Polar Coordinate Drawing of Planar Graphs with Good Angular Resolution

Christian A. Duncan[1] and Stephen G. Kobourov[2]

[1] Department of Computer Science
University of Miami
Coral Gables, FL 33124
duncan@cs.miami.edu

[2] Department of Computer Science
University of Arizona
Tucson, AZ 85721
kobourov@cs.arizona.edu

Abstract. We present a novel way to draw planar graphs with good angular resolution. We introduce the polar coordinate representation and describe a family of algorithms which use polar representation. The main advantage of using a polar representation is that it allows us to exert independent control over grid size and bend positions. Polar coordinates allow us to specify different vertex resolution, bend-point resolution and edge separation. We first describe a standard (Cartesian) representation algorithm (CRA) which we then modify to obtain a polar representation algorithm (PRA). In both algorithms we are concerned with the following drawing criteria: angular resolution, bends per edge, vertex resolution, bend-point resolution, edge separation, and drawing area.

The CRA algorithm presented here achieves 1 bend per edge, unit vertex resolution, unit bend resolution, $\sqrt{2}/2$ edge separation, $5n \times \frac{5n}{2}$ drawing area and $\frac{1}{2d(v)}$ angular resolution, where $d(v)$ is the degree of vertex v. These simultaneous bounds are an improvement over all previously known results. The PRA algorithm has an improved angular resolution of $\frac{\pi}{4d(v)}$, 1 bend per edge, and unit vertex resolution. For the PRA algorithm, the bend-point resolution and edge separation are parameters that can be modified to achieve different types of drawings and drawing areas. In particular, for the same parameters as the CRA algorithm (unit bend-point resolution and $\sqrt{2}/2$ edge separation), the PRA algorithm creates a drawing of size $9n \times \frac{9n}{2}$.

1 Introduction

In the area of planar graph drawing there has been considerable interest in algorithms that produce readable drawings [4]. Among the many properties which contribute to the readability of planar graphs, edge smoothness, vertex resolution, bend-point resolution, angular resolution, and edge separation are of great importance. Edges are often drawn as straight-line segments connecting two vertices. An edge can also be drawn as a sequence of straight-line segments, in which case the smallest number of bends is desirable. An edge may also be drawn as a smooth curve. These three types of edges generally provide aesthetically pleasing drawings.

1.1 Definitions

A graph drawing has good *vertex resolution* if vertices cannot get arbitrarily close to one another, that is, if vertices are well distributed in the drawing. As a result, a great deal of research has been concentrated on graph drawing algorithms which place vertices on the integer grid such that the *drawing area* is proportional to the number of vertices n of the graph, typically $O(n) \times O(n)$. If there are bends in the edges, then the bend-points are also placed on the integer grid. The *bend-point resolution* of a graph refers to the minimum distance between two bends. The *edge separation* of a graph refers to the minimum distance between two edges that are sufficiently away from their endpoints (since incident edges can get arbitrarily close to each other near their common endpoint).

A graph drawing has good *angular resolution* if adjacent edges cannot form arbitrarily small angles. This is achieved by ensuring that the edges emanating from a given vertex "fan out" evenly around the vertex. Note, however, that good angular resolution cannot always be achieved while simultaneously guaranteeing straight-line edges and small sub-exponential drawing area [9]. By introducing bends in the edges, however, we can guarantee both good resolution and small drawing area.

1.2 Previous Work

Garg and Tamassia [5] consider the problem of drawing with good angular resolution, and Kant [8] shows how to create drawings with angular resolution of $\Theta(1/d(v))$ in an $O(n) \times O(n)$ area grid, using edges with at most three bends each. Gutwenger and Mutzel [7] describe an improved algorithm with better constant factors which produces very aesthetically pleasing drawings in a $(2n-5) \times (3n/2 - 7/2)$ grid with at least $2/d(v)$ angular resolution using at most three bends per edge. The algorithm of Goodrich and Wagner [6] requires one less bend per edge and guarantees angular resolution of $\Theta(1/d(v))$ for each vertex v, but at the expense of larger area, $(20n-48) \times (10n-24)$. Cheng, Duncan, Goodrich, and Kobourov [1] improve the above algorithm so that every edge has at most one bend while the angular resolution is $\Theta(1/d(v))$ for each vertex v and maximum area is $30n \times 15n$.

1.3 Our Results

We first present a new Cartesian representation algorithm (CRA) which achieves better simultaneous bounds than previous algorithms. In particular, CRA guarantees 1 bend per edge, unit vertex resolution, unit bend-point resolution, $\sqrt{2}/2$ edge separation, $5n \times \frac{5n}{2}$ drawing area, and $\frac{1}{2d(v)}$ angular resolution, where $d(v)$

is the degree of vertex v. These simultaneous bounds are an improvement over all previously known results.

We then present a novel polar representation algorithm (PRA). The PRA algorithm also guarantees $\frac{\pi}{4d(v)}$ angular resolution, 1 bend per edge, and unit vertex resolution. The bend-point resolution and edge separation are parameters that can be modified to achieve different types of drawings and drawing areas. In particular, for the same parameters as the CRA algorithm (unit bend-point resolution and $\sqrt{2}/2$ edge separation), the PRA algorithm creates a drawing of size $9n \times \frac{9n}{2}$. Note that in some situations the vertex resolution is more important than the bend-point resolution or the edge separation. In such situations, all of the previous algorithms perform poorly since they are designed to maintain constant resolution particularly between vertices and bend-points. Using the PRA algorithm, we can relax the bend-point resolution constraints and get significant improvements.

The PRA algorithm relies on a novel approach for representing bends and vertices. Traditionally, vertices and bend-points are restricted to lie on integer grid coordinates. One reason for this is that the points are defined by a pair of integers. In this way, all operations on the points (for example, shifting) are performed with integer arithmetic. At the drawing stage, the integer coordinates are mapped to pixels on the screen.

Another reason for placing vertices and bend-points on integer grid coordinates is that this approach guarantees good vertex resolution, good bend-point resolution, and good edge separation [1, 6–8]. Rather than insisting that bend-points lie on integer grid coordinates, we propose an alternative approach which allows bend-points to be located on a grid represented by polar coordinates. We call this a *polar representation* approach because both the vertices and the bend-points are represented using polar coordinates.

At the exact moment of drawing the graph onto the screen, an algorithm using polar representation requires a rounding calculation to determine the exact pixel location for the bend-points. Note, however, that the traditional approach also uses a rounding calculation for scaling from the integer grid space to the pixel space.

The main advantage of using a polar representation is that it allows us to independently control grid size and bend positions. Polar coordinates allow us to specify different vertex resolution, bend-point resolution, and edge separation. We achieve this added flexibility at the expense of slightly increased storage for the graph representation. A Cartesian representation requires exactly two integers for each point while the polar representation requires up to five integers per point.

2 The CRA Algorithm

The Cartesian Representation Algorithm is a natural extension of the previous algorithms which guarantee good angular resolution [8, 7, 6, 1]. For the remainder of this paper, when we say "graph" we mean a fully triangulated, undirected,

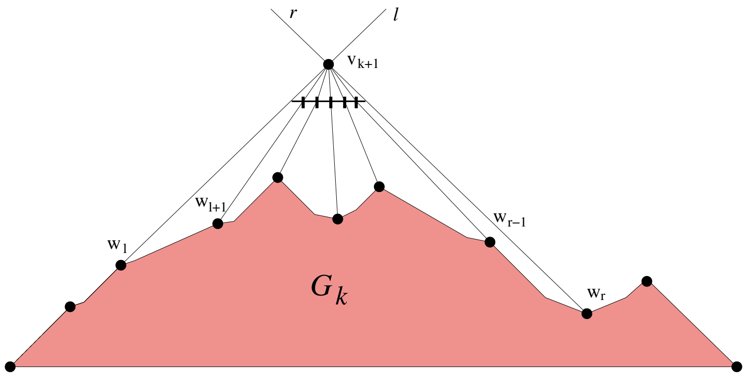

Fig. 1. Graph G_{k+1} after inserting v_{k+1}. The shaded part is G_k. Vertices w_l and w_r are the leftmost and rightmost neighbors of v_{k+1}. The horizontal line segment below v_{k+1} is the middle port region through which all the edges (v_{k+1}, w_i), $l < i < r$, are routed.

planar graph. In our algorithm the vertices of the graph are inserted sequentially by their canonical ordering, generating subgraphs $G_1, G_2, \ldots, G_n$. The canonical ordering [3] for a planar graph G orders the vertices of G so that they can be inserted one at a time without creating any crossings. We define G_k at step i to be the graph induced by vertices $1, 2, \ldots, k$. Graph G_{k+1} is created from G_k by inserting the next vertex v_{k+1} in the canonical order. Before we show the details of our algorithm we need several definitions. Following the notation of [3], let $w_1 = v_1, w_2, \ldots, w_m = v_2$ be the vertices of the exterior face C_k of graph G_k in order. For a particular subgraph G_k and vertex v_{k+1}, we refer to w_l and w_r as the leftmost and rightmost neighbors of v_{k+1} on C_k, see Fig. 1.

2.1 Vertex Regions

In the immediate vicinity of every vertex there are two types of regions: *free regions* and *port regions*. The free and port regions alternate around the vertex, see Fig. 2.1(a). For each free region there is at most one edge passing through it to v. Each port region is bounded by a line segment with a number of ports and every edge inside the port region passes through a unique port. The number of ports in a port region is as small as possible. Define and name the six regions around v as follows:

There are three free regions M^f (between $-45°$ and $45°$), R^f (between $90°$ and $135°$), and L^f (between $-135°$ and $-90°$). There are also three port regions M^p (between L^f and R^f), L^p (between L^f and M^f), and R^p (between R^f and M^f).

The algorithm draws each edge in E by "routing" it through a port of one of the two vertices in a fashion similar to [1]. Each edge consists of two connected edge segments. One edge segment, the *port edge segment*, connects a vertex with one of its ports. The other segment, the *free edge segment*, connects a vertex

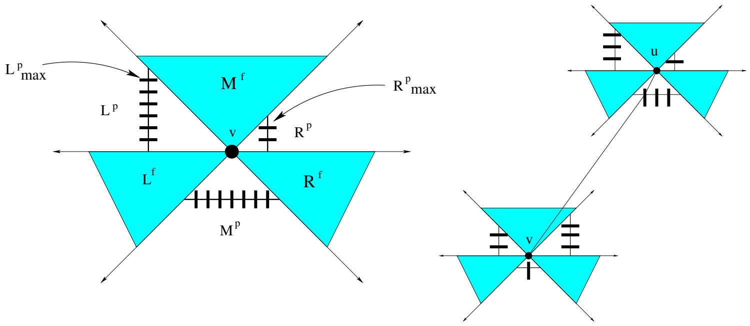

Fig. 2. Vertex regions and edge routing: the number of ports along each port region is determined by the number of edges that need to be routed through that port region. (a) The area around a vertex v is divided into 6 regions. The free regions are shaded and at most one free edge segment goes through each one of them. All the port segments use ports in the port regions of v. (b) Routing an edge $e = (u, v)$, where the port edge segment connects u to one of its ports and the free edge segment connects the port to v, going through one of v's free regions.

to one of its neighbor's ports. For example, for an edge $e = (u, v)$, if we route e through the leftmost port in u's middle free region M^p, we would draw two line segments, see Fig. 2.1(b): the *port edge segment* would pass from u to the port, and the *free edge segment* would pass from the port to v. This method of construction guarantees that the free edge segments always pass through free regions and that each port transmits at most one port edge segment.

We perform our construction in incremental stages, where each stage corresponds to the insertion of a new vertex. Observe that at each stage, for every vertex v except those on the external face, $w_1 = v_1, w_2, \ldots, w_m = v_2$, there are exactly three free edge segments. The remaining edges are connected to v via port segments. These remaining edges can be grouped into three classes based on which port region they are routed through, L^p, R^p, or M^p. Count the number of edges in each of these groups and let $d_l(v)$ be the number of port edge segments using port region L^p. Similarly, define $d_r(v)$ and $d_m(v)$ to be the number of port segments using port regions R^p and M^p. Observe that in the final stage, there are exactly three vertices on the exterior face, v_1, v_2, v_n, and then $\sum_{v \in V}(d_l(v) + d_r(v) + d_m(v) = |E|$. That is, for every edge, there is a corresponding port and free edge segment.

For a vertex v we define the *maximal right port* R^p_{max} as follows. Let v have coordinates (v_x, v_y). Then the R^p_{max} of v has coordinates $(v_x + d_r(v) + 1, v_y + d_r(v))$ if $d_r(v) > 0$. We define the *maximal left port* L^p_{max} of v in a similar fashion, see Fig. 2.1(a).

2.2 Invariants of the CRA Algorithm

By design, our algorithm is incremental with n stages, where each stage corresponds to the insertion of the next vertex in the canonical order. Thus it is natural to define several key invariants to be maintained at every stage. The four invariants below are similar in flavor to those of Cheng *et al* [1] except that here we do not need to maintain any joint boxes.

1. All vertices and ports have integer coordinates.
2. Let $w_1 = v_1, w_2, \ldots, w_m = v_2$ be the vertices of the exterior face C_k of G_k in order, and let $x(w_i)$ be the x-coordinate of vertex w_i. Then $x(w_1) < x(w_2) < \ldots < x(w_m)$.
3. The free edge segment of edge $e = (w_i, w_{i+1})$, $0 < i < m$, has slope ± 1 and e's port edge segment goes through a maximal port.
4. For every vertex v there is at most one (free) edge segment crossing each of its free regions. All other edge segments are port edge segments.

2.3 Vertex Shifting

In the algorithms that maintain good angular resolution with the aid of vertex joint boxes [1, 6], every time a new vertex is inserted, already placed vertices need to be shifted a great deal so that the joint box can fit amongst them. The amount of shifting required is typically of the order of the degree of the vertex. Invariably this leads to large constants behind the $O(n) \times O(n)$ area, e.g. $(20n - 48) \times (10n - 24)$ in [6] and $30n \times 15n$ in [1]. In our algorithm we never need to shift any vertex by more than five grid units allowing us to draw G in a $5n \times \frac{5n}{2}$ grid. When a new vertex v is inserted, we must create enough space so that the leftmost w_l and rightmost w_r neighbors of v can "see" v through their respective maximal port regions. Note that the previous R^p_{max} port of w_l and L^p_{max} of w_r were used at a previous stage. Thus, we must create an additional port along the R^p region of w_l. Similarly, additional space is necessary along the L^p region of w_r.

In order to create more space we need to move w_l and w_r. We also have to ensure that the four invariants and the planarity of the graph are maintained. This is achieved by shifting the "shifting set" of the vertex as well as the vertex itself. Using the definition of de Fraysseix *et al* [3], define the *shifting set* $M_k(w_i)$ for a vertex w_i on the external face of G_k to be a subset of the vertices of G such that:

1. $w_j \in M_k(w_i)$ iff $j \geq i$
2. $M_k(w_1) \supset M_k(w_2) \supset \ldots \supset M_k(w_m)$
3. Let $\delta_1, \delta_2, \ldots, \delta_m > 0$; if we sequentially translate all vertices in $M_k(w_i)$ by distance δ_i to the right $(i = 1, 2, \ldots, m)$, then the embedding of G_k remains planar.

These shifting sets can be defined recursively. Let w_l and w_r be the leftmost and rightmost neighbors of v on C_k. Then construct $M_{k+1}(w_i)$ recursively as follows:

$$M_{k+1}(w_i) = M_k(w_i) \cup v_{k+1}, \text{ for } i \leq l,$$

$$M_{k+1}(v_{k+1}) = M_k(w_{p+1}) \cup v_{k+1},$$

$$M_{k+1}(w_j) = M_k(w_j), \text{ for } j \geq r.$$

For convenience, define a *right-shift of m units* for a vertex w_i as shifting $M_k(w_i)$ by m units to the right so that all ports for every vertex in $M_k(w_i)$ also shift except the ports in the L^p region of w_i. Define a *left-shift of m units* for vertex w_i as shifting $M_k(w_{i+1})$ by m units to the right so that all ports for every vertex in $M_k(w_{i+1})$ also shift. Note that in a left-shift, we also shift the ports in the R^p region of w_i by m units to the right.

2.4 CRA Overview

The CRA algorithm constructs the graph one vertex at a time, by creating the graphs $G_1, G_2, \ldots, G_n$. Constructing G_i, $1 \leq i \leq 3$ is straightforward, so assume that G_k has been constructed with exterior face $C_k = (v_1 = w_1, w_2, \ldots, w_m = v_2)$. Suppose we have embedded G_k with exterior face C_k. To construct G_{k+1}, let v_{k+1} be the next vertex in the canonical ordering and recall that w_l and w_r are, respectively, the leftmost and rightmost neighbors of v_{k+1} on the exterior face C_k.

Recall that $d_r(w_l)$ is the current number of port edge segments using R^p of w_l, and that $d_l(w_r)$ is the current number of port edge segments using L^p of w_r. There are two cases to consider:

- **case (a)** $d_l(w) = 0$, see Fig. 3(a).
- **case (b)** $d_l(w) > 0$, see Fig. 3(b).

In case (a) perform a left-shift of 2 units on w_l in order to free space for a port in the R^p region of w_l. In case (b) perform a left-shift of 1 unit on w_l. Similarly, if $d_l(w_r) = 0$ then perform a right-shift of 2 units on w_r. Otherwise perform a right-shift of 1 unit on w_r.

Insert v_{k+1} at the intersection of lines l and r, where l is the line with slope $+1$ through w_l's maximal right port and r is the line with slope -1 through w_r's maximal left port, see Fig. 1. In the case where lines l and r do not intersect in a grid point it suffices to shift all the elements in $M_k(w_r)$ one additional unit to the right.

The edges from v_{k+1} to w_l and w_r are routed through w_l's maximal right port and w_r's maximal left port, respectively. The remaining edges go from v_{k+1} to vertices w_i, $l < i < r$. Let $(v_{k+1}(x), v_{k+1}(y))$ be the coordinates of vertex v_{k+1}. Let w_a be the rightmost vertex such that $w_a(x) < v_{k+1}(x)$. Before placing the M^p region of v_{k+1} it is necessary to ensure that there are enough ports on it that can be used to connect v_{k+1} to $w_l, w_{l+1}, \ldots, w_r$. The M^p region is a horizontal line segment with $1, 3, \ldots, 2m+1$ ports when the line segment is $1, 2, \ldots, m$ grid units below v_{k+1}. It is necessary to find how many w_i's are to the left and right of v_{k+1} in order to find exactly how far below vertex v_{k+1}, the region M^p must be placed. These numbers are $a - l$ and $r - a - 1$ respectively (from the definition of w_a above). Then place the M^p region of v_{k+1} so that its y coordinate is equal to $v_{k+1}(y) - \max(a - l, r - a - 1)$. As shown in the next section the M^p region can be placed correctly, that is, placed so that it does not

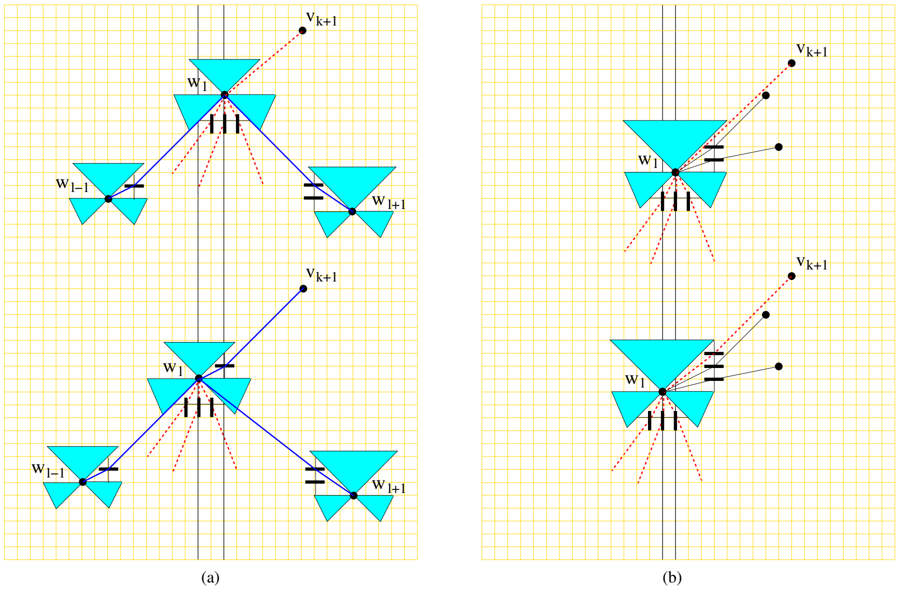

Fig. 3. Adding the current vertex v_{k+1}. Here w_l is the leftmost neighbor of v_{k+1} on the exterior face of G_k. (a) If $d_r(w_l) = 0$, then we need to shift w_l two grid units to the left. (b) If $d_r(w_l) > 0$, then it suffices to shift w_l only one unit to the left. Note that the shifting set $M_k(w_l)$ also shifts with w_l.

intersect the old graph G_k. After determining the location of M^p the edges are routed from v_{k+1} to $w_{l+1}, w_{l+2}, \ldots, w_a$ starting from the leftmost port of M^p of v_{k+1}. Similarly, route the edges from v_{k+1} to $w_{r-1}, w_{r-2}, \ldots, w_{a+1}$ starting from the rightmost port of M^p of v_{k+1}.

3 Correctness of the Algorithm

The algorithm works correctly if all four invariants are maintained. We show that free edge segments always remain in free edge regions and that there is at most one free edge segment per free region. We then need to bound the drawing area required by the algorithm and show that good angular resolution is maintained. Finally, we have to bound the number of bends created and analyze the running time. The detailed proofs are left out of this extended abstract and brief proof sketches are provided instead.

Lemma 1. *Free edge segments in free regions remain in the free regions.*

Proof Sketch: Recall that there are three types of free regions: M^f, L^f, R^f. Let us first look at free edge segments in the M^f regions. Edge segments in the M^f regions are created by a vertex v dominating another vertex w. But whenever v dominates w, w is added to the shifting set for v and is only shifted when v is shifted. Therefore, the slope of the free edge segment of the edge connecting v

and w remains constant and the free edge segment remains within M^f. As the two remaining cases are symmetric, without loss of generality, let us examine the case when the free edge segment lies in the L^f region. This implies that the slope of the free edge segment is between 0 and +1. Since shifting only moves vertices farther apart in the x-direction, the slope can only get closer to 0, thus remaining in L^f. □

Lemma 2. *Every free edge segments passes through a free region which contains no other edges.*

Proof Sketch: When a new vertex $v = v_{k+1}$ is inserted there are two types of new edges added: the *outside edges* between v and the outside neighbors, w_l and w_r, and the *inside edges* between v and w_i where $l < i < r$. In both cases the new edge is routed through a port creating one free edge segment and one port edge segment. A free edge segment of an outside edge has slope either +1 or −1 by construction; therefore it lies inside the free regions L^f and R^f of vertex v. Since v is a new vertex, there are no other segments inside these two free regions.

Dealing with the inside edges is more complex. We first need to show that there is sufficient space between the vertices on the exterior face of G_k and the new vertex v. Second, we need to show that v has enough ports in its middle port region M^p for each of the vertices on G_k that it is connected to. Third, we need to show that the free edge segments of the inside edges remain inside their free regions. We begin by showing that for every vertex w_i, $l < i < r$,

- vertex w_i lies below M^p, the middle port region for v, and
- we can assign a unique port along the M^p port region of v, such that the edge segment connecting w_i to that port fits inside w_i's middle free region M^f

First consider the vertices $w_l, w_{l+1}, \ldots, w_a$. In the worst case, they have monotonically increasing y-coordinates. Intuitively, this is the worst case because the area of $G_{k+1} - G_k$ is the smallest and hence it is more difficult to ensure that there is enough space for the M^p region of v. Using invariant (3) we can show that $w_a(y) \leq v(y) - (a - l)$, since all the edges connecting consecutive vertices on the outer face have both port and free edge segments, thus "freeing" at least one grid point distance between v and w_a for every w_i. Similarly, $w_{a+1}(y) \leq v(y) - (r - (a + 1))$.

The same argument implies that there are at least as many ports along the middle port region M^p of v as there are vertices w_i, $l < i < r$. We route the inside edges for v as follows: starting with the leftmost port on the middle port region M^p of v, we assign consecutive ports to $w_{l+1}, w_{l+2}, \ldots, w_a$ and route the corresponding free edge segments from w_i to the ports. The port edge segments for the inside edges connect the same ports to v. Similarly, starting with the rightmost port on the middle port region M^p of v we assign consecutive ports to $w_{r-1}, w_{r-2}, \ldots, w_{a+1}$ and route the corresponding free edge segments from w_i to the ports. The port edge segments for the the inside edges connect the same ports to v.

Combining the above observations, it can also be shown that for every vertex w_i, $l < i < r$, the free edge segment of its edge to v fits inside the middle free region M^f of w_i. □

Lemma 3. *If G_k maintains invariants one through four, then G_{k+1} maintains invariants one through four.*

Proof Sketch: By definition of the shifting set, invariants one and two hold, see [6]). By construction of the algorithm, invariant three holds as well. Also by construction every edge inserted has a port edge segment and a free edge segment. By lemmas 1 and 2 invariant four also holds. □

Lemma 4. *The angular resolution for vertex $v \in G$ as produced by the algorithm is $1/2d(v)$, where $d(v)$ is the degree of vertex v.*

Proof Sketch: The worst angle is achieved between a free edge segment for some edge f and a port edge segment for some edge e, where f is located at the boundary of its free region and e is the neighboring port edge segment. There are six possible cases but the argument is the same for all of them, so without loss of generality consider the case in Fig. 4. Let v be the vertex and $d(v) = d$ its degree. Also let s and t be the lengths as shown in Fig. 4. Let θ be the angle between f and e, and x the number of ports as shown in the figure. Note that all vertices have three edges connected to them via free edge segments. Then the number of ports in any port region is at most $d - 3$. From the figure, $\tan(\theta) = t/(s - t)$ and hence $\arctan(t/(s - t)) = \theta$. But

$$\frac{t}{s-t} = \frac{\sqrt{2}/2}{\sqrt{2}(x+1) - \sqrt{2}/2} = \frac{1}{2x+1}$$

Using the Maclaurin expansion for $\arctan(y)$, where $y < 1$ we have

$$\arctan(y) = y - y^3/3 + y^5/5 - ...$$

Since $1/(2x + 1) < 1$, and $x \leq d - 3$ this yields $\theta \geq 1/2d$ which completes the proof. □

Theorem 1. *For a given planar graph G, the algorithm produces in $O(n)$ time a planar embedding with grid size $5n \times 5n/2$, using at most one bend. The angular resolution for every vertex v of G is $1/2d(v)$.*

Proof Sketch: Since every edge has only two segments, there can be at most one bend per edge. Chrobak and Payne [2] show how to implement the algorithm of De Fraysseix, *et al.* [3] in linear time. Their approach can be easily extended to our algorithm. By invariants three and four and by lemma 4 the angular resolution is at most $1/2d(v)$.

It remains to show that the drawings produced by the algorithm fit on the $5n \times 5n/2$ grid. Every time we insert a vertex v_k, we increase the grid size by at most 5 units, which implies that the width of the drawing is at most $5n$. The final drawing fits inside an isosceles triangle with sides of slope $0, +1, -1$. The width of the base is $5n$ and so the height is less than $5n/2$. □

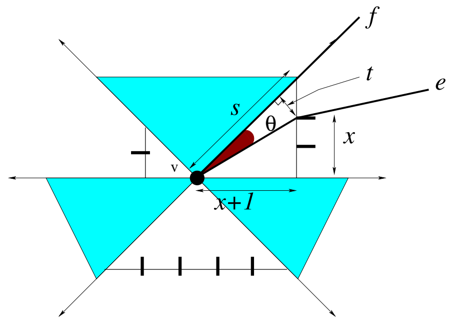

Fig. 4. The minimum angle between two edges adjacent to vertex v is proportional to the degree d of the vertex. Using our algorithm the angle cannot be smaller than $2/d$.

4 The PRA Algorithm

In this section, we introduce a novel approach to representing bends and vertices. Rather than insisting that bends lie on integer grid coordinates, we propose an alternative approach which allows bends to be located on a grid represented by polar coordinates. Using a polar representation allows us to independently control the grid size and edge bend positions. We begin by considering the polar representation in general and then present the PRA algorithm that uses the new approach.

A point p in the polar grid system is represented by at most five integers. For the vertices we only need two integers (p_x, p_y). For the bend-points we may need up to five integers. We shall see in the PRA algorithm that these five integers need not be explicitly stored for every bend-point. In general, a bend-point is given by:

– (p_x, p_y), the origin of the polar system
– p_r, the radius of the circle around the origin (p_x, p_y)
– p_d and p_n, the angle (p_θ) of the circle where the point is located, i.e., $p_\theta = 2\pi p_n/p_d$. For convenience, we consider $p_\theta = 0$ to be the vertical direction.

The PRA algorithm places vertices at integer grid coordinates, thus guaranteeing unit vertex resolution. As it is based on the CRA algorithm it also uses only 1 bend per edge. The main difference in the two algorithms is in the placement of the bend-points. In the PRA algorithm, bend-points will be placed on a circle around the vertex (rather than on a straight-line segment). Therefore, the origin, (p_x, p_y) for each bend-point need not be explicitly stored – it suffices to store the origin of the vertex that the bend-point is associated with. Similarly, groups of bend-points around a given vertex will have the same radius and hence each of the bend-points need not explicitly store p_r.

Consider the leftmost neighbor, w_l, of the next vertex in the canonical order, v_{k+1}. The ports are evenly spaced in the P_r region for w_l, Fig. 5(a). The length of the straight-line segment separating two bend-points or a bend-point and

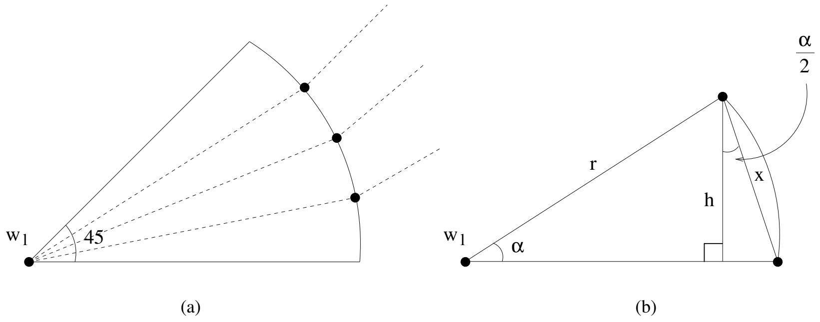

Fig. 5. Vertex w_l is the left-most neighbor of the next vertex v_{k+1} along the exterior face of G_k. The $d_r(w_l)$ ports of w_l are evenly spaced on the arc of a circle of radius $2d_r(w_l)$ bounded by the middle free region M^f and the right free region R^f. (a) An example of the layout for the R^p region with $d_r(w_l) = 3$. (b) The distance x between two adjacent ports or a port and an adjacent free region can be computed given the radius of the circle and the angle between the edges connecting the ports to w_l: $x = 2r \sin \frac{\alpha}{2}$.

an adjacent free region can be computed as follows. Consider the example in Fig. 5(b). We would like to compute the length x in terms of the radius of the circle and the angle between the two line segments connecting consecutive ports to w_l. From basic trigonometry, the angle between h and x is $\alpha/2$. We can express h in terms of r and α: $h/r = \sin \alpha$ and we can express x in terms of h and α: $h/x = \cos \alpha/2$. Combining the two expressions we obtain

$$x = \frac{h}{\cos \frac{\alpha}{2}} = \frac{r \sin \alpha}{\cos \frac{\alpha}{2}} = \frac{2r \sin \frac{\alpha}{2} \cos \frac{\alpha}{2}}{\cos \frac{\alpha}{2}} = 2r \sin \frac{\alpha}{2}.$$

Assume we have inserted $v_1, v_2, \ldots, v_k$ and have a drawing of G_k with exterior face C_k. Consider inserting the next vertex v_{k+1} in the canonical order. Let w_l and w_r be the leftmost and rightmost neighbors of v_{k+1} on the exterior face C_k. Define f_b and f_e to be the *bend-point resolution* and *edge separation* respectively. Recall that in the standard Cartesian representation algorithms $f_b = 1$ and $f_e = \sqrt{2}/2$. Let $d_r(w_l)$ be the number of port edge segments using R^p of w_l. There are two cases to consider:

- **case (a)** $d_r(w_l) = 0$
- **case (b)** $d_r(w_l) \geq 1$

In **case (a)** we insert the first edge in the port region R^p between the two free regions R^f and M^f of w_l. We place the port in the middle of the arc of a circle connecting R^f and M^f. Since there are no other bends yet in R^p we are only concerned with maintaining the edge separation. We need to place the port sufficiently away from the vertex w_l. Consider the relationship between the radius of the circle and the edge separation, see Fig. 5.

The edge separation $f_e = x = 2r \sin \frac{\alpha}{2}$. But since there is only one port and it is in the middle of the arc, $\alpha = \pi/8$. We are interested in the radius necessary to achieve the edge separation f_e which is given by

$$r = \frac{f_e}{2\sin\frac{\alpha}{2}} = \frac{f_e}{2\sin\frac{\pi}{16}} < \frac{4f_e}{\sqrt{2}} = 2\sqrt{2}f_e.$$

Since we maintain that the vertices are at integer coordinates and the radii are also integers, then the minimum radius required in case (a) is

$$r < \lceil 2\sqrt{2}f_e \rceil.$$

In **case (b)** we insert an additional port in the port region R^p which already has at least one port. In this case, we must ensure that both the edge separation f_e and bend-point resolution f_b are preserved. In this case the radius required is given by:

$$\max\{\lceil \frac{f_e}{2\sin\frac{\pi}{8(d_l(w_r)+1)}} \rceil, \lceil \frac{f_b}{2\sin\frac{\pi}{8(d_l(w_r)+1)}} \rceil\}.$$

Typically, $f_b \geq f_e$, so we assume that the bend-point resolution determines the radius in case (b). Using this together with the fact that $sin\alpha > 0.97\alpha$ for $\alpha < \pi/8$, the minimum radius required is

$$r < \lceil \frac{f_b}{2\sin\frac{\pi}{8(d_l(w_r)+1)}} \rceil < \lceil \sqrt{2}f_b(d_l(w_r)+1) \rceil$$

Summing over all vertices in the graph, the sum of the radii of the left neighbors, L, yields:

$$L = \sum_{v_i \in V : d_l(v_i)=0} \lceil 2\sqrt{2}f_e \rceil + \sum_{v_i \in V : d_l(v_i) \geq 1} \lceil \sqrt{2}f_b(d_l(v_i)+1) \rceil. \tag{1}$$

With L we bounded the number of shifts required because of "left" neighbors. Similarly, we can define R, the shifts necessary due to "right" neighbors:

$$R = \sum_{v_i \in V : d_r(v_i)=0} \lceil 2\sqrt{2}f_e \rceil + \sum_{v_i \in V : d_r(v_i) \geq 1} \lceil \sqrt{2}f_b(d_r(v_i)+1) \rceil. \tag{2}$$

Algorithm	f_v	f_b	f_e	drawing area	resolution
CRA	1	1	$\sqrt{2}/2$	$5n \times 5n/2$	$1/2d(v)$
PRA1	1	1	$\sqrt{2}/2$	$9n \times 9n/2$	$\pi/4d(v)$
PRA2	1	1/2	1/2	$7n \times 7n/2$	$\pi/4d(v)$

Table 1. Fixing specific values for the vertex resolution f_v, bend-point resolution f_b, and edge separation f_e allows us to compare the PRA and CRA algorithms.

L and R bound the number of shifts required due to left and right neighbor visibility. Note, however, that if we shift by the minimum amount required by the f_e and f_b parameters, the location of the next vertex v_{k+1} may not be at integer coordinates. We can guarantee that v_{k+1} is placed on the integer grid by performing some additional shifts. It is easy to show that by shifting at most 3 more units, we are guaranteed to find an integer location for v_{k+1}. Then the total shifting required is at most $L + R + 3n$. Since the final drawing fits inside an isoceles right-angle triangle, the total area required for the drawing is $(L + R + 3n) \times (\frac{L+R+3n}{2})$.

In order to compare the PRA algorithm to the CRA algorithm, we evaluate equations 1 and 2 using two sets of parameters, Table 1. In all three cases the algorithms guarantee at most one bend per edge. The PRA algorithms place all the vertices on grid points and each bend-point is determined by at most 5 integer polar coordinates.

5 Conclusion and Open Problems

In this paper we present two algorithms for drawing planar graphs with good angular resolution while maintaining small drawing area. Other drawing criteria optimized by the algorithms include number of bends, vertex resolution, bend-point resolution, and edge separation. The first algorithm, CRA, is a traditional algorithm in which vertices and bend-points are represented using Cartesian coordinates. It improves on the best known simultaneous bounds for the six drawing criteria. In the PRA algorithm vertices and bend-points are represented using polar coordinates. It is based on the CRA algorithm but allows for independent control over the grid size and bend positions.

Using a polar coordinate representation yields slightly worse area bounds compared to the CRA algorithm. We believe, however, that the PRA approach is more promising. The angular resolution of the PRA algorithm is better and it provides greater control over the drawing process. The PRA bounds presented in this paper can be further improved. Using two integers to represent the radius (similar to the way the angles are currently represented) will most likely result in smaller drawing area. It is likely that using only one bend per edge, the best angular resolution will be achieved for vertex regions in which each of the port and free regions have angles $\pi/3$ rather than a combination of $\pi/4$ and $\pi/2$.

References

1. C. C. Cheng, C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Drawing planar graphs with circular arcs. *Discrete and Computational Geometry*, 25:405–418, 2001.
2. M. Chrobak and T. Payne. A linear-time algorithm for drawing planar graphs. *Inform. Process. Lett.*, 54:241–246, 1995.
3. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.
4. Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, Englewood Cliffs, NJ, 1999.
5. A. Garg and R. Tamassia. Planar drawings and angular resolution: Algorithms and bounds. In *Proc. 2nd Annu. European Sympos. Algorithms*, volume 855 of *Lecture Notes Computer Science*, pages 12–23. Springer-Verlag, 1994.

6. M. T. Goodrich and C. G. Wagner. A framework for drawing planar graphs with curves and polylines. In *Proceedings of the 6th Symposium on Graph Drawing (GD '98)*, pages 153–166, 1998.
7. C. Gutwenger and P. Mutzel. Planar polyline drawings with good angular resolution. In *Proceedings of the 6th Symposium on Graph Drawing (GD '98)*, pages 167–182, 1998.
8. G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996. (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia).
9. S. Malitz and A. Papakostas. On the angular resolution of planar graphs. *SIAM J. Discrete Math.*, 7:172–183, 1994.