Counting the Number of Crossings in Geometric Graphs

Frank Duque \(^*\)
Instituto de Matemáticas UNAM
Instituto de Física UASLP

Ruy Fabila-Monroy \(^†\)
Departamento de Matemáticas CINVESTAV

César Hernández-Vélez
Facultad de Ciencias UASLP

Carlos Hidalgo-Toscano \(^†\)
Departamento de Matemáticas CINVESTAV

April 26, 2019

Abstract

A geometric graph is a graph whose vertices are points in general position in the plane and its edges are straight line segments joining these points. In this paper we give an \(O(n^2 \log n)\) algorithm to compute the number of pairs of edges that cross in a geometric graph on \(n\) points. For layered, and convex geometric graphs the algorithm takes \(O(n^2)\) time.

1 Introduction

A geometric graph is a graph whose vertices are points in general position in the plane; and its edges are straight line segments joining these points. A pair of edges of a geometric graph cross if they intersect in their interior; the number of crossings of a geometric graph is the number of pairs of its edges that cross.

Let \(G := (V, E)\) be a geometric graph on \(n\) vertices; and let \(H\) be a graph. We say that \(G\) is a rectilinear drawing of \(H\) if \(G\) and \(H\) are isomorphic as graphs. The rectilinear crossing number of \(H\) is the minimum number of crossings that appear in all its rectilinear drawings. We abuse notation and use \(\text{cr}(H)\) and \(\text{cr}(G)\) to denote the rectilinear crossing number of \(H\) and the number of crossings of \(G\), respectively.

Computing the rectilinear crossing number of the complete graph \(K_n\) on \(n\) vertices is an important and well known problem in Combinatorial Geometry. The current best bounds on \(\text{cr}(K_n)\) are

\[
0.379972 \left(\frac{n}{4}\right) < \text{cr}(K_n) < 0.380473 \left(\frac{n}{4}\right) + \Theta(n^3).
\]

\(^*\)Partially supported by FORDECYT 265667 (Mexico)
\(^†\)Partially supported by Conacyt of Mexico grant 253261.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734922.
The lower bound is due to Ábrego, Fernández-Merchant, Leaños and Salazar \cite{1}. The upper bound is due to Fabila-Monroy and López \cite{12}. In an upcoming paper, Aichholzer, Duque, Fabila-Monroy, García-Quintero and Hidalgo-Toscano \cite{3} have further improved the upper bound to
\[
\overline{\tau}(K_n) < 0.3804493 \binom{n}{4} + \Theta(n^3).
\]

For more information on crossing numbers (rectilinear or other variants) we recommend the survey by Schaefer \cite{10}.

A notable property of the improvements of \cite{12, 3} on the upper bound of $\overline{\tau}(K_n)$ is that they rely on minimizing the crossing number of rectilinear drawings of $K_n$ for some particular value of $n$; this is done via heuristics that take a rectilinear drawing of $K_n$ and move its points in various ways; the aim is to decrease the number of crossings. In this approach it is instrumental that the computation of the number of crossings is done as fast as possible.

In this paper we present an algorithm to compute $\overline{\tau}(G)$ in $O(n^2 \log n)$ time. For layered graphs, and convex geometric graphs our algorithm runs in $O(n^2)$ time. For layered graphs, when $G$ has $\omega(n^2 / \log n)$ edges, this solves Problem 33 in \cite{6}. We hope that our algorithm will pave the way for finding new upper bounds on the rectilinear crossing number of various classes of graphs.

If $G$ has $\Theta(n^2)$ edges then $\overline{\tau}(G)$ is $\Theta(n^4)$. Thus, reporting the pairs of edges of $G$ that cross might take much more time than counting them. The problems of counting and reporting the intersections of a given set of $m$ line segments in which $k$ pairs of them intersect are historically important problems in Computational Geometry. The two problems are closely related as the ability to report grants the ability to count. The starting point for both problems is the 1976 paper by Shamos and Hoey \cite{17}; they studied various geometric intersection problems and left the segment intersection-counting and segment intersection-reporting problems as open.

For the segment intersection-reporting problem we have the following. The first non-trivial algorithm was given by Bentley and Ottmann \cite{5} in 1979; they gave an $O(m \log m + k \log m)$ time and $O(n + k)$ space algorithm. At roughly the same time Nievergelt and Preparata \cite{14} gave an algorithm with same time and space complexities. In 1981, Brown \cite{7} reduced the space requirement of the algorithm of Bentley and Ottmann to $O(m)$. In 1986, Chazelle \cite{8} gave an $O(m \log^2 m / \log \log m + k)$ time algorithm. This was the first algorithm whose time dependence on $k$ is linear. Independently, around 1989, Clarkson and Shor \cite{11}, and Mulmuley \cite{13} gave a randomized algorithm of $O(m \log m + k)$ expected time. The algorithm of \cite{11} takes $O(m)$ space and the algorithm of \cite{13} takes $O(m + k)$ space. In 1992, Edelsbrunner and Chazelle \cite{10} gave a deterministic algorithm of $O(m \log m + k)$ time and $O(m + k)$ space. Finally in 1995, Balaban \cite{4} gave an optimal deterministic $O(m \log m + k)$ time and $O(m)$ space algorithm.

For the segment intersection-counting problem we have the following. In 1986, Chazelle \cite{8} gave an $O(m^{1.695})$ time algorithm. This is the first algorithm in which counting can be done faster than reporting. In 1990, Agarwal \cite{2} gave a deterministic $O(m^{4/3} \log^{(\omega+2)/3} m)$ algorithm, where $\omega$ is some constant less than 3.33. Finally, in 1993, Chazelle \cite{8} gave an $O(m^{4/3} \log^{1/3} m)$ time and linear space algorithm.

The algorithm for segment intersection-counting yields an $O(n^{2+2/3} \log^{1/3} n)$ time algorithm for counting the number of crossings in a geometric graph. Faster algorithms can be given for some classes of geometric graphs. Rote, Woeginger, Zhu, and Wang \cite{15} provided an $O(n^2)$ time algorithm for computing $\overline{\tau}(G)$ when $G$ is a complete geometric graph. A \textit{layered graph} is a geometric graph whose vertex set is partitioned into sets $L_1, \ldots, L_r$ called \textit{layers} such that the following holds.
The vertices in layer $L_i$ have the same $y$-coordinate $y_i$;

- $y_1 < y_2 < \cdots < y_r$;
- vertices in layer $L_i$ are only adjacent to vertices in layers $L_{i-1}$ and $L_{i+1}$.

Waddle and Malhotra [18] provided an $O(|E| \log |E|)$ for computing $\overline{c}(G)$ when $G$ is a bilayered graph ($r = 2$).

2 Algorithm

In [12], the authors gave an algorithm for computing in $O(n^2)$ time the number of crossings of a complete geometric graph. The algorithm is based on first defining two types of “patterns” on the set of vertices of the graph. These patterns can be computed in $O(n^2)$ time and the number of crossings depends on the number of these patterns. We follow a similar approach.

Let $p$ and $q$ be two points in the plane. Let $\overrightarrow{pq}$ be the ray with apex $p$ and that passes through $q$; let $\overleftarrow{pq}$ be the ray with apex $p$ and with opposite direction to $\overrightarrow{pq}$. Let $(v, w, e)$ be a triple where $v$ and $w$ are a pair of adjacent vertices of $G$, and $e$ is an edge of $G$. We say that $(v, w, e)$ is a pattern of type:

- I) if $\overrightarrow{vw}$ intersects $e$; and
- II) if $\overleftarrow{vw}$ intersects $e$.

See Figure 1.

Let $\alpha$ and $\beta$ be the number of patterns $(v, w, e)$ of type I and Type II defined by $G$, respectively. As the following proposition shows, $\overline{c}(G)$ is determined by $\alpha$ and $\beta$.

**Proposition 1.** $\overline{c}(G) = (\alpha - \beta)/4$.

**Proof.** Without loss of generality suppose that no two edges of $G$ are parallel. Let $\gamma$ be the number of four tuples $(u, v, w, x)$ such that $(u, v), (w, x)$ are edges of $G$ and $\overrightarrow{uv}$ crosses $(w, x)$. Note that $\alpha = 4\overline{c}(G) + \gamma$ and $\beta = \gamma$; the result follows.

We compute $\alpha$ and $\beta$ in the following four steps.

**Step 1:** For each $v \in V$, compute the counterclockwise order of the vertices in $V \setminus \{v\}$ around $v$.

**Lemma 2.** Step 1 can be done in $O(n^2)$ time.
Proof. Dualize the set of vertices of \( G \). The corresponding line arrangement can be constructed in \( O(n^2) \) with standard algorithms; the desired orders can be computed from this arrangement in \( O(n^2) \) time.

\[ \begin{align*}
\text{Figure 2: An illustration of } & \text{ } vw^+ = 5 \text{ and } vw^- = 4 \\
\end{align*} \]

Let \( v \) and \( w \) be two vertices in \( G \). Let \( vw^+ \) the number of neighbors of \( w \) to the left of the directed line from \( v \) to \( w \), and let by \( vw^- \) the number of neighbors of \( w \) to the right of the directed line from \( v \) to \( w \). See Figure 2.

**Step 2:** Compute \( vw^+ \) and \( vw^- \) for each pair of vertices in \( G \).

**Lemma 3.** Step 2 can be done in \( O(n^2) \) time.

**Proof.** For each vertex \( v \) of \( G \) do the following. Rotate a line passing through \( v \) counterclockwise; let \( w_1, \ldots, w_{n-1} \) be the vertices of \( G \) in the order as they are encountered by this rotating line. This order can be computed in \( O(n) \) time using the counterclockwise order of the vertices of \( V \setminus \{v\} \) around \( v \). Compute \( vw^+_i \) and \( vw^-_i \) in linear time. Once \( vw^+_i \) and \( vw^-_i \) have been computed, \( vw^+_{i+1} \) and \( vw^-_{i+1} \) can be computed in constant time. Thus, the \( vw^+_i \)'s and \( vw^-_i \)'s can be computed in \( O(n) \) time. Therefore, Step 2 can be done in \( O(n^2) \) time.

Without loss of generality, assume that no two vertices of \( G \) have the same \( y \)-coordinate. For every vertex \( v \) of \( G \) let \( h_v \) be the horizontal ray with apex \( v \) that goes right.

**Step 3:** For every vertex \( v \) of \( G \) compute the number of edges of \( G \) that intersect \( h_v \).

**Lemma 4.** Step 3 can be done in \( O(n^2 \log n) \) time.

**Proof.** Let \( u \) and \( v \) be two vertices of \( G \), such that \( u \) is above \( v \). Note that an edge \((u, w)\) intersects \( h_v \) if and only if the following two conditions are satisfied. The vertex \( w \) is below \( v \), and \( w \) comes after \( v \) in the counterclockwise order around \( u \). Let \( d_{uv} \) be the number of edges incident to \( u \) that intersect \( h_v \). We use this observation to compute \( d_{uv} \) for every vertex \( v \) below \( u \) as follows.

Let \( v_1, v_2, \ldots, v_m \) be the vertices in \( V \), that lie below \( u \), sorted by height from top to bottom. We construct a nearly complete binary tree \( T \), whose leaves are \( v_1, \ldots, v_m \). The left to right order of these leaves coincides with the counterclockwise order around \( u \) starting from the leftmost vertex. \( T \) can be constructed in \( O(n) \) time.

We store information on the nodes of \( T \) that enables us to iteratively compute \( d_{uv} \). We start by setting \( i := 1 \), having computed \( d_{uv_i} \) we set \( i := i + 1 \) and update the information on the tree.
accordingly. The information we store on $T$ is the following. We mark all the leaves $v_j$ of $T$ such that $v_j$ is adjacent to $u$, and $v_j$ lies below $v_i$. At each internal node $x$ we store the numbers of marked leaves on the left subtree of $x$ and the number of marked leaves on the right subtree of $x$. See Figure 3. Note that for $i := 1$, this information can be computed in linear time from the bottom up. When setting $i := i + 1$ we only need to unmark the leaf $v_i$ and update the information stored in the nodes in the path from this leaf to the root. Therefore, this information is updated in $O(\log n)$ time.

Figure 3: The information stored on $T$ for $i = 1$

Suppose that we have just unmarked the leaf $v_{i-1}$ and updated the information stored on $T$. By the two previous conditions mentioned above, $d_{uv_i}$ is equal to the number of marked leaves to the right of $v_i$ in $T$. Let $w$ be the first common ancestor in $T$ of $v_i$ and $v_j$; then $v_j$ is to the right of $v_i$, if and only if, $v_i$ is in the left subtree of $w$ and $v_j$ is in the right subtree of $w$. We can compute $d_{uv_i}$ in $O(\log n)$ time by traversing the path from $v_i$ to the root. Thus, the $d_{uv_i}$'s can be computed in $O(n \log n)$ time. Since $h_v$ equals the sum of the $d_{uv}$ where $u$ lies above $v$, Step 3 can be computed in $O(n^2 \log n)$ time.

Let $u$ and $v$ be two vertices of $G$. Let $\alpha_{uv}$ be the number of edges of $G$ that intersect $\overrightarrow{uv}$. Let $\beta_{uv}$ be the number of edges of $G$ that intersect $\overleftarrow{uv}$. Note that $\alpha = \sum_{u \in V} \sum_{v \in N(u)} \alpha_{uv}$ and $\beta = \sum_{u \in V} \sum_{v \in N(v)} \beta_{uv}$.

**Step 4:** Compute $\alpha_{uv}$ and $\beta_{uv}$ for each pair of vertices $u, v$ of $G$.

**Lemma 5.** Step 4 can be done in $O(n^2)$ time.

**Proof.** Let $v_1, \ldots, v_{n-1}$ be the vertices of $G$ in counterclockwise order around $v$.

We show how to compute the $\alpha_{uv_i}$'s in linear time. Suppose that the $v_i$'s are ordered so that $v_1$ is the first vertex encountered when rotating $h_u$ counterclockwise around $u$. Note that $\alpha_{uv_1}$ is equal to the number of edges of $G$ that intersect $h_v$ minus $uv_1^-$. For $i > 1$, $\alpha_{uv_i}$ is equal to

$$\alpha_{uv_{i-1}} + uv_{i-1}^+ - uv_i^-;$$

see Figure 4. Therefore, the $\alpha_{uv_i}$'s can be computed in linear time.

Now we show how to compute the $\beta_{uv_i}$'s. Let $h_u'$ be the horizontal ray with apex $u$ that goes left. Suppose that the $v_i$'s are ordered so that $v_1$ is the first vertex encountered when rotating $h_u'$ counterclockwise around $u$. Let $w_1, \ldots, w_{1r_1}$ be the vertices of $G$ that lie between $h_u$ and $h_{v_1}$ — starting from $h_v$, in counterclockwise order around $u$. Note that $\beta_{uv_1}$ is equal to the number of
edges of $G$ that intersect $h_v$ plus

$$\sum_{j=1}^{r_i} (uw_{1j}^+ - uw_{1j}^-).$$

For $i > 1$, $w_{i1}, \ldots, w_{ir_i}$ be the vertices of $G$ that lie between $\overrightarrow{uv_{i-1}}$ and $\overrightarrow{uv_i}$ — starting from $\overrightarrow{uv_{i-1}}$, in counterclockwise order around $u$. Then $\beta_{uv_i}$ is equal to

$$\beta_{uv_{i-1}} + \sum_{j=1}^{r_i} (uw_{ij}^+ - uw_{ij}^-).$$

Since the $w_{ij}$ are visited only once, the computation of all the $\beta_{uv_i}$'s takes linear time. The result follows.

$$\begin{align*}
\text{Figure 4: The iterative step for computing } \alpha_{uv_i} \text{ in Step 4. In this case } vw_{i-1}^+ &= 3, \ vw_i^- = 4, \\
\alpha_{vw_{i-1}} &= 6 \text{ and } \alpha_{vw_i} = 5.
\end{align*}$$

2.1 Counting Crossings in $O(n^2)$ Time

Of the four steps of the algorithm, only Step 3 takes superquadratic time. We mention two instances in which $\mathcal{C}(G)$ can be computed in $O(n^2)$ time. Note that the choice of direction of $h_v$ is irrelevant — it is only used as a starting point to compute the $\alpha_{uv}$’s and $\beta_{uv}$’s in Step 4.

- **Convex Geometric Graphs**
  Suppose that the vertices of $G$ are in convex position. For each $v$ in $V$, choose an $h_v$ that does not intersect the convex hull of $G$; thus no edge of $G$ intersects $h_v$. Therefore, in this case Step 3 can be done in $O(n \log n)$ time and $\mathcal{C}(G)$ can be computed in $O(n^2)$ time.

- **Layered Graphs**
  Suppose that $G$ is a layered graph with layers $L_1, \ldots, L_r$. Let $G_i$ be the subgraph of $G$ induced by $L_i$ and $L_{i+1}$. Note that $G_i$ can be regarded as a convex geometric graph. Since $\mathcal{C}(G) = \sum_{i=1}^{r-1} \mathcal{C}(G_i)$, $\mathcal{C}(G)$ can be computed in $O(n^2)$ time.

**Acknowledgments.** This work was initiated at the 2nd Reunion of Optimization, Mathematics, and Algorithms (ROMA 2018), held in Mexico City, 2017.
References

[1] B. M. Ábrego, S. Fernández-Merchant, J. Leaños, and G. Salazar. A central approach to bound the number of crossings in a generalized configuration. *Electronic Notes in Discrete Mathematics*, 30(0):273–278, 2008. The IV Latin-American Algorithms, Graphs, and Optimization Symposium.

[2] Pankaj K. Agarwal. Partitioning arrangements of lines. II. Applications. *Discrete Comput. Geom.*, 5(6):533–573, 1990.

[3] Oswin Aichholzer, Frank Duque, Oscar E. García-Quintero, and Carlos Hidalgo-Toscano. An ongoing project to improve the rectilinear crossing constant.

[4] Ivan J. Balaban. An optimal algorithm for finding segments intersections. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, SCG ’95, pages 211–219, New York, NY, USA, 1995. ACM.

[5] Jon L. Bentley and Thomas A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, C-28(9):643–647, Sept 1979.

[6] Franz Brandenburg, David Eppstein, Michael T. Goodrich, Stephen Kobourov, Giuseppe Liotta, and Petra Mutzel. Selected open problems in graph drawing. In Giuseppe Liotta, editor, *Graph Drawing*, pages 515–539, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[7] K. Q. Brown. Comments on “algorithms for reporting and counting geometric intersections”. *IEEE Transactions on Computers*, C-30(2):147–148, Feb 1981.

[8] Bernard Chazelle. Reporting and counting segment intersections. *J. Comput. System Sci.*, 32(2):156–182, 1986. 16th annual ACM-SIGACT symposium on the theory of computing (Washington, D.C., 1984).

[9] Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, April 1993.

[10] Bernard Chazelle and Herbert Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. Assoc. Comput. Mach.*, 39(1):1–54, 1992.

[11] Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry. II. *Discrete Comput. Geom.*, 4(5):387–421, 1989.

[12] Ruy Fabila-Monroy and Jorge López. Computational search of small point sets with small rectilinear crossing number. *Journal of Graph Algorithms and Applications*, 18(3):393–399, 2014.

[13] Ketan Mulmuley. A fast planar partition algorithm. I. *J. Symbolic Comput.*, 10(3-4):253–280, 1990.

[14] J. Nievergelt and F. P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Commun. ACM*, 25(10):739–747, October 1982.
[15] Günter Rote, Gerhard Woeginger, Binhai Zhu, and Zhengyan Wang. Counting $k$-subsets and convex $k$-gons in the plane. *Information Processing Letters*, 38(3):149 – 151, 1991.

[16] Marcus Schaefer. The graph crossing number and its variants: A survey. *The electronic journal of combinatorics*, Dynamic Survey #DS21:21–22, 2017.

[17] Michael Ian Shamos and Dan Hoey. Geometric intersection problems. In *17th Annual Symposium on Foundations of Computer Science (Houston, Tex., 1976)*, pages 208–215. IEEE Comput. Soc., Long Beach, Calif., 1976.

[18] Vance Waddle and Ashok Malhotra. An $E \log E$ line crossing algorithm for levelled graphs. In *Graph drawing (Štiřín Castle, 1999)*, volume 1731 of *Lecture Notes in Comput. Sci.*, pages 59–71. Springer, Berlin, 1999.