# Embedding Planar Graphs on a Grid

Seminar "Algorithms on Sparse Graphs", Summer 2018
Supervisors: Jan Dreier, Philipp Kuinke

Pascal Hein

## Contents



## 1 Introduction

Graphs, due to their simple and generic mathematical formulation, have many applications both as models of the real world and as purely theoretical constructs. It is commonplace to represent dependency relationships, transition diagrams, social connections, maps and circuits as graphs before applying mathematical theorems or algorithms to a problem.

Usually these algorithms are designed for space or runtime efficiency, which also influences the choice of input format. Common representations are adjacency matrices, edge sets, or other data structures specific to the task in question. However, none of these formats is intuitively accessible to humans wanting to understand the graph's structure, visualize the workings of an algorithm, or communicate about certain features. Furthermore, if a problem has been modelled and solved using graph theory, the output should be formatted in a way so that experts of other domains can understand and apply the results. This is why graph visualization is a critical and diverse topic: only if we are able to *draw* graphs are we able to *talk* about graphs and to make observations or understand algorithms on them, which makes visualization necessary to use graphs effectively.

However, there are many criteria for what makes a good drawing of a graph: Common requirements are to impose restrictions on the form of edges (e.g. straight or parallel to the coordinate axes) or the location of vertices (e.g. for maps corresponding to the real world, chronological ordering in a family tree or flowchart, or grouping vertices by a certain measure of distance). Furthermore there is a wide range of layout optimization criteria including total size of a bounding box (given a fixed minimum resolution), the number of edge crossings, or various types of regular patterns and symmetries [1].

Obviously, the choice of design is, first and foremost, determined by the purpose of the drawing, and the domain it is being applied to. However, it is also influenced by constraints on the graphs we have to draw: While certain optimization problems might be hard to solve in general (e.g. the required number of edge crossings is NP-complete [2]), they could be feasible if the input is known to be restricted to a certain subset of all graphs.

Two of the most commonly desired properties are having few edge crossings and low curvature, due to easier perception by users [3]. The algorithm by Marek Chrobak and Thomas H Payne [4] that will be discussed here optimizes these characteristics for the class of planar graphs, drawing them with straight edges and without edge crossings. In this paper, we intend to give an overview over their algorithm, elaborate on several points, and place it into context by

considering related work. In section 2, we review relevant definitions and theorems in graph theory that will be prerequisite to the algorithm. Section 3 will introduce both prior and subsequent publications covering the topic, before we present the main idea of the algorithm in section 4. Finally, section 5 is concerned with improving the runtime, and proving correctness as well as the claimed bound on the runtime.

# 2 Preliminaries

We begin by defining the terms used in this report and stating necessary theorems. Throughout the remainder of our paper, we identify a graph $G = (V, E)$ with the set of its vertices, $V$, and a set of undirected edges, $E \subseteq \{\{x, y\} \mid x, y \in V, x \neq y\}$.

The output of a drawing algorithm should be an *embedding* of the graph $G = (V, E)$, i.e. a map assigning to each vertex $v \in V$ a point $(x(v), y(v)) \in \mathbb{R}^2$, and to each edge $e = \{v_1, v_2\} \in E$ a simple curve connecting the images of its endpoints; an embedding is a *straight-line embedding* if the images of edges are line segments, in which case it is uniquely determined by the positions assigned to the vertices.

Recall that a graph is called *planar* if there exists an embedding such that any two arcs may only intersect at their endpoints. In this case, we may speak of faces which are delineated by edges, and refer to the unbounded section of the plane as the *exterior face* of $G$. The *boundary* of $G$ refers to the edges adjacent to this exterior face. Lastly, a graph is called *biconnected* if it remains connected even after removing any single vertex.

Due to Euler's well-known formula on the number of vertices, faces and edges in planar graphs, $|V| + |F| - |E| = 2$, and the fact that any face is bounded by at least three edges belonging to two faces each ($2\,|E| \geq 3\,|F|$), we obtain $|E| \leq 3\,|V| - 6$; in particular, the number of edges is at most linear in the number of vertices which will henceforth be designated by $n$.

## 2.1 Fáry's Theorem

Since planar graphs are defined in terms of arbitrary embeddings, it is not obvious that it should be possible to draw them without edge crossings when restricted to straight lines. Thus, the first result related to the problem at hand is the following theorem implying that the desired type of drawing always exists (albeit on arbitrarily large grids).

**Proposition 1** [5]**.** *Any planar graph has a straight-line embedding.*

This embedding may be obtained inductively by removing a vertex $v \in V$ with degree less than 6, embedding the remaining vertices such that the face containing $v$ is not the exterior face, and reinserting $v$ with straight edges.

## 2.2 Triangulated Planar Graphs

The algorithm we are about to present will rely on graphs being *internally triangulated*. This refers to the case when each face, except possibly the exterior one, is bounded by exactly three edges. An internally triangulated graph where the exterior face *also* is a triangle is called *triangulated* or *maximally planar*. This definition leads us to the following observation.

**Proposition 2.** *A planar graph $G$ is triangulated if and only if no edge can be added to obtain a planar graph.*

*Proof.* If $G$ is not triangulated, suppose that $v_1$, $v_2$, $v_3$, $v_4$ appear in this order on the boundary of a non-triangular face. If one of the diagonals $v_1v_3$ and $v_2v_4$ exists, it divides the outside into two regions each containing one of the other two vertices. Hence the other diagonal cannot exist and we can add it inside the face.

The converse is clear: any edge connecting vertices of a triangle must be a side, hence it already exists in $G$. □

In particular, proposition 2 implies that we are able to obtain a triangulated graph from any planar graph by iteratively adding

edges to non-triangular faces. We can thus work exclusively on triangulated graphs; any algorithm can be applied to non-triangulated graphs by adding a linear amount of missing edges and removing them from the embedding afterwards.

### 2.3 Canonical Ordering

We will now state the theorem that is fundamental to the workings of the presented algorithm, because it provides an order in which vertices may be iteratively inserted.

**Theorem 1** [6]**.** *Let $G = (V, E)$ be a triangulated planar graph. Then $G$ has a* canonical ordering*, i.e. an ordering $v_1, \ldots, v_n$ of $V$ such that $v_1, v_2, v_n$ bounds its exterior face and, for each $k = 3, \ldots, n-1$:*

1. *the subgraph $G_k$ induced by $v_1, \ldots, v_k$ is biconnected and internally triangulated,*
2. *the boundary of its exterior face contains the edge $v_1v_2$,*
3. *$v_{k+1}$ is in the exterior face of $G_k$ and its neighbours in $G_k$ are consecutive on the boundary excluding $v_1v_2$.*

*Proof (adapted from [1]).* Let $v_1, v_2, v_n$ be the vertices on the boundary of the triangular outer face of $G_n$ in any order. The first two conditions are clearly satisfied, so we may induct backwards on $n$.

Now assume that $G_{k+1}$ satisfies the conditions for some $k = 3, \ldots, n-1$; we consider chords, i.e. edges between vertices of $G_{k+1}$'s exterior boundary $B = (v_1, v_2, \ldots, v_1)$ which are not adjacent to the exterior face. If there is any chord at all, pick $c = \{x, y\}$ such that the path $P = (x, \ldots, v_1, v_2, \ldots, y) \subsetneq B$ (along the boundary and containing $v_1v_2$) is as long as possible. Vertices on $B \setminus P$ cannot be incident to any chords because these would intersect $c$ or induce a longer path containing $v_1v_2$. Therefore, it is always possible to choose $v_{k+1} \notin \{v_1, v_2\}$ such that it is not incident to any chord. We claim that $G_k$ obtained by removing $v_{k+1}$ satisfies the conditions.

Clearly, $v_{k+1}$ must be on the exterior face of $G_k$ since it is on the boundary of $G_{k+1}$; furthermore the boundary of $G_k$ also contains $v_1v_2$ because $v_{k+1} \notin \{v_1, v_2\}$. $G_k$ is internally triangulated because $G_{k+1}$ was, and none of the edges or faces inside the boundary of $G_k$ have changed. The edges from $v_{k+1}$ to its neighbours cannot cover $v_1v_2$ since this edge is on the outside of $G_{k+1}$, and any vertex between two neighbours of $v_{k+1}$ in $G_k$ must also be a neighbour because $G_{k+1}$ is internally triangulated.

Assume that $G_k$ is not biconnected and has a vertex $v_j$ such that removing $v_j$ would disconnect $G_k$. Clearly, $v_j$ must be on the boundary of $G_k$ because otherwise removing it would also disconnect the biconnected graph $G_{k+1}$. In $G_{k+1} \setminus \{v_j\}$, the only connection between the components of $G_k \setminus \{v_j\}$ is via $v_{k+1}$. Together with $v_{k+1}$ and $v_j$, the connecting edges incident to $v_{k+1}$ induce a cycle with at least four vertices as shown in figure 1, so the (non-boundary) edge $v_{k+1}v_j$ must exist on its inside due to triangulation, and cannot exist on its outside. Since the neighbours of $v_{k+1}$ are consecutive, no edge can "hide" $v_j$ and $v_{k+1}v_j$ is a chord, contradicting the choice of $v_{k+1}$. □

The computation of a canonical ordering will later be the first step of the algorithm; it can be performed efficiently in a way resembling the proof of the previous theorem [1], or using an algorithm by Kant [7].

## 3 Related Results

In this section, we will give an overview over algorithms producing straight-line embeddings of planar graphs.

Fáry's Theorem stated in section 2 guarantees that straight-line embeddings exists; however, early drawing algorithms [8, 9] can lead to dense regions that may cause problems due to finite precision in the implementation and limited resolution when displaying the result (e.g. on a computer screen).

Thus, Rosenstiehl and Tarjan's proposed solution is "a straight-line embedding such

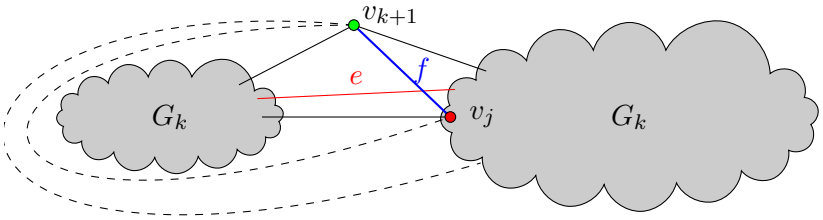


Figure 1: Proof of biconnectedness: The edge $e$ does not exist, so $f$ must be present; then dashed lines cannot exist and $f$ is a chord

that the vertices map to integer lattice points with coordinates bounded by $n^k$ for some constant $k$" [10], the existence of which was proved by de Fraysseix et al. [6]. Their algorithm has a runtime of $O(n \log n)$; however, they "suspect that a linear time algorithm exists".

The algorithm by Chrobak and Payne [4] which will be presented below is based on the method used by de Fraysseix et al. [6]. However, the computation of coordinates is different, leading to significantly simpler steps and a better runtime complexity, indeed achieving linear time as suggested by de Fraysseix et al. They also indicate that it is possible to reduce the grid size to $(n-2) \times (n-2)$ which will be pointed out later.

Schnyder [11] gave a different linear time algorithm for drawing planar graphs on a $(n-2) \times (n-2)$ grid; however Chrobak and Payne claim that "[their] approach is easier to implement, and the resulting embeddings tend to be more aesthetic" [4].

Nevertheless, the algorithms by de Fraysseix et al. [6] as well as Chrobak and Payne [4] have a significant drawback due to the way they produce the embedding based on triangulated graphs. Even though the original input may have been very sparse, the algorithms will add edges until the graph is triangulated and remove them after the vertices' positions have been determined. This can result in unaesthetic drawings if the edge density varies significantly, and it can produce complicated shapes for the faces of the graph. A solution is provided by *convex* drawings guaranteeing that every face will be embedded as a convex polygon. Tutte [8] gave a barycentric method for constructing these for 3-connected graphs, and algorithms by Chiba et al. [9, 12] produce these drawings in linear time whenever this task is possible.

A method for convex graph drawings on the grid with linear time complexity has also been presented by Chrobak and Kant [13]; it has similarities with the algorithm discussed below but uses sets forming a canonical decomposition instead of adding one vertex at a time.

# 4 Algorithm

What follows is a description of the general idea of iteratively shifting vertices, and a high-level view of the algorithm presented by Chrobak and Payne [4].

## 4.1 Description of the Embedding

We start with a triangulated planar graph $G = (V, E)$; assume that a canonical ordering $v_1, \ldots, v_n$ of its vertices is given. Beginning by embedding $v_1$ at $(0, 0)$, $v_2$ at $(2, 0)$, and $v_3$ at $(1, 1)$, we add one vertex $v_{k+1}$ at a time to the embedding of $G_k$ until we have embedded $G_n = G$. We will denote by $x(v)$ and $y(v)$ the $x$- and $y$-coordinate, respectively, of the image of $v$ under the embedding, and define the *contour* of $G_k$ to be the vertices $v_1 = w_1, \ldots, w_m = v_2$ along the boundary of $G_k$'s exterior face excluding the edge $v_1v_2$.

We will maintain the following invariants [4]:

1. In $G_k$, $v_1$ and $v_2$ are embedded at $(0, 0)$ and $(2k-4, 0)$, respectively.
2. The vertices along the contour of $G_k$ are assigned ascending $x$-coordinates.

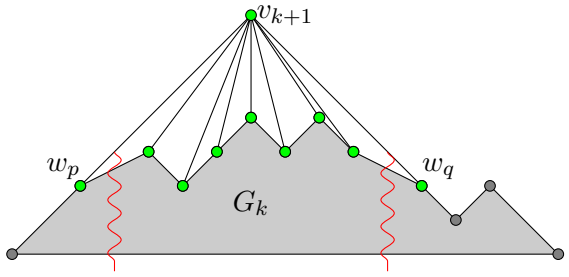


Figure 2: Shifting vertices to create gaps

3. The edges along the contour of $G_k$ have slope $\pm 1$.

### 4.2 Adding the next Vertex

Assume that we have successfully embedded $G_k$ into the grid such that its contour is $v_1 = w_1, \ldots, w_m = v_2$. We now consider the problem of placing $v_{k+1}$; by the properties of a canonical embedding, it has at least two neighbours in $G_k$ (since $G_{k+1}$ is biconnected) and they form a subsequence $w_p, \ldots, w_q$ of the contour with $1 \leq p < q \leq m$.

If we were to simply add $v_{k+1}$ such that the embedding of $w_p v_{k+1}$ has slope $+1$ and that of $v_{k+1}w_q$ has slope $-1$, these edges might overlap with $w_p w_{p+1}$ or $w_{q-1}w_q$ if they have the same slope. This problem is solved by first creating gaps, i.e. by shifting every $w_i$ with $p < i < q$ to the right by 1, and every $w_i$ with $q \leq i \leq m$ to the right by 2 as shown in figure 2. Now, the slope of the (imaginary) edges $w_p w_j$ and $w_j w_q$ for $p < j < q$ is strictly between $-1$ and $+1$, so the slope of $w_j v_{k+1}$ has absolute value strictly greater than 1. Therefore, the embeddings of edges incident to $v_{k+1}$ can neither intersect each other nor any already existing edge, except at their endpoints.

However, this only holds as long as we do not move any of the concerned vertices. For instance, shifting $v_{k+1}$ to the right by 1 in any further step could result in the edges $w_{p+1}w_{p+2}$ and $w_{p+1}v_{k+1}$ both having slope $+1$. We therefore need to store the information that $v_{k+1}$ and $w_{p+1}, \ldots, w_{q-1}$ should never be moved independently. This is accomplished using sets $L(v) \supseteq \{v\}$ (the vertices *moved with* $v$) such that we only ever move a set $L(v)$ as a whole: they are therefore computed recursively as $L(v_{k+1}) = \{v\} \cup \bigcup_{i=p+1}^{q-1} L(w_i)$ whenever $v_{k+1}$ covers $w_{p+1}, \ldots, w_{q-1}$ on the contour of $G_k$, in order to prevent the covered neighbours of $v_{k+1}$ from moving relative to $v_{k+1}$. We have to take the union of the sets $L(w_i)$ since all vertices moving with $w_i$ should, of course, also move with $v_{k+1}$ by transitivity.

It is easily observed that the embeddings of $w_p v_{k+1}$ and $v_{k+1}w_q$ have the correct slope if and only if the coordinates of $v_{k+1}$ are computed as

$$\begin{aligned} x(v_{k+1}) &= \frac{1}{2}(x(w_q) + y(w_q) \\ &\quad + x(w_p) - y(w_p)), \\ y(v_{k+1}) &= \frac{1}{2}(x(w_q) + y(w_q) \\ &\quad - x(w_p) + y(w_p)). \end{aligned} \tag{1}$$

Assembling all the steps outlined above, listing 1 shows the pseudocode for the drawing algorithm.

Note that the graph is embedded in a $(2n-4) \times (n-2)$ grid because $v_2$ starts out at $(2, 0)$ in $G_3$ and is moved by 2 in each step until we have reached $G_n$; $v_1$ always remains at $(0, 0)$. The third point on the convex hull is $v_n$ which must be placed at $(n-2, n-2)$ because the slopes of $v_1 v_n$ and $v_n v_2$ are 1 and $-1$ respectively.

It is possible to modify this embedding by creating only one gap between $w_p$ and $w_{p+1}$. The new vertex $v_{k+1}$ will be placed so that $x(v_{k+1}) = x(w_p) + 1$, and the slope of $v_{k+1}w_q$ is $-1$. While downward edges along the contour still have slope $-1$, upward edges may then have any slope in $[0, \infty)$. The result is an asymmetric drawing on a grid of size $(n-2) \times (n-2)$; Chrobak and Payne cite a private communication with Schnyder [4].

De Fraysseix et al. [6] also give an example of nested triangles requiring a grid size of $\left(\frac{2}{3}n - 1\right) \times \left(\frac{2}{3}n - 1\right)$ as shown in figure 3; the quadratic grid size is therefore asymptotically optimal while it is unclear whether it is possible to use grids of size at most $cn^2 + O(n)$ for any $c < 1$.

**Listing 1** Pseudocode for high-level drawing algorithm

```
procedure DRAW(G: graph, v_1, ..., v_n: canonical ordering)
    Embed v_1 ↦ (0,0), v_2 ↦ (2,0), v_3 ↦ (1,1)
    Set L(v_1) = {v_1}, L(v_2) = {v_2}, L(v_3) = {v_3}
    for 3 ≤ k < n do
        // Let v_1 = w_1, ..., w_m = v_2 be the contour of G_k
        // Let w_p, ..., w_q be the neighbours of v_{k+1} in G_k such that 1 ≤ p < q ≤ m
        Shift L(w_{p+1}), ..., L(w_{q-1}) by 1
        Shift L(w_q), ..., L(w_m) by 2
        Embed v_{k+1} at coordinates in (1)
        Set L(v_{k+1}) = {v_{k+1}} ∪ ⋃_{i=p+1}^{q-1} L(w_i)
    end for
end procedure
```

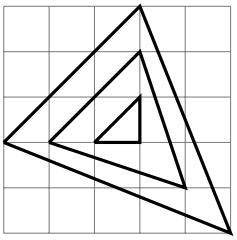

Figure 3: Graph class requiring a grid size of $\left(\frac{2}{3}n-1\right) \times \left(\frac{2}{3}n-1\right)$

# 5 Analysis & Improvement

## 5.1 Prerequisites

We first consider the problem of triangulating the graph and obtaining a canonical ordering. Kant [7] provides an algorithm running in linear time that, given a planar embedding of a graph $G$, triangulates $G$ and computes a canonical ordering at the same time. However, we can embed a graph using the Hopcroft-Tarjan planarity testing algorithm [14] in linear time. After completion of the drawing algorithm, we can redraw all edges in the graph $G$ using only the positions of the vertices since their number is linear in the number of vertices, thus "removing" those edges added only for triangulation.

## 5.2 Binary Tree and Offsets

We are therefore interested in a linear-time implementation of the procedure in listing 1; however, in the given version we have to move a large fraction of vertices in every iteration which leads to a quadratic runtime. Noticing that these shifts are the deciding factor for the complexity, we intend to store the information about offsets in one location only instead of in every vertex.

Note that the sets $L(w_1), \ldots, L(w_m)$ form a partition of all vertices $v_1, \ldots, v_k$ if $w_1, \ldots, w_m$ is the contour of $G_k$ (because $L(w)$ is merged into a new contour vertex's set whenever $w$ is covered), and each of them only moves as a whole. We can therefore shift each $L(w_i)$ with a single offset $\Delta x(w_i)$.

We will further represent the sets $L(w_i)$ as a single binary tree with root $v_1$, where the left child of a vertex $v_k$ is the leftmost vertex on the contour of $G_{k-1}$ that was covered by adding $v_k$ (i.e. $v_k$'s second neighbour from the left, if there are more than two neighbours), and the right child is the right neighbour on the contour of the current graph, or the last graph before $v_k$ was covered (if this neighbour was covered at the same time). Notice that, at any step, the boundary of $G_k$ is obtained by starting from $v_1$ and following the sequence of right children. Furthermore the set $L(w_i)$ corresponds to $w_i$ together with its entire left subtree.

Each vertex $v$ of this binary tree will store an $x$-offset $\Delta x(v)$ and the $y$-coordinate $y(v)$. This $x$-offset is measured relative to the $x$-coordinate of its parent, which is the left neighbour along the contour of the last graph $G_k$ before $v$ was covered (i.e. $v$ is on the contour of $G_k$ but not of $G_{k+1}$), or

the covering vertex $v_{k+1}$ itself if $v$ was the leftmost covered vertex.

We can now perform the shift operations of $\bigcup_{i=p+1}^{q-1} L(w_i)$ by 1 and of $\bigcup_{i=q}^{m} L(w_i)$ by 2 simply by increasing $\Delta x(w_{p+1})$ and $\Delta x(w_q)$ by 1 each, because this also affects all successors along the contour and their corresponding sets $L(w_i)$.

When adding a new vertex $v_{k+1}$, we need to update the binary tree to maintain the structure described above. This is achieved by making $v_{k+1}$ the right child of $w_p$, and $w_q$ the right child of $v_{k+1}$ as illustrated in figure 4. If there were any vertices $w_{p+1}, \ldots, w_{q-1}$ that have now been covered, the first of these should be made the left child of $v_{k+1}$ since it now belongs to $L(v_{k+1})$, while the last covered vertex's right child should no longer be $w_q$ which remains on the contour. Care must be taken whenever the parent of a vertex changes because the offset needs to be recomputed relative to the new parent.

It remains to be shown that we can compute the location of $v_{k+1}$ exclusively given the information stored in this tree. This is achieved by defining $\Delta x = x(w_q) - x(w_p)$ and rewriting the formula in (1) as

$$\begin{aligned} \Delta x(v_{k+1}) &= x(v_{k+1}) - x(w_p) \\ &= \frac{1}{2}\left(y(w_q) - y(w_p) + \Delta x\right), \\ y(v_{k+1}) &= \frac{1}{2}\left(y(w_q) + y(w_p) + \Delta x\right). \end{aligned} \tag{2}$$

Notice that $\Delta x = \Delta x(w_{p+1}) + \cdots + \Delta x(w_q)$ in $G_k$. The number of terms is less than the degree of $v_{k+1}$; since the total degree of the graph is linear in $|V|$, the summations in all the iterations together require only linear time.

In the end, the $x$-coordinates of all vertices are computed by recursively summing up the offsets on the path in the binary tree starting from $v_1$, using a simple procedure shown in listing 2.

### 5.3 Final Algorithm

We can now put together all the computations presented above to describe the complete algorithm, as displayed in listing 3.

### 5.4 Correctness

As argued above, the modified algorithm in listing 3 computes exactly the same vertex positions as the pseudocode in listing 1 because the sum of offsets on the path in the tree always corresponds to the $x$-coordinate of any vertex. We therefore only need to prove that the algorithm in listing 1 produces a straight-line embedding of its input graph on a $(2n-4) \times (n-2)$ grid.

In order to obtain such an embedding, it is necessary that the coordinates in (1) are integers, which is implied by following theorem.

**Theorem 2.** *For $3 \leq k \leq n$, the coordinates $x(w_i)$ and $y(w_i)$ of each vertex $w_i$ on the boundary of $G_k$ are integers with even sum.*

*Proof.* We proceed by induction, where the base case $k = 3$ is satisfied by the choice of embedding for $G_3$.

Now assume that the theorem holds for $k$ and we are embedding $v_{k+1}$. Contour vertices up to and including $w_p$ are not moved at all, while those starting from $w_q$ are moved by 2. Therefore, all vertices except $v_{k+1}$ that belong to the contour of $G_{k+1}$ still have integer coordinates with even sum.

Since $x(w_p)$ and $y(w_p)$ are integers with even sum, their difference must also be even. Applying this to equation (1), $2x(v_{k+1})$ is the sum of two even numbers and $2y(v_{k+1})$ is the difference of two even numbers, so the coordinates of $v_{k+1}$ are both integers. Their sum is $x(v_{k+1}) + y(v_{k+1}) = x(w_q) + y(w_q)$ and therefore even. □

We can observe that in each step, the contour vertices $w_1, \ldots, w_m$ of the previous graph $G_k$ and their corresponding sets $L(w_i)$ are shifted by a sequence of nondecreasing integers before adding the new vertex. Their coordinates remain integers; however we need to prove that these shifts do not introduce any new edge crossings.

**Theorem 3** [6]**.** *Let $G_k$ be a straight-line embedded graph that has been obtained after a certain number of iterations of the algorithm. If its contour is $v_1 = w_1, \ldots, w_m = v_2$ and*

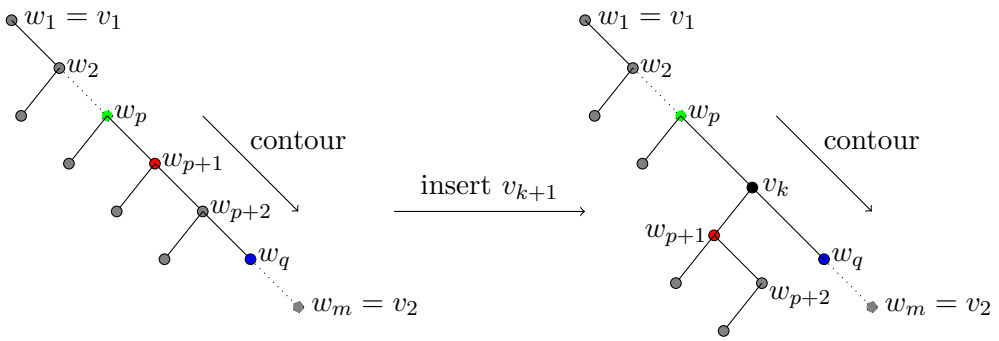


Figure 4: Inserting a new vertex into the binary tree

**Listing 2** Offset accumulation

```
procedure AccumulateOffsets(v: vertex, x(p): x-coordinate of parent)
    x(v) ← x(p) + Δx(v)
    if v.left ≠ nil then
        AccumulateOffsets(v.left, x(v))
    end if
    if v.right ≠ nil then
        AccumulateOffsets(v.right, x(v))
    end if
end procedure
```

*$\rho_1 \leq \cdots \leq \rho_m$ is a nondecreasing sequence of integers, then moving each set $L(w_i)$, $1 \leq i \leq m$, by $\rho_i$ to the right does not produce any edge crossings.*

*Proof.* We induct on $k$; clearly, in the triangle $G_3$, we can shift the vertices as desired.

Now assume that the statement holds for $G_k$, for some fixed $k = 3, \ldots, n-1$, and we intend to prove it for $G_{k+1}$. As before, let the vertices covered by $v_{k+1}$ be $w_p, \ldots, w_q$.

We are given a sequence satisfying $\rho_1 \leq \cdots \leq \rho_p \leq \rho \leq \rho_q \leq \cdots \leq \rho_m$ by which the contour $w_1, \ldots, w_p, v_{k+1}, w_q, \ldots, w_m$ of $G_{k+1}$ together with the sets $L(w_i)$ should be shifted.

Due to the way the algorithm constructs an embedding of $G_{k+1}$, this is equivalent to first shifting each set $L(w_i)$ for the contour vertices $w_1, \ldots, w_p, w_{p+1}, \ldots, w_{q-1}, w_q, \ldots, w_m$ by

$$\underbrace{0, \ldots, 0}_{1 \leq i \leq p}, \underbrace{1, \ldots, 1}_{p+1 \leq i \leq q-1}, \underbrace{2, \ldots, 2}_{q \leq i \leq m},$$

before applying the shifts

$$\rho_1, \ldots, \rho_p, \underbrace{\rho, \ldots, \rho}_{q-p-1 \text{ times}}, \rho_q, \ldots, \rho_m$$

to the sets $L(w_i)$, because $L(v_{k+1}) \setminus \{v_{k+1}\} = L(w_{p+1}) \cup \cdots \cup L(w_{q-1})$.

Therefore, the sequence of total shifts applied to the contour of $G_k$ is given by

$$\rho'_i = \begin{cases} \rho_i, & 1 \leq i \leq p \\ \rho + 1, & p+1 \leq i \leq q-1 \\ \rho_i + 2, & q \leq i \leq m. \end{cases}$$

Since $\rho'_1 \leq \cdots \leq \rho'_m$, by our induction hypothesis $G_k$ does not contain any edge crossings after applying these shifts. However we have argued above that placing $v_{k+1}$ cannot introduce any edge crossings, and $v_{k+1}$ does not move relative to any of $w_{p+1}, \ldots, w_{q-1}$. Widening the gap between $w_p$ and $w_{p+1}$ or $w_{q-1}$ and $w_q$ clearly cannot lead to edge crossings involving $v_{k+1}$ either.

Hence, the resulting graph is still straight-line embedded. In particular, every $G_k$ as obtained from the algorithm is planar, and the algorithm correctly produces a straight-line embedding of $G_n = G$ as claimed. □

We therefore obtain a straight-line embedding of $G_n = G$ on the grid; this concludes the proof of correctness for the algorithm.

**Listing 3** Complete drawing algorithm

```
procedure DRAW(G: graph) // Computes a straight-line embedding of G
    Embed G into the plane // Hopcroft-Tarjan algorithm [14]
    Triangulate G and compute a canonical ordering v_1, ..., v_n // Kant's algorithm [7]
    // Initialize binary tree, embedding of G_3
    v_1.right ← v_3, v_3.right = v_2, v_2.right ← nil
    v_1.left ← nil, v_2.left ← nil, v_3.left ← nil
    Δx(v_1) ← 0, Δx(v_3) ← 1, Δx(v_2) ← 1
    y(v_1) ← 0, y(v_3) ← 1, y(v_2) ← 0
    for 3 ≤ k < n do
        // Let w_p, ..., w_q be the neighbours of v_{k+1} in G_k
        Δx(w_{p+1}) ← Δx(w_{p+1}) + 1 // Create gaps next to w_p and w_q
        Δx(w_q) ← Δx(w_q) + 1
        Δx ← Δx(w_{p+1}) + ··· + Δx(w_q)
        Δx(v_{k+1}) ← 1/2 (y(w_q) − y(w_p) + Δx) // Compute location using equation (2)
        y(v_{k+1}) ← 1/2 (y(w_q) + y(w_p) + Δx)
        w_p.right ← v_{k+1}
        v_{k+1}.right ← w_q
        Δx(w_q) ← Δx − Δx(v_{k+1}) // Update offset from new parent for w_q
        if p + 1 < q then // Handle covered vertices
            v_{k+1}.left ← w_{p+1}
            Δx(w_{p+1}) ← Δx(w_{p+1}) − Δx(v_{k+1})
            w_{q−1}.right ← nil
        else
            v_{k+1}.left ← nil
        end if
    end for
    ACCUMULATEOFFSETS(v_1, 0)
end procedure
```

### 5.5 Time Complexity

At the beginning of this section we already referred to related publications proving linear bounds on the runtime of graph embedding and triangulation as well as the computation of a canonical ordering.

Notice that each iteration in listing 3 has constant complexity except for the computation of the sum $\Delta x$. However, we argued that the runtime of these sums aggregated over all iterations is linear in the number of edges, which again is linear in the number of vertices. Furthermore, the procedure ACCUMULATEOFFSETS runs in linear time because it is called exactly once for every vertex of the graph. Edges added for the triangulation can be removed after computing all positions because their total number is linear in $n$.

Hence we obtain the claimed result: The runtime of the entire drawing algorithm is linear in the number of vertices, which is clearly optimal. We further note that the algorithm also has linear space complexity because a constant amount of memory is required for each vertex.

## 6 Conclusion

Graph visualization is an equally important and diverse topic that has many applications. There are a number of desirable and concurrent properties, many of which are hard to optimize in general but tractable for planar graphs. Of special interest are algorithms that minimize crossings and curvature of edges.

In this paper, we have given an overview over results related to such embeddings. In particular, we presented and elaborated on an algorithm published by Chrobak and Payne that produces a straight-line embedding of a planar graph on a small coordinate grid, before proving its correctness and a bound on its time complexity. A suitable, "canonical" ordering of the vertices is used as the basis for an iterative approach that produces planar straight-line embeddings by shifting vertices.

By means of an aptly chosen representation of the dependencies among vertex coordinates and an efficient structure for performing shifts, only linear time is required.

Even though other algorithms exist producing similar drawings, some of them guaranteeing convexity or requiring a smaller grid, the presented algorithm is notable for its efficiency and simplicity.

## References


[1] Tamara Mchedlidze and Martin Nöllenburg. *Lecture on Algorithms for Visualization of Graphs*. 2014. URL: `i11www.iti.kit.edu/teaching/winter2014/graphvis/index`.

[2] Michael R Garey and David S Johnson. "Crossing number is NP-complete". In: *SIAM Journal on Algebraic Discrete Methods* 4.3 (1983), pp. 312–316.

[3] Chris Bennett et al. "The aesthetics of graph visualization". In: *Computational Aesthetics* (2007), pp. 57–64.

[4] Marek Chrobak and Thomas H Payne. "A linear-time algorithm for drawing a planar graph on a grid". In: *Information Processing Letters* 54.4 (1995), pp. 241–246.

[5] Istvan Fáry. "On straight line representation of planar graphs". In: *Acta Scientiarium Mathematicarum Szeged* 11 (1948), pp. 229–233.

[6] Hubert de Fraysseix, János Pach and Richard Pollack. "How to draw a planar graph on a grid". In: *Combinatorica* 10.1 (1990), pp. 41–51.

[7] Goossen Kant. "Algorithms for drawing planar graphs". PhD thesis. 1993.

[8] William Thomas Tutte. "How to draw a graph". In: *Proceedings of the London Mathematical Society* 3.1 (1963), pp. 743–767.

[9] Norishige Chiba, Tadashi Yamanouchi and Takao Nishizeki. "Linear algorithms for convex drawings of planar graphs". In: *Progress in graph theory* (1984), pp. 153–173.

[10] Pierre Rosenstiehl and Robert E Tarjan. "Rectilinear planar layouts and bipolar orientations of planar graphs". In: *Discrete & Computational Geometry* 1.4 (1986), pp. 343–353.

[11] Walter Schnyder. "Embedding planar graphs on the grid". In: *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 1990, pp. 138–148.

[12] Norishige Chiba, Kazunori Onoguchi and Takao Nishizeki. "Drawing plane graphs nicely". In: *Acta Informatica* 22.2 (1985), pp. 187–201.

[13] Marek Chrobak and Goos Kant. "Convex grid drawings of 3-connected planar graphs". In: *International Journal of Computational Geometry & Applications* 7.03 (1997), pp. 211–223.

[14] John Hopcroft and Robert E Tarjan. "Efficient planarity testing". In: *Journal of the ACM (JACM)* 21.4 (1974), pp. 549–568.