Pointed Binary Encompassing Trees: Simple and Optimal

Michael Hoffmann* Csaba D. Tóth†

Abstract

For n disjoint line segments in the plane we construct in optimal $O(n \log n)$ time an encompassing tree of maximal degree three such that every vertex is *pointed*. Moreover, at every segment endpoint all incident edges lie in a halfplane defined by the incident input segment.

1 Introduction

Interconnection graphs of disjoint line segments in the plane are fundamental structures in computational geometry. Often more complex objects can be modeled by their boundary segments or polygons. One particularly well-studied example is a crossing-free spanning graph: the *encompassing graph* for disjoint line segments in the plane is a connected planar straight line graph (PSLG) whose vertices are the segment endpoints and that contains every input segment as an edge. One well-known example of encompassing graphs are constrained (Delaunay) triangulations [9].

Bose et al. [4, 3] showed that any finite set of disjoint line segments in the plane admits an encompassing tree of maximum degree *three*. Moreover, they gave an $O(n \log n)$ time algorithm to construct such an encompassing tree for n given segments. Both the degree bound and the runtime are best possible (the latter in the algebraic computation tree model).

Hoffmann, Speckmann, and Tóth [5] extended the result of Bose et al. by showing that for n disjoint segments in the plane a *pointed* binary encompassing tree can be constructed in $O(n^{4/3} \log n)$ time. A PSLG is *pointed* iff for every vertex v all edges incident to v lie in a halfplane whose boundary contains v.

Here, we improve this result in several 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. As an additional benefit, the presented algorithm is also considerably simpler (to understand and to implement) than the existing approach.

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

*Theoretical Computer Science, ETH Zürich, hoffmann@inf.ethz.ch

†Department of Mathematics, MIT, Cambridge, toth@math.mit.edu

Motivation *Pointed* PSLGs are closely related to minimum pseudo-triangulations, which have numerous applications in motion planning [11], kinetic data structures [8], collision detection [1], and guarding [10]. Streinu [11] 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 degree pseudo-triangulation of a set of points in the plane [7]. 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. [2].

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 [6].

2 Definitions

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

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 exterior.

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.

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 the weakly simple polygon ∂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.

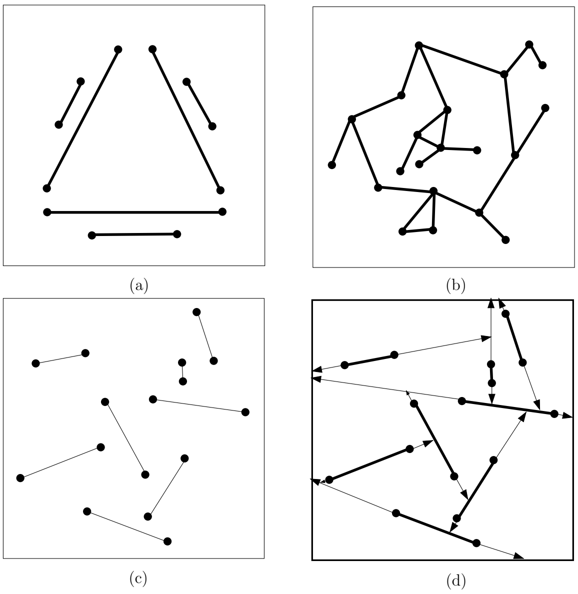


Figure 1: Six segments not admitting an encompassing path (a), a connected PSLG with 4 faces including the outer face (b), disjoint segments (c), and their convex partition (d).

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 iff 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 iff it is a tree.

Theorem 2 *For any set S of n disjoint line segments, we can construct in $O(n \log n)$ time a convex partition $P(S)$ and an assignment τ such that the tunnel graph $T(S, P(S), \tau)$ is a tree.*

We note that the choice of the convex partition is important in Theorem 2: Figure 2(d) shows seven disjoint line segments and a convex partition such that there is no assignment for which the tunnel graph is connected. We obtain Theorem 1 as a corollary of Theorem 2.

Proof of Theorem 1. Consider a partition $P(S)$ and an assignment τ provided by Theorem 2. 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.

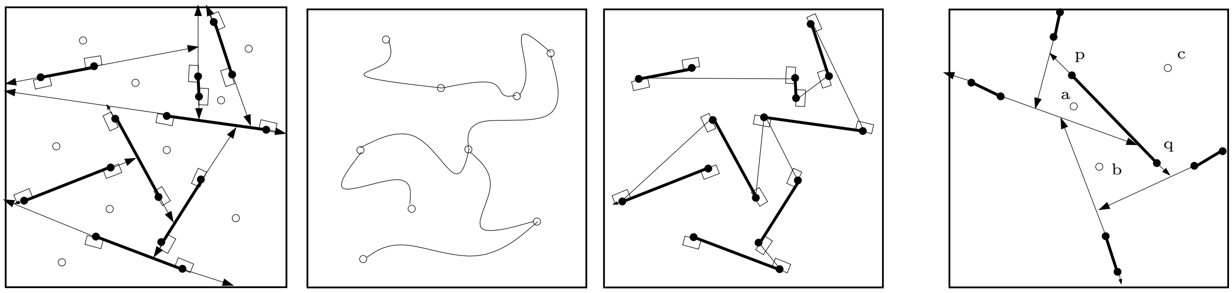
The resulting graph is clearly a PSLG that encompasses the input segments. The maximal degree is three because we add at most two new edges at every segment endpoint. It remains to prove connectivity. Let p and r be two segment endpoints. We know that the tunnel graph is connected, so there is an alternating 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 encompassing graph. □

4 Constructing the Convex Partition

This section is devoted to the proof of Theorem 2. Given n disjoint line segments in the plane, we partition the free space around the segments into $n+1$ convex cells and we assign an incident cell to every segment endpoint in $O(n \log n)$ time.

Let R be a bounding box of the input segments. We construct a convex partition of the free space in two phase line sweep algorithm. In the first phase, we apply 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, then an arbitrary one continues and the other one ends.

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 endpoint along ∂C_0. Let p_i (q_i) denote the left (right) endpoint of s_i. We extend every input segment s_i, $i = 1, 2, \ldots, n$, in this order, beyond its left endpoint p_i until the extension hits another segment, another extension, or the boundary of R. Let γ_i denote the left extension of s_i. Every segment γ_i recursively partitions a cell of our cell complex into two subcells. Notice that all left extensions γ_i can be constructed in a single right-to-left sweep which gives priority to the segment of smaller index whenever two



(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).

extensions meet. Thus we can compute a convex partition P by two line sweeps in $O(n \log n)$ time.

For constructing the assignment τ, we assume that all right extensions are in place, and we insert the left extensions one by one (even though we have pre-computed all left extensions in a single sweep). We define the assignment τ on the endpoints of s_i, $i = 1, 2, \ldots, n$, as soon as γ_i has been inserted. When the first $i-1$ left extensions have been inserted, we have a partition P_{i-1} into i cells and a partial assignment τ_{i-1} on the endpoints of the first $i-1$ segments. P_{i-1} and τ_{i-1} define a tunnel graph T_{i-1} on i nodes. We choose the assignment at the endpoints of s_i inductively such that T_i (a graph with $i+1$ nodes) remains connected.

Assume that γ_i splits a cell C_i of P_{i-1} into two cells $C'_i, C''_i \in P_i$. The node $v(C) \in T_{i-1}$ corresponding to cell C is split into two nodes C' and C'', which lie 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 $p_i \in \gamma_i$. 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. We can always assign p_1 to C'_i or C''_i, whichever lies in the other component of T'_{i-1} as $\tau(q_i)$, thus ensuring that T_i is a tree.

We have shown that there exists an assignment $\tau = \tau_n$ for which the tunnel graph $T = T_n$ is connected. It remains to prove that such an assignment can be computed in $O(n \log n)$ time. That is, we need to decide efficiently whether two cells are in the same connected component of the current tunnel graph T_i.

Data structure. For each cell C of P_{i-1}, we maintain a doubly linked list of all segment endpoints and vertices along ∂C. The assignments τ_i carries one bit information for each segment endpoint r: It assigns r to the cell lying *below* or *above* r. We can insert a splitting segment γ_i by splitting the doubly connected list of of C_i into C'_i and C''_i in constant time. We also note an interval $g(v) \subset [1, n]$ for each vertex v of the right extension tree such that the descendants of v contain every left segment endpoint p_j, $j \in g(v)$. We maintain a *coloring* on the segments and their left and right extensions: Every input segment and every right extension is *blue*. The color of right 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. We assign 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 3 *Choosing $\tau(p_i)$ by the above rule maintains the connectivity of T_i*

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.

By a case analysis, we can verify 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. (1) If γ_i is red then β_i is x-monotone and its two branches pass

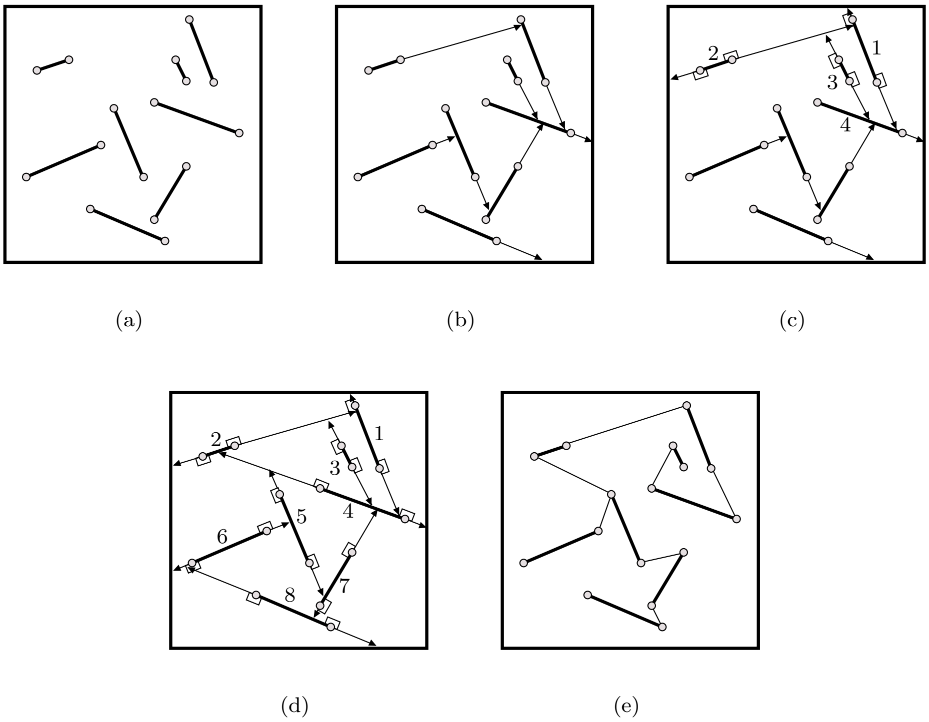


Figure 3: Constructing the partition: First all right extensions (b), then the left extensions are inserted one by one (c) and (d), and from the final partition together with the assignment we can construct the encompassing tree.

through right endpoints only, so p_i is the only vertex that might be assigned to the region below β_i. (2) Suppose that γ_i is blue: The left branch of β_i starts out x-monotone decreasing, then it hits a segment or a right extension and turns back in x-monotone increasing direction. Let A_i be the region not adjacent to the left side of R. A_i is an x-monotone region. (2a) If γ_i hits a segment s_j, $j > i$, or its right extension, then A_i must be below s_i. We know that B_i is above q_i and A_i is below p_i. (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 $v_i \in s_i$. $\square$

Acknowledgments We thank Bettina Speckmann for several helpful discussions.

References

[1] P. K. Agarwal, J. Basch, L. J. Guibas, J. Hershberger, and L. Zhang, Deformable free space tilings for kinetic collision detection, in *Proc. 4th WAFR*, 2001, 83–96.

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

[3] 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.

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

[5] 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.

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

[7] 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.

[8] D. Kirkpatrick and B. Speckmann, Kinetic maintenance of context-sensitive hierarchical representations for disjoint simple polygons, in *Proc. 18th SoCG*, 2002, pp. 179–188.

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

[10] B. Speckmann and Cs. D. Tóth, Allocating vertex π-guards in simple polygons, *14th SODA*, 2003, pp. 109–118.

[11] I. Streinu, A combinatorial approach to planar non-colliding robot arm motion planning, *41st FOCS*, 2000, pp. 443–453.