vertex-colored encompassing tree of maximum degree three in polynomial time.*

In fact, we prove a theorem for a slightly broader class of vertex-colored PSLGs.

THEOREM 3. *For any vertex-colored planar straight line forest G on n vertices with no singleton component, we can construct in polynomial time a vertex-colored encompassing tree G' such that $\deg_G(v) \leq \deg_{G'}(v) \leq \deg_G(v) + 2$ for every vertex v of G'.*

Our proof for Theorem 3 is constructive, but our algorithm is based on multiple visibility sweeps, and so we cannot expect its runtime to be optimal. Also, we do not know if the combination of Theorem 1 and 2 holds: Does every set of disjoint vertex-colored segments admit an encompassing tree of maximum degree three that is pointed and vertex-colored simultaneously?

SCG'05, June 6–8, 2005, Pisa, Italy.

Motivation and related work. Pointed PSLGs are closely related to minimum pseudo-triangulations, which have numerous applications in motion planning [21], kinetic data structures [17], collision detection [1], and guarding [20]. Streinu [21] showed that a minimum pseudo-triangulation of V is a pointed PSLG on the vertex set V with a maximal number of edges. As opposed to triangulations, there is always a bounded (vertex-)degree pseudo-triangulation of a set of points in the plane [16]. A bounded degree pointed encompassing tree for disjoint segments leads to a bounded degree pointed encompassing pseudo-triangulation, due to a result of Aichholzer et al. [4].

Recently, Hoffmann, Speckmann, and Tóth [8] have shown that for every n disjoint segments in the plane a *pointed* binary encompassing tree can be constructed in $\widetilde{O}(n^{4/3})$ time. Our Theorem 1 extends this result in two aspects: We construct an encompassing tree in optimal $O(n \log n)$ time and guarantee a stronger sense of pointedness where all edges incident to a vertex v lie in a halfplane aligned with the input segment whose endpoint is v.

A simple construction (Figure 1a) shows that not every set of n disjoint segments in the plane admits an *encompassing path*. But there is always a path that encompasses $\Theta(\log n)$ segments and does not cross any other input segment [9]. Also, there is always an encompassing graph which is Hamiltonian [10].

Vertex-colored PSLGs and geometric graphs have also received considerable attention recently. (A *geometric graph* is a straight line graph whose edges may cross.) In these problems, the input typically consists of a set R of red points and a set B of blue points in the plane; we ask for certain types of vertex-colored PSLGs. Every vertex-colored graph is a subgraph of the complete bipartite geometric graph $K(R, B)$. A pioneer result in this area claims that any n red and n blue points in the plane can be covered by a vertex-colored planar straight line *matching* (e.g., a minimum length bipartite matching is planar).

Akiyama and Urrutia [2] found n red and n blue points in the plane for which no vertex-colored planar straight line Hamiltonian tour exists. Kaneko, Kano, and Yoshimoto [15] proved that such a Hamiltonian tour may have up to $n-1$ self-crossings. Kaneko and Kano [14] showed that if $|R| = \Theta(|B|^2)$ then all *red* points can be covered by a vertex-colored planar straight line path.

Kaneko [12] proved that any n red and n blue points in the plane can be covered by a vertex-colored planar straight line tree with maximum degree three. Our Theorem 3 states that such a tree can encompass a given vertex-colored matching of the $2n$ input points. Very recently, Hurtado et al. [11] showed that every set of disjoint vertex-colored edges admits a vertex-colored encompassing tree. Theorem 3 extends their result and shows that such a tree exists with maximum degree three. For other recent results on geometric red-blue graphs, we refer the reader to an excellent survey by Kaneko and Kano [13].

A minimum[1] encompassing tree for a set of n disjoint line segments may require a vertex of degree seven, and, conversely, a minimum weight encompassing tree of maximum degree seven can always be obtained greedily [7]. On the other hand, a color-conforming minimum encompassing tree for a set of colored disjoint line segments may require a vertex of linear degree [5].

[1] Edge weight is the distance between the two endpoints.

Organization. We set forward the proof of our two main results as follows. First we briefly define a few commonly used geometric terms in Section 2. We prove Theorem 1 in Section 3 via a *tunnel graph* that we define for a convex partition of our input set of n line segments. A crucial lemma on constructing *connected* tunnel graphs in $O(n \log n)$ time is presented in Section 4. Then we proceed with the proof of Theorem 3 in Sections 5 and 6. Section 7 states a conjecture regarding an extension of Theorem 3 to arbitrary vertex-colored PSLGs. We conclude in Section 8 with a few more related open problems.

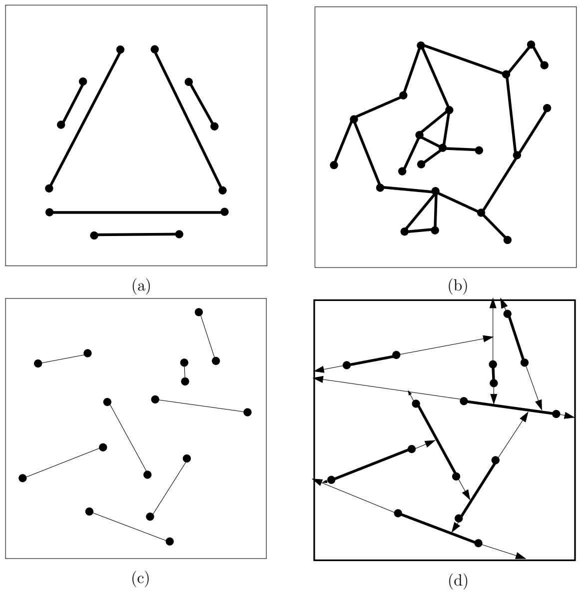


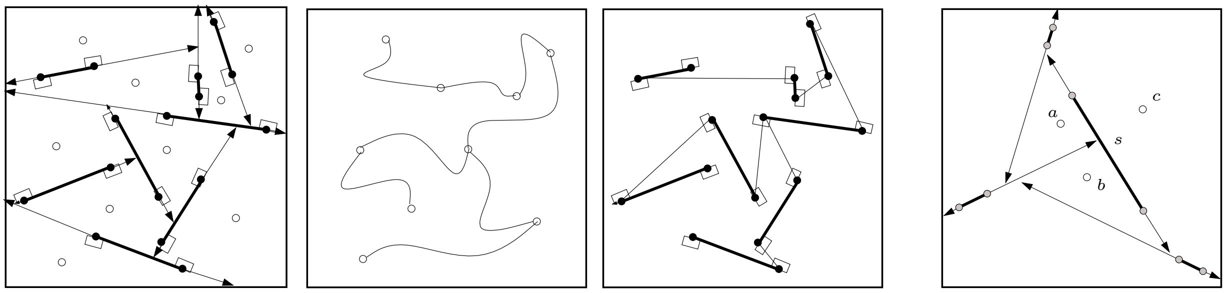
Figure 1. Six segments which do not admit an encompassing path (a), a connected PSLG with 4 faces including the outer face (b), disjoint segments (c), and one of their convex partitions (d).

2 Definitions

Polygons. A *polygon* P is a finite sequence $(p_1, p_2, \ldots, p_k)$ of points in the plane. The set of vertices of polygon P is $V(P) = \{p_1, p_2, \ldots, p_k\}$, and is the set of edges is $E(P) = \{p_1p_2, p_2p_3, \ldots, p_{k-1}p_k, p_kp_1\}$.

A *weakly simple polygon* is a polygon without self-crossings. Any weakly simple polygon P partitions $\mathbb{R}^2 \setminus P$ into an interior and an exterior. The interior of P is denoted by $\mathrm{int}(P)$, while the closure of the interior is the closed polygonal domain $\overline{P} = \mathrm{int}(P) \cup P$.

A planar set $D \subset \mathbb{R}^2$ is *polygonal* if it is the union of a finite number of line segments and triangles. The boundary of every simply connected polygonal set D can be covered by a weakly simple polygon ∂D. In particular, every planar straight line tree A can be covered by a weakly simple polygon ∂A. Note, however, that a vertex of the tree A can occur several times among the vertices of ∂A. One way to distinguish distinct occurrences of the same point along ∂A is by the *angles* (three consecutive vertices) along ∂A.



(a) Partition. (b) Tunnel Graph. (c) Encompassing Tree. (d) Disconnected.

Figure 2. An example for a partition with an assignment (a), the corresponding tunnel graph (b), the resulting tree (c). A partition for which no assignment gives a connected tunnel graph (d).

Faces of a PSLG. The complement of a connected PSLG A can have several connected components, which we call the *faces* of A. The boundary of each face F can be covered by a weakly simple polygon ∂F. We say that a vertex v_i of ∂F is convex (reflex) if the angle $\angle v_{i-1}v_iv_{i+1}$ whose angular domain contains F is less than (more than) $180°$. This angle is the exterior angle of ∂F for the outer face, and the interior angle of ∂F for all bounded faces.

Convex partition and cells. The free space around n disjoint line segments in the plane can be partitioned into $n+1$ convex cells by the following well known partitioning algorithm. (For simplicity, we assume that no three segment endpoints are collinear.) For every segment endpoint p of every input segment s_p, extend s_p beyond p until it hits another input segment, a previously drawn extension, or to infinity. There may be many different partitions depending of the order in which we consider the segment endpoints, but the number of convex cells is always $n+1$.

3 Tunnel Graphs

Consider a set of disjoint segments S in the plane and a convex partition $P(S)$ obtained by the above algorithm. Let us assign every segment endpoint p to an incident cell $\tau(p)$ of the partition. We define the *tunnel graph* $T(S, P(S), \tau)$ for S, a partition $P(S)$, and an assignment τ as follows: The nodes of T correspond to the convex cells of $P(S)$. Two nodes a and b are connected by an edge if and only if there is a segment $pq \in S$ such that $\tau(p) = a$ and $\tau(q) = b$. The tunnel graph is clearly planar; and T has $n+1$ nodes and n edges, therefore it is connected if and only if it is a tree.

Theorem 4. *For any set S of n disjoint line segments, we can construct in $O(n \log n)$ time and linear space a convex partition $P(S)$ and an assignment τ such that the tunnel graph $T(S, P(S), \tau)$ is a tree.*

Note that the choice of the convex partition is important in Theorem 4: Figure 2(d) shows four disjoint line segments and a convex partition such that there is no assignment for which the tunnel graph is connected. (Consider the endpoints of the segment s: The left endpoint is the only segment incident to Cell a and must hence be assigned to a. Similarly, the right endpoint of s has to be assigned to Cell b. But then regardless of the assignment for the other points, $\{a, b\}$ is always a component of size two in the tunnel graph.) We obtain Theorem 1 as a corollary of Theorem 4.

Proof of Theorem 1. Consider a partition $P(S)$ and an assignment τ provided by Theorem 4. We construct a binary encompassing tree as follows: In each cell connect all segment endpoints assigned to it by a simple path; for example, connect them in the order in which they appear along the boundary of the cell.

The resulting graph is clearly a PSLG that contains all the input segments. The maximum degree is three because we add at most two new edges at every segment endpoint. It remains to prove connectivity. Let p and r be two segment endpoints. We know that the tunnel graph is connected, so there is an alternating sequence of cells and segments $(a_1 = \tau(p), p_1q_1, a_2, \ldots, p_{k-1}q_{k-1}, a_k = \tau(r))$ such that $\tau(p_i) = a_i$ and $\tau(q_i) = a_{i+1}$, for every i. As all segment endpoints assigned to the same cell are connected, this path corresponds to a path in the constructed graph. □

4 Convex Partitioning

This section is devoted to the proof of Theorem 4. Consider a set S of n disjoint line segments in the plane and let R be an axis-parallel box which encloses all segments from S. We use a two-phase line sweep algorithm to

- partition the free space around the segments into $n+1$ convex cells and to
- assign an incident cell to every segment endpoint.

The first phase is a left-to-right sweep: We extend every input segment beyond its right endpoint until the extension hits another segment, another extension, or the boundary of R. If two extensions meet, an arbitrary one continues and the other one ends (Figure 3(b)). The segments and their right extensions jointly form a *right extension tree* in the plane whose root correspons to the the boundary of R (right extensions may hit the boundary of R at several points, these points are glued into a sinlge root vertex of this tree). The free space of the input segments and their right extensions is a simply connected set $C_0 \subset R$. Order the segments $s_1, \ldots, s_n$ according to the order of their left endpoints along the boundary ∂C_0 (in clockwise direction starting from upper left corner of R).

In the second phase, the left extensions of the segments $s_1, \ldots, s_n$ are inserted one by one. Denote by $\mathcal{A}_i$, $0 \le i \le n$, the arrangement of the input segments, all their right extensions, and the left extensions of $s_1, \ldots, s_i$. At the beginning of the second phase, no left extension has been drawn yet. We face the arrangement $\mathcal{A}_0$ in which there is only one single cell C_0. After the second phase, the arrangement to be considered is $\mathcal{A}_n$, which consists of $n+1$ convex cells.

LEMMA 5. *There exists an assignment τ for the segment endpoints such that the corresponding tunnel graph is connected.*

PROOF. We define the assignment τ on the endpoints of s_i, $i = 1, 2, \ldots, n$, as soon as the left extension γ_i of s_i is inserted. At this point we have an arrangement $\mathcal{A}_{i-1}$ that consists of i cells and a partial assignment τ on the endpoints of the first $i-1$ segments. $\mathcal{A}_{i-1}$ and τ define a tunnel graph T_{i-1} on i nodes. We choose the assignment at the endpoints of s_i inductively such that the resulting tunnel graph T_i remains connected. Clearly T_0 is connected because it is a graph on one node only.

For the induction step consider the ray γ_i that splits a cell C_i of $\mathcal{A}_{i-1}$ into two cells C_i' and C_i'' of $\mathcal{A}_i$. Correspondingly, a node of T_{i-1} is split into two nodes that are in different components of the resulting graph T_{i-1}'. The left endpoint p_i of s_i is incident to both C_i' and C_i'' because p_i is the source of the ray γ_i that separates both cells. The right endpoint q_i, however, may be incident to neither C_i' nor C_i''. We always assign q_i to the cell lying above q_i. Then p_i is assigned to C_i' or C_i'', whichever lies in the other component of T_{i-1}' as $\tau(q_i)$. As T_{i-1}' has exactly two components, this assignment ensures that the resulting tunnel graph T_i is connected. □

We have shown that there exists an assignment τ for which the tunnel graph T is connected. It remains to prove that such an assignment can be computed in $O(n \log n)$ time. We assign every right segment endpoint to the cell lying above it. In order to determine the assignment τ on each right segment endpoint in $O(\log n)$, we devise a data structure on the arrangement $\mathcal{A}_\rangle$.

Data structure. For each cell C of $\mathcal{A}_{i-1}$, we maintain a doubly linked list of all segment endpoints and vertices along ∂C. The assignments τ carries one bit information for each segment endpoint r: It assigns r to the cell lying *below* or *above* r. We can insert a right extension γ_i by splitting the doubly connected list of of C_i into C_i' and C_i'' in constant time. For each vertex v of the right extension tree, we store the interval $g(v) \subset [1, n]$ such that the descendants of v contain the left segment endpoints p_j, for $j \in g(v)$. We maintain a *coloring* on the segments and their right and left extensions: Every input segment and every right extension is *blue*. The color of left extensions is defined recursively: γ_i is blue if its left endpoint hits a blue segment, otherwise it is *red*. We also maintain an index $\mathrm{ind}(e)$ for every blue input segment or blue extension. The index of s_i or its right extension is i. If γ_i hits a segment of index j then $\mathrm{ind}(\gamma_i) = j$.

Assignment rule. For every left segment endpoint p_i, we define the assignment $\tau(p_i)$ according to the following rule:

> If γ_i is blue and $v_i \notin s_i$ where v_i is the deepest vertex in the right extension tree such that $[\mathrm{ind}(\gamma_i), i] \subseteq g(v_i)$, then we assign p_i to the cell above it, otherwise to the cell below it.

It takes $O(\log n)$ time to find v_i in the right extension tree, and so $\tau(p_i)$ can be computed for all $i = 1, 2, \ldots, n$ in $O(n \log n)$ time.

PROPOSITION 6. *For every $i = 1, 2, \ldots, n$, if T_{i-1} is connected and we choose the assignment $\tau(p_i)$ by the above rule, then the tunnel graph T_i is also connected.*

PROOF. We define an orientation on the input segments and their extensions. Every segment and every right extension is directed to the right, every left extension is directed to the left. Note that there are no cycles in this orientation. For every $i = 1, 2, \ldots n$, we define a curve β_i through p_i: two branches of β_i start out from p_i to the left along γ_i and to the right along s_i, they follow the above orientation until the two branches meet or until both hit the bounding box R. Curve β_i partitions R into two *regions* A_i and B_i such that p_i lies on their common boundary. Observe that the curve does not pass through any left segment endpoint, and recall that every right segment endpoint q_j, is assigned to the region above q_j.

We verify by a case analysis that s_i is the only segment whose left and right endpoints are assigned to regions A_i and B_i, respectively, and the assignment rule assigns p_i and q_i to distinct regions. That is, if we choose $\tau(p_i)$ contrary to the assignment rule, then T_i would be disconnected. This implies that if $\tau(p_i)$ obeys the assignment rule,then T_i must be connected.

Case (1). If γ_i is red then β_i is x-monotone and its two branches pass through right segment endpoints only, so p_i is the only vertex that might be assigned to the region below β_i. *Case (2).* Suppose that γ_i is blue: The left branch of β_i is x-monotone decreasing until it hits a segment or a right extension, from that point it continues in x-monotone increasing direction until it hits the right branch or the boundary R. Let A_i be the region *nonadjacent* to the left side of R. *Subcase (2a).* If γ_i hits a segment s_j, $j > i$, or its right extension, then A_i must be below s_i. Therefore, B_i is above q_i and A_i is below p_i. *Subcase (2b).* If γ_i hits a segment s_j, $j < i$, or its blue extension, then A_i is above s_i, so we know that A_i is above p_i. The rightmost point of A_i is v_i. The only case where A_i does not lie above q_i is that $v_i \in s_i$. □

PROOF OF THEOREM 4. The existence of a convex partition and an assignment that leads to a connected tunnel graph was shown in Lemma 5. It remains to show the claimed runtime bound.

The arrangement $\mathcal{A}_n$ can be constructed using a standard line sweep algorithm in $O(n \log n)$ time and linear space. First the right extensions are handled in a left-to-right sweep. Then the left extension are inserted in a right-to-left sweep. Whenever two extensions meet, only one of them continues. Therefore the combinatorial complexity of $\mathcal{A}_n$ is $O(n)$. Any type of incidence and adjacency information —such as which two cells a segment endpoint is adjacent to— can be extracted from this arrangement. The intervals $g(v)$ can be computed in a simple traversal of the right extension tree in $O(n)$ time, the coloring of the right extensions is computed recursively in $O(n)$ time, too.

Our assignment rule allows us to choose an assignment for every left segment endpoint in $O(\log n)$ time. By Proposition 6, it maintains a connected tunnel graph for every $i = 1, 2, \ldots n$. Altogether, the runtime is $O(n \log n)$ and the space consumption is linear. □

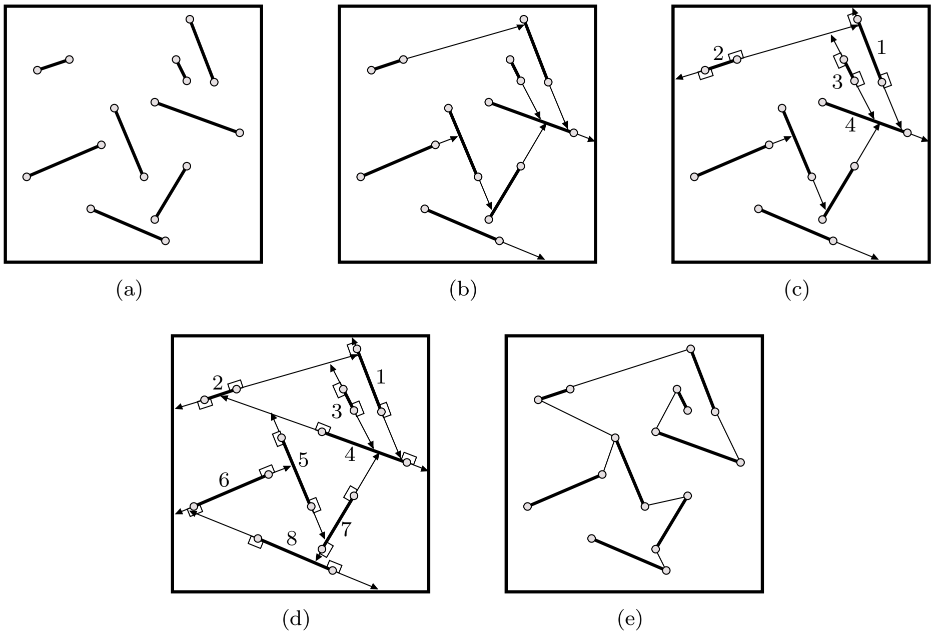


Figure 3. Constructing the partition: First all right extensions (b), then the left extensions are inserted one by one (c) and (d), and from the final partition together with the assignment we can construct the encompassing tree.

5 Vertex-Colored Forests

We present a constructive proof for Theorem 3. Our proof relies on a recursive scheme of Hurtado et al. [11] (that constructs a vertex-colored encompassing tree without any degree bound), which we briefly recall here. Assume that we are given a vertex colored planar straight line forest with k components.

> **(H)** Choose a vertex a_0 on the convex hull of G. Repeat until G is connected:
> Let A be the component of G containing a_0 and let $B = G - A$. Find a vertex-edge pair (u, vw) such that $u \in A$ and u sees an entire edge vw in a component of B (Hurtado et al. [11] show that such a pair exists). Since v and w have different colors, we can augment G with the edge uv or uw, thus reducing the number of components.

We use the same recursive scheme, but we choose the pairs (u, vw) more carefully. If $u \in A$ sees an edge $vw \subset B$ and we augment G with uv or uw, then we say that u *docks* to vw and u is a *docking vertex*. We control the increase in the degree of vertices by maintaining the following property:

> ($\star$) Every vertex is a docking vertex in at most one iteration.

This guarantees that the degree of any vertex c increases by at most one in iterations where c belongs to component A, and by at most one at the iteration when we connect the component of c to A. The degree of any vertex x increases by at most two in total.

Notice that the recursive scheme (H) uses the fact that the edges of B are not monochromatic, but no such assumption is necessary about A. We may add dummy edges to A, even monochromatic edges, and the scheme (H) still works.

Component $A \subset G$, when augmented with dummy edges, may have several faces. We may search for pairs (u, vw) in each face of A that contains a component of B, independently, by a visibility sweep algorithm (cf., Section 6). The dummy edges will be essential in simplifying the visibility sweeps.

Stem vertices. Consider a PSLG G, let A be a *connected* component of G touching the convex hull of G and let $B = G - A$. The PSLG A partitions the plane into disjoint faces (there is one face if and only if A is a tree). We choose a *stem vertex* $a(F)$ for every face F of A: The stem vertex of the outer face is a vertex on the convex hull of G. The stem vertex of a bounded face is a *convex* vertex of ∂A.

We define two operations to augment A. Both are operations relative to a face F of A. Our first operation is the augmenting step from the recursive scheme (H) with a twist: For a pair (u, vw), we add both edges uv and uw to the graph independently of the vertex colors. This operation partitions a face F into a triangle uvw and $F \setminus uvw$.

> $\texttt{Add}_F(A, B, u, vw)$. Precondition: F is a face of A. $u \in A$ is a vertex of ∂F. vw is an edge of B and u sees the entire edge vw. Let $B(vw)$ be the component of B containing vw.
> Operation: Let $A' := A \cup \{uv, uw\} \cup B(vw)$.

The second operation adds a dummy edge d_1d_2 between two vertices of A and splits a face F. Since we want d_1 to be the stem vertex of one of the new faces, say F_1, and a stem vertex of a bounded face F_1 is convex vertex of ∂F_1, we need to impose conditions on the angles at d_1.

> $\texttt{Split}_F(A, B, d_1, d_2)$. Precondition: F is a face of A. d_1 and d_2 are non-consecutive vertices of ∂F. d_1d_2 does not cross $A \cup B$. d_1d_2 splits F into

two faces F_1 and F_2 such that $a(F)$ is incident to F_1 and d_1 is a convex vertex of ∂F_2.
Operation: Let $A' = A \cup \{d_1 d_2\}$. Let $a(F_1) := a(F)$ and $a(F_2) := d_1$

We augment A by these two operations until $G \subset A$. In the next section, we present algorithms that examine every face that contains a component of B and apply one of the two operations. Property ($\star$) requires that $\texttt{Add}(A, B, u, vw)$ is applied at most once for every vertex u. A vertex $u \in A$ may be a vertex of several faces of A (in fact, u may be a vertex of the same face several times since the boundary of a face is a *weakly* simple polygon). We call these the *occurrences* of u, each occurrence corresponds to an angular domain between consecutive edges incident to u. We impose two more preconditions for the two operations to ensure that convex nonstem occurrences of u are not used in operations $\texttt{Add}_F(A, B, u, vw)$.

(1) In $\texttt{Add}_F(A, B, u, vw)$, u is either the stem vertex or a reflex vertex of ∂F;

(2) in $\texttt{Split}_F(A, F, d_1, d_2)$, d_1 is either the stem vertex or a reflex vertex of ∂F.

These conditions guarantee that if u is a convex nonstem vertex of ∂F, then u never docks to any edge lying in the interior of F. We replace the property ($\star$) by two properties that impose conditions relative to a face F of A only.

($\heartsuit$) After applying $\texttt{Add}_F(A, B, u, vw)$, vertex u never docks to any edge in the interior of F.

($\diamondsuit$) After applying $\texttt{Split}_F(A, F, d_1, d_2)$, vertex d_1 never docks to any edge in the interior of F_1.

Property ($\heartsuit$) guarantees that if $u \in A$ docks to an edge vw in F, then it never docks again in any face of A. Property ($\diamondsuit$) ensures that every vertex $u \in A$ has at most one occurrence $u \in \partial F$ that may dock to an edge $vw \subset B$ (lying in the corresponding face F) after any number of recursive $\texttt{Split}$s. We have reduced the problem of maintaining ($\star$) to problems in individual faces of A. To prove Theorem 3, it is enough to show that in every face F containing a component of B, one can apply either $\texttt{Add}_F(A, B, u, vw)$ or $\texttt{Split}_F(A, F, d_1, d_2)$ satisfying all preconditions and maintaining properties ($\heartsuit$) and ($\diamondsuit$).

The easiest way to guarantee that a reflex or stem vertex of a face F never docks to an edge of B in $\text{int}(B)$ is showing that $u \in A$ does not see an entire edge of B in F. For vertices on the outer face of A and stem vertices of bounded faces, we will use this argument in the next section. For reflex vertices of bounded faces, we apply a more complex argument.

6 Visibility Sweeps

We recursively run a visibility-sweep algorithm in each face F of A that contains a component of B. The algorithm either docks a vertex of ∂F to an edge of B or splits F into two faces. We apply one algorithm for the outer face and another one for bounded faces.

6.1 Algorithm for the Outer Face

Consider the outer face F_0 of the connected PSLG A. Let the stem vertex $a(F_0)$ be the vertex $a_0 \in A$ such that a_o lies on $\text{conv}(A \cup B)$. For the sake of our arguments, we split a_0 into two distinct occurrences a_0 and a_{-1} and we connect them by an infinitesimal edge such that $a_{-1}a_0$ defines a tangent of $\text{conv}(A \cup B)$, and a_{-1} is the clockwise neighbor of a_0 along ∂A. Both a_{-1} and a_0 are reflex vertices of ∂F_0.

Let V_0 be a visibility vector along the ray $\overrightarrow{a_{-1}a_0}$, such that a_0 is the tail of V_0 and its head is at infinity. We apply the following visibility sweep algorithm.

ALGORITHM 7. *Input:* A, B, F, a_0, *and* V_0. *Initialize* $i := 0$, $j := 0$, $x := 0$, *and* $V := V_0$.

1. *Repeat:*
   (a) *Rotate the visibility vector* V *counter-clockwise around* a_i *until it hits a vertex* $c \in A \cup B$.
   (b) *If* $c \in A$, *adjacent to* a_i *along* ∂F, *and a reflex vertex of* ∂F, *then let* $a_{i+1} := c$, $i := i + 1$.
   (c) *If* $c \in A$, *adjacent to* a_i *along* ∂F, *and a convex vertex of* ∂F *then* $\texttt{stop}$.
   (d) *If* $c \in A$ *but not adjacent to* a_i *along* ∂F *then* $\texttt{Split}_F(A, B, a_i, c)$ *and* $\texttt{stop}$.
   (e) *If* $c \in B$ *such that all incident edges are on the left side of* V, *then let* $b_{j+1} := c$ *and* $j := j + 1$.
   (f) *If* $c \in B$ *such that* B *has an incident edge on the right side of* V *then* $\texttt{Add}_F(A, B, a_i, b_j c)$ *and* $\texttt{stop}$.

Analysis. During Algorithm 7, the vertices $a_0, a_1, \ldots, a_i$ at the tail of the visibility vector V form a reflex chain of ∂F. The head of V sweeps either infinity or edges of B: Indeed, if V hits a vertex $c \in A$ (step 1d), then the algorithm splits the outer face into two faces along the diagonal $a_i c$ and terminates. Since the head of V never sweeps along an edge of A, Algorithm 7 does not terminate with step 1c. (This step will be functional when applied to bounded faces in Subsection 6.2.) If V ever hits a vertex of B, then its head keeps sweeping along edges of B until the algorithm terminates.

We still need to show that Algorithm 7 terminates. Assume, by contradiction, that it does not terminate. Since V always rotates counter-clockwise in the outer face F_0 and its tail travels through consecutive vertices of of ∂F_0, V must return to its original position V_0. ∂F_0 must be the convex hull of A because the vertices $a_0, a_1, a_2, \ldots$ lie on a reflex chain of ∂F_0. Since B has a component in $\text{int}(F_0)$, V hits a vertex of B at some step. After this step, the head of V keeps sweeping along edges of B (but never reaches a left endpoint of an edge of B, otherwise it would terminate with step 1f). Therefore, V cannot return to its initial position, where V would point to infinity.

We conclude that Algorithm 7 terminates with step 1d or 1f for some $i = \ell$. If it terminates with step 1d, then the preconditions of $\texttt{Split}_F(A, B, a_\ell, c)$ are satisfied since a_i sees c. If the algorithm terminates with step 1f, we show that the preconditions of $\texttt{Add}_F(A, B, a_\ell, b_j c)$ are satisfied, that is, a_i sees $b_j c \subset B$. In this last step, V hits the left endpoint c of an edge of B. V must have hit the right endpoint of this edge for some $i = \ell'$. Since V did not hit any right endpoint before c, so b_jc is in indeed an edge of B.

The viability vector V sweeps through the polygon

$$W = (a_{\ell'}, a_{\ell'+1}, \ldots, a_i, c, b_j)\,.$$

Hence, the interior of W is disjoint from the PSLG $A \cup B$. Since the sequence $(a_0, a_1, \ldots, a_\ell)$ is a reflex chain of ∂W, we conclude that a_ℓ sees the entire edge b_jc within $W \subset F_0$.

Let A' augmented graph and let $B' = G - A'$ denote its complement after Algorithms 7. Let F_0' denote the outer face of A'. We establish (♢) and (♡) by the following proposition.

PROPOSITION 8. *No reflex occurrence of a vertex a_i, $i = 0, 1, \ldots, \ell$, sees an entire edge of B' in the interior of F_0'.*

PROOF. Every a_i, , $i = 0, 1, \ldots, \ell$, is a reflex vertex of F_0. Let a_i^- and a_i^+ be the vertices along ∂F_0 preceding and following a_i (e.g., we have $a_i^- = a_{i-1}$).

If Algorithm 7 terminates with step 1f then a_ℓ occurs (at least) twice along the weakly simple polygon $\partial F_0'$: The two occurrences correspond to the angles $\angle a_\ell^- a_\ell b_j$ and $\angle c a_\ell a_\ell^+$. Observe that $\angle c a_\ell a_\ell^+$ is always convex: This clearly holds if $\ell = 0$; and if $\ell > 0$ then the sweep vector V passed the line $a_\ell^- a_\ell$ in the step 1b where i was increased to ℓ. In this case the only possible *reflex* occurrence of a_ℓ corresponds to $\angle a_\ell^- a_\ell b_j$.

Consider the polygon Π swept by the visibility vector V between its initial and last positions, V_0 and $a_\ell c$. We show that a_i cannot see points outside the (closure of) this polygon, and so it cannot see any entire edge of B'. The boundary of polygon Π include the first and last positions of V: V_0 and $a_\ell c$; the tail of V sweeps through a reflex chain $(a_0, a_1, \ldots, a_\ell)$; the head of V sweeps through a staircase chain $(b_0, d_0, b_1, d_1, \ldots d_{j-1}, b_j, c)$. Here, every $b_k d_k$, $k = 0, 1, \ldots, j-1$, lies along an edge of B, where b_k is the left endpoint and d_k is a (relative) interior point of the edge. Every $d_k b_{k+1}$, $k = 0, 1, \ldots, j-1$, lies along a visibility vector $a_{h(k)} d_k$ for some h, $0 \le h \le \ell$. By contradiction, suppose that a_i sees a point $p \in A \cup B$ outside Π: Necessarily, segment $a_i p$ crosses a segment $d_m b_{m+1}$, but any curve within Π from a_i to $d_m b_{m+1}$ have to cross $a_{h(m)} b_{m+1}$. Hence, $a_i p$ cannot be a straight line segment because it crosses $d_m b_{m+1}$ and $a_{h(m)} b_{m+1}$, which are collinear. □

Proposition 8 implies that *reflex occurrences* of a_i, $0 \le i \le \ell$, will not dock to any edge of B'. If Algorithm 7 applies the operation $\texttt{Add}_F(A, B, a_\ell, b_j c)$, then a_ℓ never docks to any edge in the interior of F_0, which proves (♡). If it applies $\texttt{Split}_F(A, B, a_\ell, c)$ then the reflex occurrence of no a_i along $\partial F_0'$ docks to any edge in the interior of F_0', which proves property (♢).

6.2 Algorithm for Bounded Faces

Consider a bounded face F of A that contains a component of B, and let $a = a(F)$ denote its stem vertex. We assumed that a is a *convex* vertex of ∂F. Let q be a vertex adjacent to a along ∂F in counter-clockwise direction. The initial position V_0 of the sweep vector is along the ray $\overrightarrow{aq}$. Let r be the point on the boundary of F where the ray $\overrightarrow{aq}$ exits the face F. Segment ar divides F into a left subface F^+ and right subface F^-. The right subface F^- may be empty if $r = q$ but F^+ is always nonempty. The head of the *visibility* vector V_0 is either at the relative interior of an edge of B, or it is r. If the head of V_0 is a point $b^* \in B$, then we add an artificial vertex b^* to B. We apply an algorithm that either splits F into two faces, or finds a reflex or stem vertex along ∂F that sees an entire edge of B lying in the interior of F. We compile this algorithm by applying Algorithm 7 in the subfaces F^+ and F^- as follows.

ALGORITHM 9. *Input: $A, B, F, a(F)$, and V_0.*

1. *Run Algorithm 7 on the left subface F^+ with initial vector V_0. If it applies the* `Add` *or* `Split` *operations but not $\texttt{Add}_{F^+}(A, B, a_i, b^* c^+)$ involving the artificial vertex b^*, then we apply the same operation for F and* `stop`.

2. *Run Algorithm 7 on the mirror image*[2] *of the right subface F^- with initial vector V_0. If it applies the* `Add` *or* `Split` *operations but not $\texttt{Add}_{F^-}(A, B, g_k, b^* c^-)$ involving the artificial vertex B^*, then we apply the same operation for F and* `stop`.

3. *If the two calls to Algorithm 7 in the previous steps apply $\texttt{Add}_{F^+}(A, B, a_i, b^* c^+)$ and $\texttt{Add}_{F^-}(A, B, g_k, b^* c^-)$, respectively, then:*
   If a_i sees the entire edge $c^- c^+$, $\texttt{Add}_F(A, B, a_i, c^- c^+)$, otherwise $\texttt{Add}_F(A, B, g_k, c^- c^+)$.

Analysis. Step 1 Algorithm 7, and the tail of the visibility vector V sweeps along a reflex chain $(a_0 = a(F), a_1, \ldots, a_i)$ of ∂F^+. If step 2 is executed, then in a second call to Algorithm 7 the tail of V sweeps along a reflex chain $(g_0 = a(F), g_1, \ldots, g_k)$ of ∂F^-. Note that V rotates in counterclockwise and clockwise directions in the two calls, and so the head of V never hits ar, the boundary between the subcells F^+ and F^-.

If either call to Algorithm 7 applies a `Split` or an `Add` operation within F^+ or F^-, resp., then it is a valid operation in F, too. A `Split` operation is invoked if a visibility vector V connects two nonadjacent vertices along ∂F. An `Add` operation docks a stem or reflex vertex along ∂F to an edge of B swept by the head of the visibility vector. A call to Algorithm 7 for a bounded face F^+ or F^- can differ from a call for the outer face F_0 in two aspects:

1., The head of V may sweep along edges of A (i.e., edges of ∂F). If V hits a vertex of B, then the head of V keeps seeping edges of V until the call to Algorithm 7 terminates. So V can only sweep along edges of A if $V_0 = \overrightarrow{ar}$ and V does not hit any vertex of B. In this case, the call to Algorithm 7 terminates with its step 1c, and this implies $\mathrm{int}(F^+) \cap B = \emptyset$. Since we assume that F contains a component of B, F^- must be nonempty (that is, $\mathrm{int}(F^-) \cap B \neq \emptyset$), and so the second call to Algorithm 7 applies a `Split` or an `Add` operation in F^-.

2, If a call to Algorithm 7 applies the `Add` operation, then it docks a vertex of ∂F^+ (resp., ∂F^-) to an edge $bc \subset B$ swept by the head of the visibility vector V. Edge bc lies in F^+ (resp., F^-) because V rotates counter-clockwise. We only have to worry about the case that ar hits an edge of B and an `Add` operation dock a vertex of ∂F to an edge incident to the artificial vertex b^*. Let $c^- c^+ \subset B$ be the edge containing b^* such that $c^- \in \mathrm{int}(F^-)$ and $c^+ \in \mathrm{int}(F^+)$. Suppose

[2] The orientations clockwise and counter-clockwise, and the sides left and right are exchanged in the mirror image.

that the first call to Algorithm 7 applies $\texttt{Add}_{F^+}(A, B, a_i, b^*c^+)$ for some i. In this case, step 2 of Algorithm 9 is executed. Suppose that it applies an operation $\texttt{Add}_{F^-}(A, B, g_k, b^*c^-)$ for some k, involving vertex b^* (see Figure 4(l)).

Notice that the tail of V in the two calls to Algorithm 7 sweep along two reflex chains: $(a_0, a_1, \ldots, a_i)$ in F^+ and $g_0, g_1, \ldots, g_k$ in F^-. The visibility vector V sweeps the pseudo-triangle

$$(a_0, a_1, \ldots, a_i, c^+, c^-, g_k, g_{k-1}, \ldots, g_1),$$

where $a_0 = g_0 = a(F)$ is the stem vertex of F. In this pseudo-triangle, either a_i or g_k sees the entire edge c^+c^-, which is docked in step 3 of Algorithm 9.

In order to establish properties ($\diamondsuit$) and ($\heartsuit$), we can use Proposition 8 in subfaces F^+ and F^-. It immediately implies the following weaker proposition for the reflex vertices of ∂F.

PROPOSITION 10. *A reflex occurrence of a vertex a_ℓ, $\ell = 1, \ldots, i$, along ∂F does not see an entire edge of B' within F^+. (Similarly, a reflex occurrence of a vertex g_ℓ, $\ell = 1, 2, \ldots, k$, along ∂F does not see an entire edge of B' within F^-.)* □

Proposition 10 does not speak about the *convex* stem vertex a; furthermore is leaves open the possibility that a vertex a_ℓ, $\ell = 1, \ldots, i$, along the reflex chain of ∂F^+ sees an entire edge of B lying partly in F^- (and similarly, a vertex g_ℓ along the reflex chain of ∂F^- may see an entire edge of B in F^+).

Consider the (convex) stem vertex a of F. Step 1 of Algorithm 9 may apply $\texttt{Split}_F(A, F, a, d_2)$ for some $d_2 \in \partial F$ and split F into two faces F_1 and F_2, where a is the stem vertex of both F_1 and F_2. Vector V has swept through points visible by a in the face F_1 containing V_0, and so the occurrence of a corresponding to F_1 is never a docking vertex (otherwise $\texttt{Add}_F(A, F, a, vw)$ would have been applied instead of $\texttt{Split}_F(A, F, a, d_2)$). This proves ($\diamondsuit$) for a. In step 1 of Algorithm 9, a may dock to an edge $b_jc^+ \subset \text{int}(F^+)$. The stem vertex a of ∂F is split into two convex occurrences in the resulting face F'. Vector V has swept through the points visible by one occurrence, this should be the stem vertex of F'. The other occurrence may still see an entire edge of B in F', but it is a convex nonstem vertex of $\partial F'$. So property ($\heartsuit$) is maintained for a.

Now assume that the reflex vertex a_i, $i > 0$, (resp., g_k, $k > 0$) along the reflex chains of ∂F^+ (resp., ∂F^-) docks to an edge of B, and its reflex occurrence still sees an entire edge $v'w' \subset B'$ lying in the interior of F. By Proposition 10, $v'w' \not\subset \text{int}(F^+)$, and so a_iv' or a_iw' crosses the initial visibility vector V_0 on the boundary between F^+ and F^-. Since Algorithm 9 docks a_i, $i > 0$ (or d_k, $k > 0$), we know that the stem vertex $a_0 = g_0 = a(F)$ does not see any entire edge of B nor a nonadjacent vertex of ∂A. Therefore, in every face F' that intersects V_0 in all subsequent iterations, the stem vertex is $a(F') = a(F)$, and the initial visibility vector is the same V_0. This implies that our algorithm never docks a_i (resp., d_k) to $v'w'$ on the other side of V_0. This completes the proof of ($\heartsuit$) for reflex vertices of ∂F.

7 Arbitrary Vertex-Colored PSLGs

By Theorem 3, for a vertex-colored planar straight line forest without singleton components, there is a vertex-colored encompassing tree G' such that the degree of every vertex increases by at most two. Our proof extend to an arbitrary vertex-colored PSLG G without singleton components, if we can find a stem vertex for every face of the G such that every vertex is the stem or reflex vertex of at most two faces.

CONJECTURE 11. *We are given a connected PSLG G and a vertices v_0 incident to its outer face. There is a vertex-face assignment with the following properties: (i) Every vertex is assigned to at most two faces; (ii) v_0 is assigned to at most one face; (iii) Every face is assigned to all its reflex vertices; (iv) Every bounded face is assigned to one of its convex vertices.*

For an input of a vertex colored PSLG, our algorithm compute iteratively an encompassing graph. We choose an initial component $A \subset G$, and choose a convex stem vertex for each bounded face of A by the assignment in Conjecture 11. We run our Algorithm 7 or 9 in each face of A, independently. Consider a component $C \subset G$ along a face F of A. When a vertex $u \in A$ docks to an edge vw of a component $C \subset \text{int}(F)$ and we augment G with, say, edge uv, then the degree of $v = v_0(C)$ increases by one. We set $a := A \cup \{uv, uw\} \cup C$, and we choose a convex stem vertex for each bounded face of C according to the assignment of Conjecture 11 for C and $v_0(C)$.. Once we have $C \subset A$, our algorithm guarantees that we augment the degree of every reflex or stem occurrence of vertex of C by at most one, and we do not increase the degree of nonstem convex vertices.

Note that Conjecture 11 does not take the vertex colors into account, and it is enough to find an assignment independently for each connected component of G.

8 Conclusion

We presented an optimal $O(n \log n)$ time algorithm for computing a pointed binary encompassing tree for disjoint line segments in the plane. The edges incident to each segment endpoint lie on one side of the segment. We defined a tunnel graph on the convex partition of the complement of the segments, and showed that there is a partition for which a connected tunnel graph exists. We have shown that every vertex-colored planar straight line matching has a vertex-colored encompassing graph of maximum degree three. A couple of related questions remain open:

• Does every vertex-colored planar straight line matching have a pointed and vertex-colored encompassing tree of maximum degree three?

• Is there such an encompassing tree with the stronger version of pointedness, where we require that at every vertex all incident edges lie in a half-plane defined by the incident input edge?

• Does every vertex-colored PSLG G have an encompassing tree G' such that for every vertex v, we have $\deg_G(v) \le \deg_{G'}(v) \le \deg_G(v) + 2$?

Acknowledgment

We thank David Rappaport for insights into visibility sweep algorithms, and for many useful comments on this paper.

Figure 4. The steps of our algorithm for a vertex-colored planar straight line forest. Edges of G are fat, dummy edges of A are thin, stem vertices of faces containing parts of B are marked with small arrows, regions swept by the visibility vector V are grey.

References

[1] P. K. Agarwal, J. Basch, L. J. Guibas, J. Hershberger, and L. Zhang, Deformable free space tilings for kinetic collision detection, in *Algorithmic and Computational Robotics: New Directions, WAFR'00*, AK Peters, Boston, 2001, pp. 83–96.

[2] J. Akiyama & J. Urrutia, Simple alternating path problem, *Discrete Math.* **84** (1990), 101–103.

[3] M. Bern, Triangulations and mesh generation, in Handbook of Discrete and Computational Geometry, CRC Press, 2004, pp. 563–582.

[4] O. Aichholzer, M. Hoffmann, B. Speckmann, and Cs. D. Tóth, Degree bounds for constrained pseudo-triangulations, in: *Proc. 15th Canadian Conf. Comput. Geom.*, 2003, pp. 155–158.

[5] M. Grantson, H. Meijer, and D. Rappaport. Bichromatic minimum spanning trees, in *Abstracts of 21st Euro. Workshop Comput. Geom.*, 2005, pp. 199–202.

[6] P. Bose, M. E. Houle, and G.T. Toussaint, Every set of disjoint line segments admits a binary tree, *Discrete Comput Geom.* **26** (2001), 387–410.

[7] P. Bose and G. T. Toussaint, Growing a tree from its branches, *J. Algorithms* **19** (1995), 86–103.

[8] M. Hoffmann, B. Speckmann, and Cs. D. Tóth, Pointed binary encompassing trees, in *Proc. 9th SWAT*, vol. 3111 of LNCS, Springer-Verlag, 2004, pp. 442–454.

[9] M. Hoffmann and Cs. D. Tóth, Alternating paths through disjoint line segments, *Inf. Proc. Letts.* **87** (2003), 287–294.

[10] M. Hoffmann and Cs. D. Tóth, Segment endpoint visibility graphs are Hamiltonian, *Comput. Geom. Theory Appl.* **26** (2003), 47–68.

[11] F. Hurtado, M. Kano, D. Rappaport, and Cs. D. Tóth, Encompassing colored crossing-free geometric graphs, in Proc. *16th Canadian Conf. Comput. Geom.*, 2004, pp. 48–52.

[12] A. Kaneko, On the maximum degree of bipartite embeddings of trees in the plane, in *Discrete and Computational Geometry, JCDCG'98*, vol. 1763 of LNCS, Springer, 2000, pp. 166–171.

[13] A. Kaneko and M. Kano, Discrete geometry on red and blue points in the plane—a survey, in *Discrete and Computational Geometry, The Goodman-Pollack Festschrift*, Springer-Verlag, 2003, pp. 551–570.

[14] A. Kaneko and M. Kano, On paths in a complete bipartite geometric graph, in *Discrete and Computational Geometry, JCDCG'00*, vol. 2098 of LNCS, Springer-Verlag, 2001, pp 187–191.

[15] A. Kaneko, M. Kano, and K. Yoshimoto, Alternating Hamiltonian cycles with minimum number of crossings in the plane, *Internat. J. Comput. Geom. Appl.* **10** (2000), 73–78.

[16] L. Kettner, D. Kirkpatrick, A. Mantler, J. Snoeyink, B. Speckmann, and F. Takeuchi, Tight degree bounds for pseudo-triangulations of points, *Comput. Geom.* **25** (2003), 1–12.

[17] D. Kirkpatrick and B. Speckmann, Kinetic maintenance of context-sensitive hierarchical representations for disjoint simple polygons, in *Proc. 18th ACM Sympos. Comput. Geom.*, ACM Press, 2002, pp. 179–188.

[18] D. T. Lee and A. K. Lin, Generalized Delaunay triangulations for planar graphs, *Discrete Comput. Geom.* **1** (1986), 201–217.

[19] J. Pach and E. Rivera-Campo, On circumscribing polygons for line segments, *Comput. Geom. Theory Appl.* **10** (1998), 121–124.

[20] B. Speckmann and Cs. D. Tóth, Allocating vertex π-guards in simple polygons, *Discrete Comput. Geom.* **33** (2) (2005), 345-364.

[21] I. Streinu, A combinatorial approach to planar noncolliding robot arm motion planning, *41st IEEE Sympos. Foundations Comp. Sci.*, 2000, pp. 443–453.