

## Drawing Planar Graphs with Large Vertices and Thick Edges

*Gill Barequet*

Dept. of Computer Science, The Technion—Israel Institute of Technology,  
<http://www.cs.technion.ac.il/~barequet>, [barequet@cs.technion.ac.il](mailto:barequet@cs.technion.ac.il)

*Michael T. Goodrich*

Dept. of Information and Computer Science, Univ. of California, Irvine,  
<http://www.cs.uci.edu/~goodrich>, [goodrich@ics.uci.edu](mailto:goodrich@ics.uci.edu)

*Chris Riley*

Dept. of Computer Science, Johns Hopkins University, Baltimore,  
<http://www.cs.jhu.edu/~chrisr>, [chrisr@cs.jhu.edu](mailto:chrisr@cs.jhu.edu)

### Abstract

We consider the problem of representing size information in the edges and vertices of a planar graph. Such information can be used, for example, to depict a network of computers and information traveling through the network. We present an efficient linear-time algorithm which draws edges and vertices of varying 2-dimensional areas to represent the amount of information flowing through them. The algorithm avoids all occlusions of nodes and edges, while still drawing the graph on a compact integer grid.

| Article Type  | Communicated by | Submitted      | Revised    |
|---------------|-----------------|----------------|------------|
| regular paper | J. Mitchell     | September 2003 | March 2004 |

---

Work by the first author supported in part by an Abraham and Jennie Fialkow Academic Lectureship and by ARO MURI Grant DAAH04-96-1-0013. Work by the second and third authors supported in part by the NSF under Grants CCR-9732300 and PHY-9980044, and by ARO MURI Grant DAAH04-96-1-0013. A preliminary version of this paper appeared in [1]

## 1 Introduction

An important goal of information visualization is presenting information hidden in the structure of a graph to a human viewer in the clearest way possible. Most graph drawing algorithms fulfill this by making visually pleasing drawings that minimize the number of crossings, condense the area, ensure approximately uniform edge lengths, and optimize for many other aesthetics [5]. Without these techniques, the graph may appear “cluttered” and confusing, and difficult to study for a human. Nevertheless, in addition to being aesthetically pleasing, a graph drawing may need to convey additional information beyond connectivity of nodes. In the world of computer science and mathematics, “graphs” are development processes, computer networks, or many other things. In the example of a network, it is often useful to know the amount of traffic traveling across each edge and through each node, in order to visualize network problems like imbalances or Denial-of-Service attacks. Commonly-used graph-drawing algorithms do not handle this sort of additional information and do not have any method for displaying it.

A simple solution that maintains the current drawing of a graph is to label each edge (or node) with a number corresponding to the volume of information passing through (or being generated by or received by) it. Although technically this is a display of the information, it is nevertheless not fully using the visual element of the display. For example, a user would need to individually examine each edge and its label just to select the maximum. Therefore, we believe that visualizing traffic in a network requires that we modify the representation of the nodes and edges to best indicate levels of that traffic.

Before we describe our approach, we would like to first mention some trivial solutions that would require little modification to existing techniques. It would be fairly easy, for example, to simply send animated pulses along an edge with density or rate proportional to the data flow. All we need in this case is space for the pulses to be drawn (since, if edges were too close together, their pulses might be indistinguishable). Nevertheless, this solution does not differentiate volume well (as short high-volume edges might get missed), it requires a dynamic display, and it is potentially confusing.

Another approach that requires a few algorithmic modifications is introducing a chromatic variation in the edges, similar to that used by weather forecasters in Doppler radar images. The two possible implementations of this involve using several distinct color levels and a corresponding key (which does not allow for much variation), or a continuous spectrum of colors. But edges in most graph drawings are thin, and it is not easy to compare two different edges in the continuous scale (particularly for those who are color-blind or color-deficient, which means 8% of all men).

Instead, the approach we advocate in this paper is to differentiate between nodes and edges of varying volume by drawing them in different sizes, possibly augmenting such a display with labels if exact values are needed. This approach is inspired by Minard’s classic graphic of the march of Napoleon’s army in

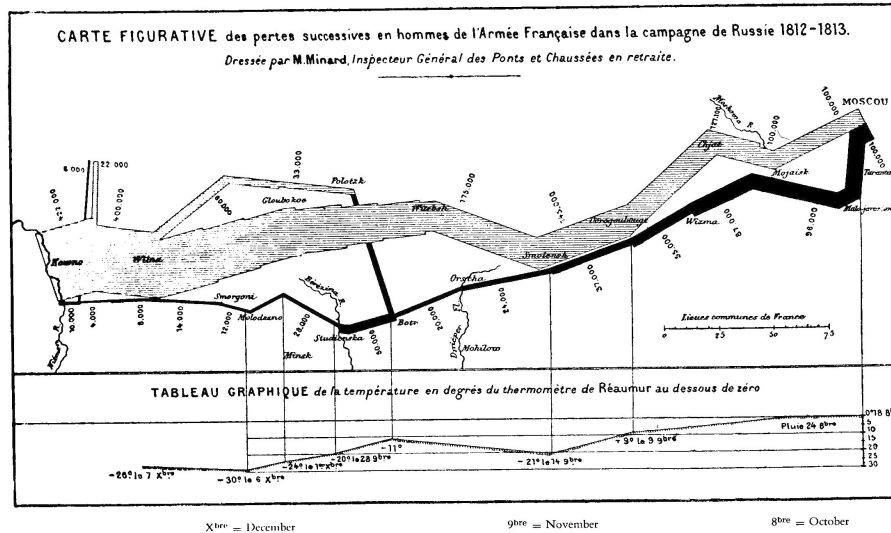


Figure 1: Image taken from Tufte [17], showing the movements of Napoleon’s army in Russia. Edge widths depict army strength, with exact values labeling most edges. Note that this graph has four degree-three vertices and at least 32 edges. Also, two shades are used, with retreating armies shown with solid black edges.

Russia [17, p. 41]<sup>1</sup> (see Figure 1), which geometrically illustrates the army’s movements while using edge widths to depict its strength. Among the benefits of width-based drawings is the fact that they easily separate low- and high-volume nodes and edges, and that they can be depicted on any medium. There is a challenge when using width to represent edge and vertex weights, however, in that increasing edge and vertex sizes introduces the possibility of occlusion of vertices or edges. Such occlusion considerations are not present in other graph drawing problems, which usually consider vertices and edges to be drawn as points and curves, respectively. When we allow vertices and edges to take on a significant two-dimensional area, especially if they are large enough to stand out, then they may obscure each other, which is unacceptable. We therefore need algorithms for drawing graphs with wide edges and large vertices that avoid edge and vertex occlusions.

### 1.1 Standard Approaches and Previous Related Work

One way to avoid occlusions when introducing vertex and edge width is to ensure a sufficiently large edge separation and a bounded angular resolution around vertices. Then, one can scale up the entire drawing and increase the

<sup>1</sup>Attributed to E.J. Marey, *La Méthode Graphique* (Paris, 1885), p. 73.

width of weighted vertices and edges as a proportional fraction of this factor. The easiest approach to performing this scaling is to define a parameter  $w$  as the maximum width of any edge, and expand the drawing output from a bounded-angular resolution algorithm to ensure an edge separation of at least  $w + 1$ . Then edges can be drawn at a weighted proportion of the maximum width  $w$ . The problem with this approach is that it produces a drawing with area  $\Theta(Aw^2)$ , where  $A$  is the original (unweighted) drawing area. We would prefer a method without such a quadratic blow-up in area. Note, in addition, that the overall width and height of a drawing made according to this method would be a multiplicative factor of  $w + 1$  times the width and height of the drawing with an edge separation of 1. Thus, when such a drawing is compressed to fit on a standard display device, the result would be the same as if we took the original algorithm and simply drew the edges wider within the space already allotted to them (up to a width of  $w/(w + 1)$ ), since it would be compressed  $w + 1$  times as much in height and width. Ideally, we would like a weighted graph-drawing algorithm that “shifts” edges and vertices around to make room for edges and vertices of larger widths.

The aesthetics of bounded angular resolution and edge separation have been studied by several researchers (see, e.g., [2, 8, 10, 11, 12, 13, 14, 16]). One significant early result is by Malitz and Papakostas [16], who prove that a traditional straight-line drawing of a planar graph with bounded angular resolution can require area exponential in the complexity of the graph. Goodrich and Wagner [12] describe an algorithm for computing a straight-line drawing of a planar graph on  $n$  vertices with at most two bends per edge on an integer grid in  $O(n^2)$  area with an asymptotically optimal angular resolution upper bound. An improvement to this, by Cheng et al. [2], reduces the maximum to one bend per edge, but the constants in the area bound increase slightly. Both algorithms are based on a classic algorithm by de Fraysseix, Pach, and Pollack [9], which introduces the “canonical ordering” for drawing vertices of a planar graph used in [2, 12] and elsewhere. Their original algorithm produces a planar straight-line drawing of the graph in an  $O(n) \times O(n)$  area, but does not bound angular resolution.

A few works dealt with compaction of graphs with vertices of prescribed sizes; see, e.g., [4, 7, 15]. The only work on drawing graphs with “fat” edges, of which we are aware, is that of Duncan et al. [6]. This work describes a polynomial-time algorithm for computing, given a graph layout, the thickest possible edges of the graph.

## 1.2 Our Results

### 1.2.1 Objective

In this paper we present an algorithm to draw a maximally-planar graph with a given set of edge traffic volumes. The resulting graph fits in an  $O(n + C) \times O(n + C)$  integer grid ( $C$  is the total cost of the network, defined below), with vertices centered at grid points. The algorithm draws nodes as solid diamonds,

but other shapes such as circles could also be used. Edges are drawn as “pipes” of varying size with a minimum separation of one unit at the base of each edge. There are no bends in the drawing, though edges can leave nodes at various angles. The drawing contains no edge crossings or occlusions of nodes or edges.

### 1.2.2 Results

One of the main advantages of our algorithm is that it benefits from the disparity between low and high volume levels in the weights of different edges and nodes. Intuitively, our algorithm uses this disparity to take up less space for drawing edges and nodes, when possible. As the upper limit for the traffic on an edge, we use the *capacity* of that edge, and we upper bound the sum of the capacities of adjacent edges as the capacity of a node. We assume that traffic information is supplied as a normalized list of edge thicknesses in the range  $[0..w]$ , for some parameter  $w$  (an edge of width 0 would be considered to have been added to make the graph maximally planar and would not be included in the final drawing). For the graph layout, we consider edge weights to be integers, though in the rendering stage edges can easily be drawn with noninteger width within the integer space allocated to them (and in fact can be drawn with dynamic values changing over time, as long as they are less than the capacity). Denote the degree of a node  $v$  by  $d(v)$ . Define the thickness or *cost* of an edge  $e$  to be  $c(e)$ , and the size or weight of a node  $v$  to be  $w(v) = \sum c(e)$  for all edges adjacent to  $v$ . We assign cost 0 to edges added to the given graph in order to augment it to a maximally-planar graph. Let  $C = \sum_e c(e)$  be the total cost of the network. As mentioned above, our algorithm draws a weighted planar graph with edge- and vertex-widths proportional to their weights in an  $O(n + C) \times O(n + C)$  integer grid. Thus, the total area is  $O(n^2 + C^2)$ . Note that if  $w$  denotes the maximum width of an edge in a given graph  $G$ , then the area of our drawing of  $G$  is never more than  $O(n^2 w^2)$ , since  $C$  is  $O(nw)$  in a planar graph. Moreover, the area of one of our drawings can be significantly below the corresponding  $O(n^2 w^2)$  upper bound for the naive approach. For example, if  $C$  is  $O(w)$ , then the area of our drawing is  $O(n^2 + w^2)$ , and even if  $C$  is  $O(n + wn^{0.5})$ , then the area is still at most  $O(n^2 + nw^2)$ .

### 1.2.3 Extensions

Our algorithm supports only planar graphs, which prevents us from directly applying it to many graphs derived from real-world processes. It is not possible to achieve our strict goals with nonplanar graphs, as some edges must cross (and thus one must be occluded). We can extend our model to support occlusions for the purposes of edge crossings and then can consider extensions of our algorithm that allow these but prevent all other node and edge occlusions. Since the canonical ordering that underlies our algorithm requires a maximally-planar graph, we cannot use these techniques without first using planarization. One obvious planarization technique is continually removing edges until the graph is planar and then reinserting the edges after the completion of the algorithm.

However, this is insufficient since it is not possible, in general, to ensure that the reinsertion of the additional edges will not cause unnecessary occlusions. Adding dummy vertices to the graph to represent edge intersections would avoid this issue, though this would create bends in the edges (since the dummy vertices would not be drawn).

It is likely that the extension of the canonical ordering of de Fraysseix et al. [9] presented by Kant [13] (which assumes that the input graph is triconnected rather than maximally planar) could be used with our algorithm. For sparse graphs, this could allow some improvement in the visual appeal of the final drawing, since fewer edges would need to be added. Too many such edges can cause large gaps in the final drawing when they are removed.

## 2 The Algorithm

Suppose we are given a maximally-planar graph  $G$  with  $n$  vertices and integer weights in the range  $[0, w]$  assigned to its edges. Our algorithm for drawing  $G$  is as follows. Order the vertices of a maximally-planar graph  $v_1, v_2, \dots, v_n$  according to their canonical ordering [9]. The following are then satisfied, for all  $k \geq 3$ :

1. For the graph  $G_k$  restricted to the first  $k$  vertices in the canonical ordering,  $G_k$  is biconnected (internally triconnected), and the cycle  $C_k$  of the external vertices of  $G_k$  contains  $(v_1, v_2)$ .
2. The vertex  $v_{k+1}$  is in the exterior face of  $G_{k+1}$  and has at least two neighbors in  $G_k$ , all of which are consecutive on  $(C_k - (v_1, v_2))$ . These are the only neighbors of  $v_{k+1}$  in  $G_k$ .

Such an ordering exists for every maximally-planar graph and can be constructed in linear time (see, e.g., [3, 9]). Figure 2 shows a sample graph with the canonical ordering of its vertices.

Let us define a structure called a *hub* around each vertex (see Figure 3). This is a diamond-shaped area with corners  $w(v) + d(v)$  unit spaces above, below, left, and right of the vertex, similar to the *join box* of [12]. The diagonal of each unit square along the perimeter of the hub (see Figure 4) is called a *slot*, and a collection of sequential slots used by a single edge is called a *port*. At insertion time each edge is allocated a port containing one slot per unit cost (if zero-cost edges are allowed, then the edge is drawn at the boundary between two slots), leaving a free slot between edges.

In order to show that an edge separation of at least 1 is guaranteed, we maintain the following invariants (adapted from the invariants in [12]) that must be met for all  $G_k$ :

1. The vertices and slot boundaries of  $G_k$  are located at lattice points (have integer coordinates).
2. Let  $c_1 = v_1, c_2, c_3, \dots, c_m = v_2$  (for some  $m = m(k)$ ) be the vertices along the exterior cycle  $C_k$  of  $G_k$ . Then the  $c_j$ 's are strictly increasing in  $x$ .

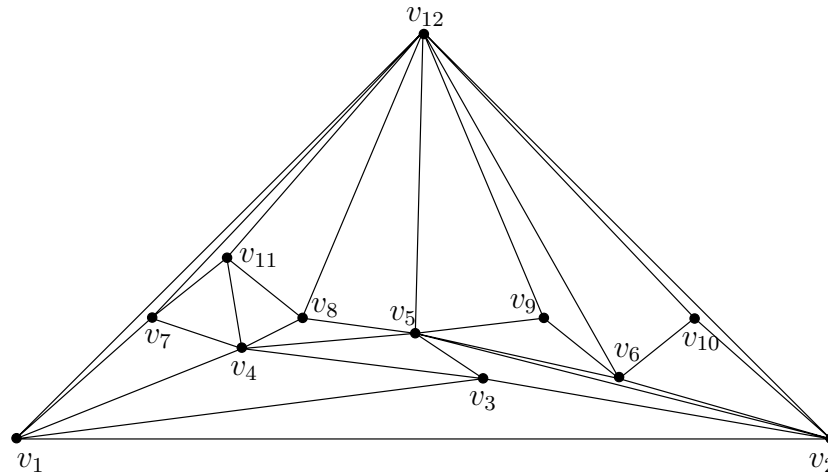


Figure 2: A sample canonical ordering.

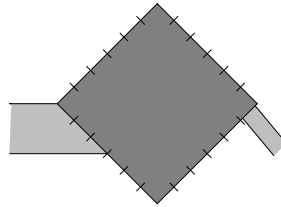


Figure 3: A sample hub with a pair of edges.

3. All edges between slots of  $c_1, c_2, \dots, c_m$  have slope  $+1$  or  $-1$ , with the exception of the edge between  $v_1$  and  $v_2$ , which has slope  $0$ .
4. For each  $v \notin \{v_1, v_2\}$  in  $G_k$ , the slots with the left and right corners as their top boundaries have been used. Also, any slots used in the upper half of the hub are consecutive above either the left or right corner (with a space left in between), except for the slot used by the final edge when a node is *dominated* (see Section 2.2.3).
5. Each edge is monotone in both  $x$  and  $y$ .

These invariants are also reminiscent of the original canonical-order invariants.

## 2.1 Geometry

There are a few geometric issues regarding drawing thick edges out from a diamond-shaped box. Here we are focusing on the drawing of the edges outside the hub, since we intend to draw the entire hub solid as a node in the final graph. The perimeter length allocated to an edge of thickness  $t \in \mathbf{Z}$  is actually

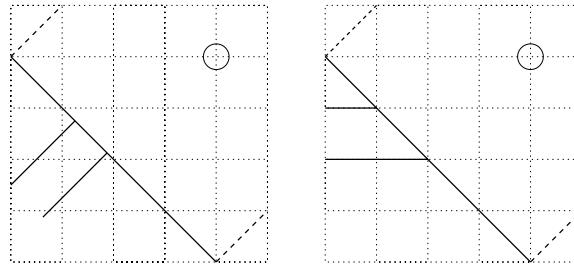


Figure 4: An edge of width 1 using minimum and maximum perimeter space. Note that if the entry angle were shallower than the right image, the edge would no longer be monotone, since once inside the hub it needs to go up to reach the center.

$t\sqrt{2}$  since it is the diagonal of a square of side length  $t$ . This may be necessary, though, as the perimeter space needed by an edge can vary based on the angle it makes with the side of the hub. Thanks to the monotonicity of edge segments (condition 5), the allocated length is sufficient to draw the edge, since the angle made between the incoming edge segment and the side of the hub is at least  $\pi/4$ , meaning the intersection segment in the unit square is of length at most  $1/\cos(\pi/4) = \sqrt{2}$  (see Figure 4). Because of this, we also do not need to concern ourselves with bends in the edges, as we can simply not draw the interior portion. We only need to draw the segment between hubs, making sure that it is at the correct angle when it leaves the node. If an edge does not need the full space, simply use the center of the allocated port.

The definition of monotonicity in traditional graph representations is easily extended to monotonicity in graphs with two-dimensional representations of nodes and edges. Consider extending edges by redirecting the edge from its port to the center of the node and narrowing it to a point; then ensure that the one-dimensional lines bounding each edge are monotone. If and only if this is true, we will say that the thick edge is monotone.

It is also possible to draw the nodes as circular in shape, by using any circle centered within the diamond. This is a simple implementation detail: bend the edges at the segment of the hub, and narrow them as they approach the node. This can be accomplished by bending the sides of the edge differently, pointing each towards the center of the node (Figure 5).

The above proves the following lemma:

**Lemma 1** *If the five conditions listed above are maintained, then a port containing one slot per integer thickness of an edge is sufficient to draw the edge at its thickness, regardless of its incoming angle, without occluding other adjacent edges.*

## 2.2 The Construction

We now describe the *incremental* construction of the graph.



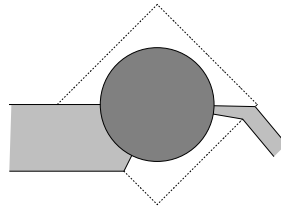


Figure 5: The hub of Figure 3 drawn with a circular vertex.

### 2.2.1 First two vertices

Refer to Figure 6. Build the canonical ordering and place the center of node  $v_1$  at the origin of a 2-dimensional  $x, y$  graph. Center  $v_2$  at  $(x, 0)$  where  $x = w(v_1) + d(v_1) + 1 + w(v_2) + d(v_2)$ . Our nodes are drawn solid as the entire hub, so this placement of  $v_2$  creates the minimum acceptable separation of one unit between the right corner of  $v_1$  and the left corner of  $v_2$ . This graph,  $G_2$ , clearly maintains the five conditions (conditions 3 and 4 are trivial with only two nodes).

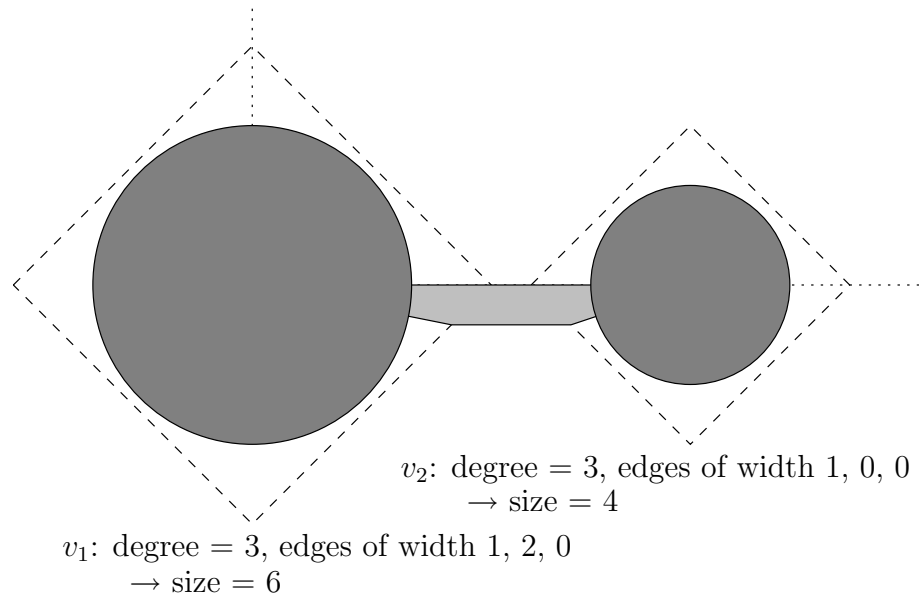


Figure 6: Sample graph  $G_2$ .

### 2.2.2 Inserting $v_3$

Refer to Figure 7. By the properties of the canonical ordering,  $v_3$  must have

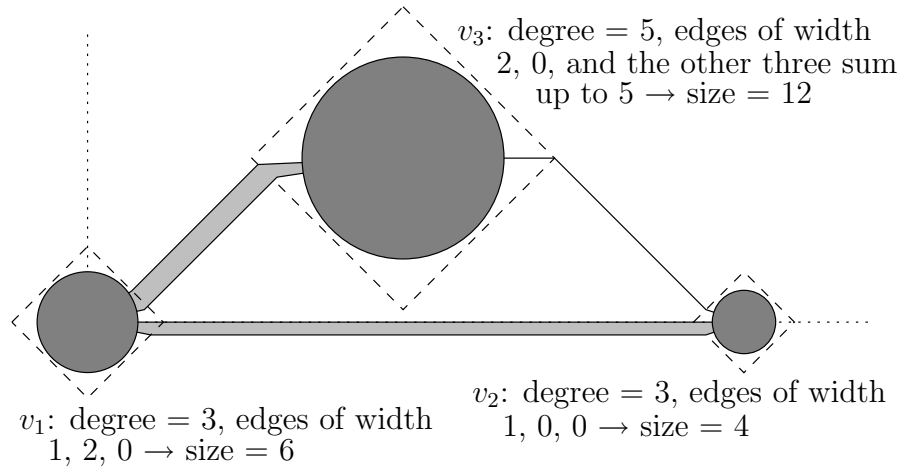


Figure 7: Sample graph  $G_3$ .

edges to  $v_1$  and  $v_2$ . Use the lowest slots available on the appropriate segments of  $v_1$  and  $v_2$  (upper-right for  $v_1$ , upper-left for  $v_2$ ) and the slots in  $v_3$  whose top points are the left and right corners. Shift  $v_2$  horizontally to the right to allow the edges to be drawn at the correct slopes and to allow  $v_3$  to be drawn without occluding edge  $(v_1, v_2)$ . Set  $v_3$  at height  $h = 2 * (w(v_3) + d(v_3))$ . The top of the edge  $(v_1, v_2)$  is at  $y = 0$ , so the top of  $v_3$  must be at  $y = h + 1$  to clear it. The top of  $v_3$  is also the intersection of the lines of slope +1 and -1 drawn from the tops of the ports allocated to the edges  $(v_1, v_3)$  and  $(v_2, v_3)$  on  $v_1$  and  $v_2$ , respectively. Since we are dealing with lines of slope  $\pm 1$ , starting from even integer grid points (as assured for  $v_2$ , see below), their intersection is an integer grid point.

We need the intersection of the lines from these two ports to be at height  $h + 1$ . This requires that their  $x$ -coordinates (if extended to the line  $y = 0$ ) be  $2h + 2$  units apart. The actual necessary distance between  $v_1$  and  $v_2$  is  $(2h + 2) - (2 * (c((v_1, v_3)) + 1)) - (2 * (c((v_2, v_3)) + 1))$ , where  $c((u, v))$  is the cost of edge  $(u, v)$ . Shift  $v_2$  right one unit less than this (since it is currently one unit to the right).

The case of inserting  $v_3$  should be handled separately because it is the only situation where the top boundary of the initial graph contains edges not of slope  $\pm 1$ . We will generalize to handle the remaining cases.

### 2.2.3 Induction

Refer to Figure 8. Assume as an inductive hypothesis that the graph  $G_k$  maintains the five conditions and has an edge separation of 1 between all edges. We now need to insert vertex  $v_{k+1}$  and its incident edges to  $G_k$ . Let  $c_l, c_{l+1}, \dots, c_r$  be the neighbors of  $v_{k+1}$  in  $G_{k+1}$ . By the properties of the canonical ordering these neighbors are sequential along the outer face of  $G_k$ . Before inserting  $v_{k+1}$ ,

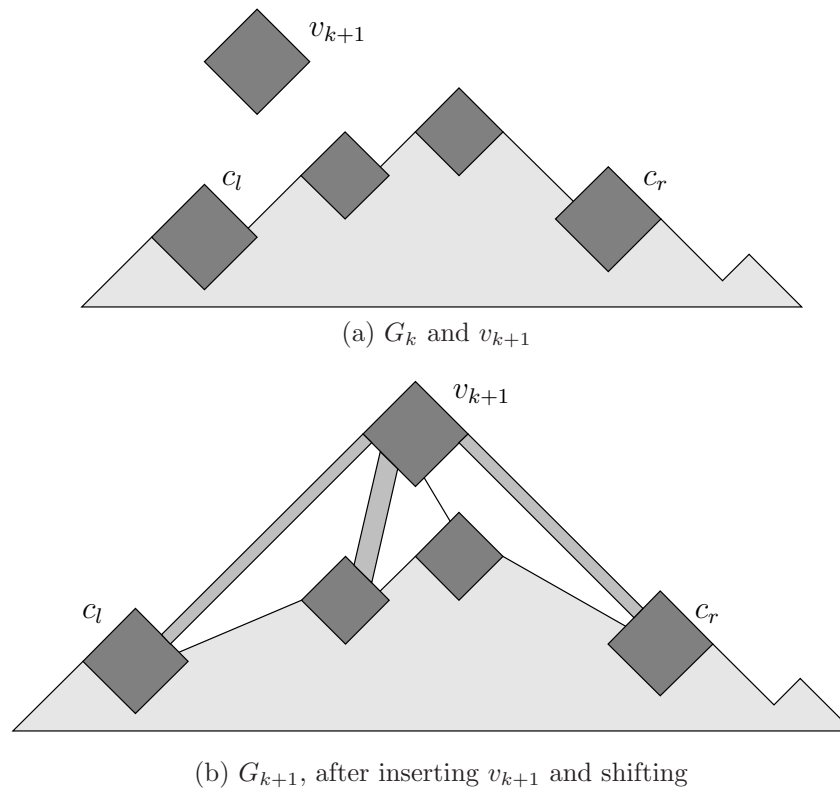


Figure 8: Induction on the number of nodes in the graph.

we need to make room for it and its edges to be drawn, and to ensure that the five conditions are still maintained for  $G_{k+1}$ . In order to do this, we shift the vertices along the exterior cycle  $C_k$  to the right. We also need to shift vertices in the interior portion of the graph to preserve planarity and to prevent occlusions. The remainder of this section discusses these shifts.

A node  $u$  is *dominated* when it is one of the neighbors of  $v_{k+1}$  in  $G_k$  other than  $c_l$  or  $c_r$ . A dominated node  $u$  has used its last edge (since it is an interior node in  $G_{k+1}$  and therefore additional edges would make  $G_{k+1}$  nonplanar), and is included in the shifting set of  $v_{k+1}$  (see below), so any slots remaining on  $u$  can be used to connect to  $v_{k+1}$  without creating edge crossings or occlusions in the shifting process. This enables edge  $(u, v_{k+1})$  to select a port on  $u$  to maintain monotonicity.

#### 2.2.4 Shifting sets

The paper by de Fraysseix et al. [9] outlines the concept of shifting sets for each vertex on the outer cycle  $C_k$  of  $G_k$ , which designate how to move the interior vertices of the graph. We will use the same concept in our algorithm. The

shifting set  $M_k(c_i)$  for all  $c_i$  ( $1 \leq i \leq m$ ) on  $C_k$  contains the set of nodes to be moved along with  $c_i$  to avoid edge crossings and occlusions. Define the  $M_k$ 's recursively, starting with  $M_3(c_1 = v_1) = \{v_1, v_2, v_3\}$ ,  $M_3(c_2 = v_3) = \{v_2, v_3\}$ ,  $M_3(c_3 = v_2) = \{v_2\}$ . Then, for the shifting sets used in  $G_{k+1}$ , let:

- $M_{k+1}(c_i) = M_k(c_i) \cup \{v_{k+1}\}$  for  $i \leq l$ ;
- $M_{k+1}(v_{k+1}) = M_k(c_{l+1}) \cup \{v_{k+1}\}$ ;
- $M_{k+1}(c_j) = M_k(c_j)$  for  $j \geq r$ .

The sets obey the following properties for all  $k$ :

1.  $c_j \in M_k(c_i)$  if and only if  $j \geq i$ ;
2.  $M_k(c_1) \supset M_k(c_2) \supset M_k(c_3) \supset \dots \supset M_k(c_m)$ ;
3. For any nonnegative numbers  $\alpha_i$  ( $1 \leq i \leq m$ ), sequentially shifting  $M_k(c_i)$  right by  $\alpha_i$  maintains planarity,<sup>2</sup> and does not introduce any edge or node occlusions.

The proofs of the first two properties are found in [9]. For the third, it is clearly true for the base case  $k = 3$ . Consider the graph  $G_{k+1}$ ,  $v_{k+1}$ , and the vertices  $c_1, c_2, \dots, c_m$  along the cycle  $C_k$  of the exterior face of  $G_k$ . Let us fix shift amounts  $\alpha(c_1), \alpha(c_2), \dots, \alpha(c_l), \alpha(v_{k+1}), \alpha(c_r), \dots, \alpha(c_m)$  corresponding to the vertices along the cycle  $C_{k+1}$ . The graph under the cycle  $C_k$  satisfies the condition by induction:

1. Set  $\alpha(c_{l+1}) = 1 + 2 * (w(v_{k+1}) + d(v_{k+1})) + \alpha(v_{k+1})$  (the sum of the first two terms is the amount  $c_{l+1}$  will be shifted when  $v_{k+1}$  is inserted, and the last term is how much  $c_{l+1}$  and nodes in its shifting set will be shifted because of the shifting of  $v_{k+1}$ );
2. Set all other interior  $\alpha$ 's ( $\alpha(c_{l+2})$  through  $\alpha(c_{r-1})$ ) to 0; and
3. Set the exterior  $\alpha$ 's ( $\alpha(c_1), \dots, \alpha(c_l)$  and  $\alpha(c_r), \dots, \alpha(c_m)$ ) to their above values.

The portion of the graph above  $C_k$ , with the exception of the edges  $(c_l, v_{k+1})$  and  $(c_r, v_{k+1})$ , is shifted in a single block with  $v_{k+1}$ . The edge  $(c_l, v_{k+1})$  cannot be forced to occlude or intersect the next edge,  $(c_{l+1}, v_{k+1})$ , since the latter edge can only be pushed farther away, moving along with the former when it shifts. Similarly,  $(c_{r-1}, v_{k+1})$  cannot occlude or intersect  $(c_r, v_{k+1})$  (see Figure 8(b)).

This proves the following lemma:

**Lemma 2** *For all  $G_k$ , sequentially shifting the nodes in the shifting sets of each node in the exterior cycle of  $G_k$  by any nonnegative amount cannot create edge crossings or node or edge occlusions.*

---

<sup>2</sup>This property of the shifting sets is stronger than what we need. Our algorithm performs only two shifts per iteration.

### 2.2.5 Shifting and placement

Similar to [2], we will shift twice. First, shift  $M_k(c_{l+1})$  by the width of node  $v_{k+1}$  plus 1, which is  $2 * (w(v_{k+1}) + d(v_{k+1})) + 1$ . Also shift  $M_k(c_r)$  by the same amount. (To ensure that  $c_r$  and  $c_l$  are separated by an even amount of units, shift  $M_k(c_r)$  by one more unit if necessary.) The intuition behind this is simple. We cannot allow node  $v_{k+1}$  to occlude any portion of  $G_k$ . Since the graph could rise as high in  $y$  as half the distance between  $c_l$  and  $c_r$  in  $x$ , placing  $v_{k+1}$  at the intersection of the edges of slope  $\pm 1$  from these nodes could place it on top of another vertex. Separating  $c_l$  and  $c_r$  by  $(2 \text{ or } 3) + 4 * (w(v_{k+1}) + d(v_{k+1}))$  moves  $v_{k+1}$  half that much higher, allowing it to clear the graph.

Now that we have sufficiently shifted all nodes in  $G_k$ , we can place  $v_{k+1}$ . Define  $l_1$  (resp.,  $l_2$ ) as the line of slope  $+1$  (resp.,  $-1$ ) from the top of the port of  $c_l$  (resp.,  $c_r$ ) allocated to the edge  $(c_l, v_{k+1})$  (resp.,  $(c_r, v_{k+1})$ ). Select the ports of  $c_l$  and  $c_r$  that maintain condition 4's requirement of minimum separation between edges. If the top corner of  $v_{k+1}$  is placed at the intersection of  $l_1$  and  $l_2$ , all the edges between  $v_{k+1}$  and nodes in  $C_k$  can be drawn monotonically in  $x$  and  $y$  without creating occlusions. Note also that this placement of  $v_{k+1}$  assigns the edge  $(c_l, v_{k+1})$  to the port whose top is the left corner of  $v_{k+1}$ , and likewise  $(c_r, v_{k+1})$  is assigned to the port at the right corner of  $v_{k+1}$ . These edges are clearly monotone. Monotonicity for the new interior edges is ensured by selecting a port from the side of the  $v_{k+1}$  facing the target node, and a port from the target node facing  $v_{k+1}$ . Since each of the four sides of every node is of size  $d(v) + w(v)$ , ports can be chosen on arbitrary sides (maintaining condition 4, of course), and sufficient space for the edge is guaranteed. Also, since the edges are at least a distance of 1 apart on  $v_{k+1}$ , and their destination ports are all on different nodes, each of which are at least a unit apart in  $x$ , no occlusions or intersections can be created.

By the third property of the shifting sets, this movement cannot cause edge occlusions or intersections. However, it remains to show that the graph maintains the five conditions listed above. The first is trivially true since everything is shifted by integer values. Likewise the second is true, since  $v_{k+1}$  is inserted between  $c_l$  and  $c_r$ , and each node is shifted at least as much to the right as the node before it, so their ordering remains intact. Since the edges before  $c_l$  and after  $c_r$  have not been changed (both endpoints of each have been moved by the same amounts), and the edges  $(c_l, v_{k+1})$  and  $(c_r, v_{k+1})$  were inserted at slopes of  $\pm 1$ , condition 3 is still true. Monotonicity is maintained regardless of any horizontal shifting, so the edges of  $G_k$  remain monotone. The outside edges  $(c_l, v_{k+1})$  and  $(c_r, v_{k+1})$  are clearly monotone, and the interior edges were assigned ports on each node to make them monotone.

When  $v_{k+1}$  is inserted, its left- and rightmost neighbors on  $C_k$  are assigned the slots whose tops are at the left and right corner, thus maintaining the first portion of condition 4. The rest is maintained by selecting the correct ports of  $c_l$ ,  $c_r$ , and the interior nodes. Such ports must be available at every node, since each side of a node is large enough to support every edge adjacent to it. Thus, graph  $G_{k+1}$  meets all conditions and has a minimum edge separation of 1.

### 2.3 Analysis

After inserting all vertices, the graph  $G$  still maintains the five conditions, and thus is planar, without crossings or occlusions, and has an edge separation of at least 1. The question of angular resolution is not necessarily relevant, since most or all of the hub area is drawn as a solid node for significance. But if one extended the edges to a point node at the center of the hub, then the boundary lines of the edges have a minimum angular resolution of  $O(1/(w(n) + d(n)))$  for all nodes (see Figure 9).

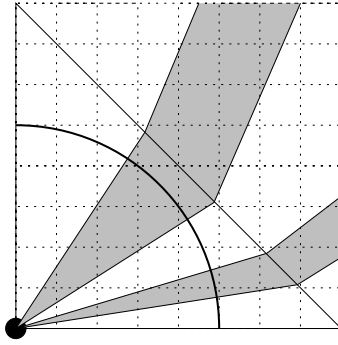


Figure 9: The upper-right quadrant of a node.

We also would like a well-bounded area for the complete drawing of  $G$ .

**Theorem 1** *The area of the grid necessary to draw the graph is  $O(n + C) \times O(n + C)$ , where  $C$  is the total cost of the network, defined as  $C = \sum_e c(e)$  for a given input set of edge costs  $c(e)$ .*

**Proof:** Since  $G$  is drawn within the convex hull of  $v_1$ ,  $v_2$ , and  $v_n$ , the width is equal to the distance between the left corner of  $v_1$  and the right corner of  $v_2$ . This initial distance at  $G_2$  is 1 plus the widths of  $v_1$  and  $v_2$ . Shifting all  $v_i$  for  $i \geq 4$  moves  $v_2$  to the right (and increases the width of  $G$ ) by at most  $3 + 4 * (w(v_i) + d(v_i))$ , and the insertions of  $v_1$  through  $v_3$  can be upper bounded by this. Therefore, the width of the drawing is bounded above by  $\sum_{i=1}^n (3 + 4 * w(v_i) + 4 * d(v_i)) = 3n + 8C + 8|E|$ , where  $E$  is the set of edges in the graph. Since in any planar graph  $|E| \leq 3n - 6$ , the width is bounded above by  $27n + 8C$ . The resulting drawing is approximately an isosceles triangle with slope  $\pm 1$  (approximately since the edges begin below the peak of  $v_1$  and  $v_2$ , thus slightly lowering the top of the triangle). The height, therefore, would be bounded by  $14n + 4C$ , except that the nodes  $v_1$  and  $v_2$  extend below the graph by half their height, and this height is not previously accounted for as it is outside the triangle. Therefore, the bound on the height of the drawing is  $14n + 4C + \max(w(v_1) + d(v_1), w(v_2) + d(v_2))$ . The  $\max()$  term is bounded

above by  $n + 2C$ . The final upper bound is thus  $15n + 6C$ , and the theorem holds.  $\square$

For running time analysis, we refer the reader to the  $O(n)$  time implementation of the algorithm of de Fraysseix et al. [9] by Chrobak and Payne [3]. This solution can be extended so as to implement our algorithm without changing the asymptotic running-time complexity.

See Figure 10 for a sample drawing of a weighted version of Figure 2.

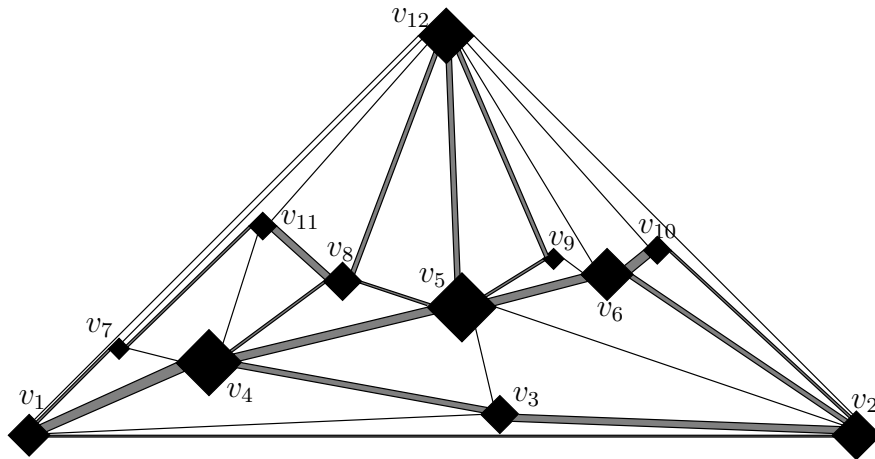


Figure 10: A sample graph drawn by our method.

The edge weights used are the following:

| Edge     | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $v_6$ | $v_7$ | $v_8$ | $v_9$ | $v_{10}$ | $v_{11}$ | $v_{12}$ |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|----------|----------|
| $v_1$    | -     | 1     | 0     | 5     | -     | -     | 1     | -     | -     | -        | -        | -        |
| $v_2$    | 1     | -     | 4     | -     | 0     | 2     | -     | -     | -     | 1        | -        | -        |
| $v_3$    | 0     | 4     | -     | 3     | 0     | -     | -     | -     | -     | -        | -        | -        |
| $v_4$    | 5     | -     | 3     | -     | 4     | -     | 0     | 1     | -     | -        | 0        | -        |
| $v_5$    | -     | 0     | 0     | 4     | -     | 4     | -     | 1     | 1     | -        | -        | 3        |
| $v_6$    | -     | 2     | -     | -     | 4     | -     | -     | -     | 0     | 4        | -        | -        |
| $v_7$    | 1     | -     | -     | 0     | -     | -     | -     | -     | -     | -        | 1        | -        |
| $v_8$    | -     | -     | -     | 1     | 1     | -     | -     | -     | -     | -        | 3        | 2        |
| $v_9$    | -     | -     | -     | -     | 1     | 0     | -     | -     | -     | -        | -        | 2        |
| $v_{10}$ | -     | 1     | -     | -     | -     | 4     | -     | -     | -     | -        | -        | -        |
| $v_{11}$ | -     | -     | -     | 0     | -     | -     | 1     | 3     | -     | -        | -        | -        |
| $v_{12}$ | -     | -     | -     | -     | 3     | -     | -     | 2     | 2     | -        | -        | -        |

The induced vertex sizes are then:

| Vertex Size | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $v_6$ | $v_7$ | $v_8$ | $v_9$ | $v_{10}$ | $v_{11}$ | $v_{12}$ |
|-------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|----------|----------|
|             | 11    | 13    | 11    | 19    | 20    | 14    | 5     | 11    | 6     | 7        | 7        | 10       |

### 3 Future Work

There are many possibilities for future related work:

- Combine awareness of edge thicknesses with force-directed graph drawing techniques by modifying the forces of nodes and edges according to their individual weights in order to 'make room' for them to be drawn larger.
- Establish an asymptotic lower bound on the area necessary to draw a graph with edge thickness as used in our paper. Node size can be reduced as long as the perimeter is of sufficient length to support all edges with a bounded separation. It is possible such a drawing could be done in  $o((n + C)^2)$  area.
- Allow general graphs and edge crossings when necessary, but still use thick edges and large nodes and prevent occlusions, except in edge crossings.
- Combine the algorithms above with graph clustering techniques to represent potentially very large networks. One could add the sizes of nodes and edges clustered together. It could also be useful to represent the amount of information flowing within a cluster node in addition to between the nodes.
- Extend to three dimensions. The algorithm used here would not extend well, but drawings of graphs in three dimensions with thick edges and large nodes could be useful. Projections of such a graph to two dimensions would not be aesthetic.
- Study common network traffic patterns to optimize the algorithm based on real world data.



## References

- [1] G. Barequet, M. T. Goodrich, and C. Riley. Drawing graphs with large vertices and thick edges, *Proc. 8th Workshop on Algorithms and Data Structures*, Ottawa, Ontario, Canada, *Lecture Notes in Computer Science*, 2748, Springer-Verlag, 281–293, 2003.
- [2] C. Cheng, C. Duncan, M. T. Goodrich, and S. Kobourov. Drawing planar graphs with circular arcs, *Discrete & Computational Geometry*, 25 (2001), 405–418.
- [3] M. Chrobak and T. Payne. A linear-time algorithm for drawing planar graphs, *Information Processing Letters*, 54 (1995), 241–246.
- [4] G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Orthogonal and quasi-upward drawings with vertices of prescribed size, *Proc. 7th Int. Symp. on Graph Drawing*, Prague, Czech, *Lecture Notes in Computer Science*, 1731, Springer-Verlag, 297–310, 1999.
- [5] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, Upper Saddle River, NJ, 1999.
- [6] C. A. Duncan, A. Efrat, S. G. Kobourov, and C. Wenk. Drawing with fat edges, *Proc. 9th Int. Symp. on Graph Drawing*, Vienna, Austria, *Lecture Notes in Computer Science*, 2265, Springer-Verlag, 162–177, 2001.
- [7] M. Eiglsperger and M. Kaufmann. Fast Compaction for orthogonal drawings with vertices of prescribed size, *Proc. 9th Int. Symp. on Graph Drawing*, Vienna, Austria, *Lecture Notes in Computer Science*, 2265, Springer-Verlag, 124–138, 2001.
- [8] M. Formann, T. Hagerup, J. Haralambides, M. Kaufmann, F. T. Leighton, A. Simvonis, E. Welzl, and G. Woeginger. Drawing graphs in the plane with high resolution, *SIAM J. of Computing*, 22 (1993), 1035–1052.
- [9] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid, *Combinatorica*, 10 (1990), 41–51.
- [10] E. R. Gansner, E. Koutsofios, S. C. North, and K. P. Vo. A technique for drawing directed graphs, *IEEE Trans. on Software Engineering*, 19 (1993), 214–230.
- [11] A. Garg and R. Tamassia. Planar drawings and angular resolution: Algorithms and bounds, *Proc. 2nd Ann. European Symp. on Algorithms*, Utrecht, The Netherlands, *Lecture Notes in Computer Science*, 855, Springer-Verlag, 12–23, 1994.

- [12] M. T. Goodrich and C. Wagner. A framework for drawing planar graphs with curves and polylines, *Proc. 6th Int. Symp. on Graph Drawing*, Montreal, Québec, Canada, *Lecture Notes in Computer Science*, 1547, Springer-Verlag, 153–166, 1998.
- [13] G. Kant. Drawing planar graphs using the canonical ordering, *Algorithmica*, 16 (1996), 4–32.
- [14] G. W. Klau and P. Mutzel. Optimal compaction of orthogonal grid drawings, *Proc. 7th Int. Integer Programming and Combinatorial Optimization Conf.*, Graz, Austria, *Lecture Notes in Computer Science*, 1610, Springer-Verlag, 304–319, 1999.
- [15] G. W. Klau and P. Mutzel. Combining graph labeling and compaction, *Proc. 7th Int. Symp. on Graph Drawing*, Prague, Czech, *Lecture Notes in Computer Science*, 1731, Springer-Verlag, 27–37, 1999.
- [16] S. Malitz and A. Papakostas. On the angular resolution of planar graphs, *SIAM J. of Discrete Mathematics*, 7 (1994), 172–183.
- [17] E. R. Tufte. *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, CT, 1983.