Improved Circular Layouts

Emden R. Gansner and Yehuda Koren
AT&T Labs — Research
Florham Park, NJ 07932, USA
{erg,yehuda}@research.att.com

Abstract. Circular graph layout is a drawing scheme where all nodes are placed on the perimeter of a circle. An inherent issue with circular layouts is that the rigid restriction on node placement often gives rise to long edges and an overall dense drawing. We suggest here three independent, complementary techniques for lowering the density and improving the readability of circular layouts. First, a new algorithm is given for placing the nodes on the circle such that edge lengths are reduced. Second, we enhance the circular drawing style by allowing some of the edges to be routed around the exterior of the circle. This is accomplished with an algorithm for optimally selecting such a set of externally routed edges. The third technique reduces density by coupling groups of edges as bundled splines that share part of their route. Together, these techniques are able to reduce clutter, density and crossings compared with existing methods.

1 Introduction

Circular layouts are among the most prominent and oldest conventions used to draw graphs. In such layouts, nodes are drawn on a circle, while the edges connecting these nodes are line segments passing within the circle, e.g., Figure 1(a). This drawing convention is often used for the layout of networks and systems management diagrams, where it naturally captures the essence of ring and star topologies. It can be also used for other kinds of graphs, such as social networks and WWW graphs. In particular, a circular layout is appropriate for applications that emphasize the clustering decomposition of a graph, where each cluster is drawn on a separate circle. Much work [1, 4, 12, 13, 17, 18, 20] has been done on these layouts, most of it addressing both the layout of a single circle as well as positioning multiple circles together in order to show the various clusters composing the full graph. Here we concentrate on the former. Circular layouts are highly regularized – nodes placed on a circle – achieving a very clear depiction of each individual node. A node cannot be occluded by another node or by an edge. Moreover, since it is impossible to have three collinear nodes, the problem of two edges obscuring each other is avoided. In general, these layouts can provide a compact presentation, focusing on individual nodes and edges. Additionally, well-designed circular layouts sometimes reveal global properties of the graph such as symmetries and patterns of collective behavior. On the other hand, this strong regularity can obscure other information. For example, these drawings can be very dense, and following paths on them can be difficult.

In this work we suggest methods for improving the clarity of circular layouts through better node placement and edge routing. This is achieved using three contributions. The first shows how to adapt traditional energy based node placement considerations in order to shorten edges in circular layouts. This is different from most previous work which concentrated on reducing edge crossings. Experiments show that our method is competitive in terms of edge crossing minimization, while being constantly better in terms

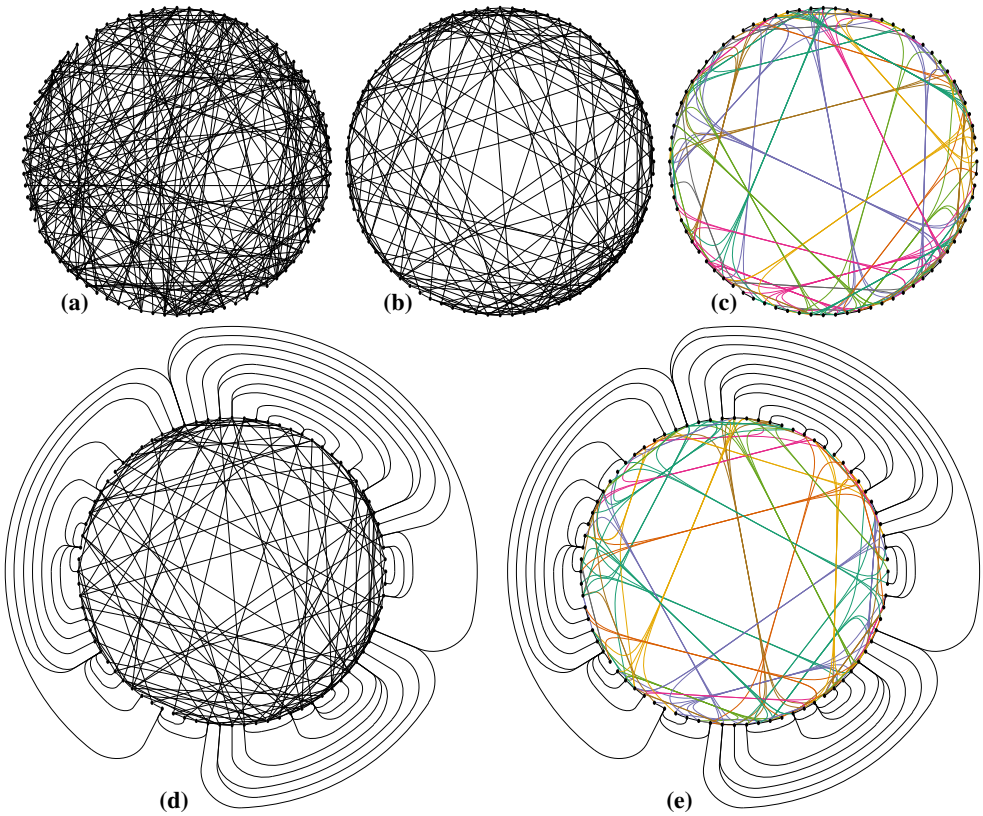

Fig. 1. Variations on circular layouts of a random graph ($|V| = 80$,$|E| = 241$): **(a)** random order; **(b)** edge-length minimizing order; **(c)** bundling edges to save ink and to improve area utilization (colors used to enhance readability); **(d)** exterior routing lessens crossings and alleviates density; **(e)** combining exterior routing with edge bundling.

of overall edge length. Such a shortening of edge lengths allows the use of less "ink" for drawing the graph, thereby improving clarity. This ink saving paradigm brings us to the second contribution of the paper. We suggest a novel edge routing technique, which uses less ink compared with the common convention of drawing edges as straight lines. This is performed by carefully bundling together line segments between a few edges in a way that frees up drawing area without compromising structural clarity. Considering non-straight line edges opens up even more possibilities for better clarity. Accordingly, our third contribution suggests routing some of the edges through the external face. The externally routed edges are optimally selected in order to minimize certain criteria. In particular, external routing can be very effective in reducing edge crossings.

When used together, one normally performs node placement, followed by exterior routing, then edge bundling. Sections 2-4 consider these techniques in that order. Experimental studies for the techniques are given in Section 5.

2 Node placement

We are looking for the "best placement" of the n nodes of a graph $G(V = \{1, \ldots, n\}, E)$. By convention, we assume nodes are equally spaced on the circle, which reduces the problem to finding a *circular ordering* of the nodes. This requirement imposes a certain regularity on the resulting layouts, while having no effect on the number of edge crossings, since only the ordering affects edge crossings.

Some related computational problems are known to be NP-hard. One example is the Minimum Circular Arrangement problem, where nodes are arranged on a circle (with equal gaps) in order to minimize the total angular edge lengths. This problem is reducible from the extensively studied NP-complete problem of Minimum Linear Arrangement [9]. Additional related circular arrangement problems and applications are mentioned in [6, 14]. Another NP-hard problem is Circular Crossing Minimization [15], where the goal is to minimize the number of edge crossings in the layout. Given the NP-hardness of the relevant problems, our approach is based on heuristics that cannot guarantee finding an optimal solution.

2.1 Mean- and median- iterations

While previous work [1, 4, 12, 17] explicitly addressed edge crossings, we prefer to deal with the simpler node-node interactions governing edge lengths. That way we can use ideas developed in other areas of graph drawing, which seek to minimize edge length. The rationale here is that long edges are hard to follow, prone to crossings, and cause unnecessary clutter and density. One such class of methods consists of force-directed algorithms, which define the layout by minimizing a cost function. The methods of Tutte [21] and Hall [10] are probably closest to the one used here. In addition, our technique is closely related to the mean-iteration and the median-iteration heuristics widely used within the crossing minimization phase of Sugiyama-based digraph drawing algorithms [19].

We denote the coordinates of a node $i \in V$ by $(x_i, y_i) \in \mathbb{R}^2$. Assume that the nodes are arranged on the unit circle centered at the origin. We would like to minimize the total squared edge lengths, resulting in the following optimization problem:

$$\begin{aligned} &\min_{x,y} \sum_{\langle i,j \rangle \in E} (x_i - x_j)^2 + (y_i - y_j)^2 \\ &\text{subject to}: \;\; x_i^2 + y_i^2 = 1, \; i = 1, \ldots, n \end{aligned} \tag{1}$$

Tutte [21] and Hall [10] dealt with strategies to minimize the same function, but here we also need to account for the unit circle constraints. Such equality constraints are usually addressed by *Lagrange multipliers*. Therefore, for each node i, we introduce a Lagrange multiplier λ_i, and define the function:

$$f(x, y, \lambda) = \sum_{\langle i,j \rangle \in E} (x_i - x_j)^2 + (y_i - y_j)^2 + \sum_{i=1}^{n} \lambda_i (x_i^2 + y_i^2 - 1) \tag{2}$$

Any minimum of (1) must be a zero for all partial derivatives of (2). In other words, we require $\partial f/\partial x = 0$, $\partial f/\partial y = 0$, $\partial f/\partial \lambda = 0$. Notice that $\partial f/\partial \lambda = 0$ means that all constraints are satisfied. The other equalities, $\partial f/\partial x = \partial f/\partial y = 0$, imply that for each node i:

$$(x_i, y_i) = \frac{1}{1-\lambda_i} \frac{\sum_{j \in N(i)} (x_j, y_j)}{\|N(i)\|}$$

where $N(i) = \{j \mid \langle i, j \rangle \in E\}$ is the set of neighbors of i. Notice that the $\frac{1}{1-\lambda_i}$ multiplier provides the degree of freedom necessary for satisfying the unit circle constraint. In plain words, these equations state that each node on the unit circle should lie on the line connecting the origin and the barycenter of its neighbors. Equivalently, the angular coordinate of each node is the mean of the angular coordinates of its neighbors, while the radial coordinate is always 1.

To solve this problem, we fix the positions of all the nodes but one, giving rise to an iterative optimization process, which we naturally name the *mean iteration*. At each iteration, we sequentially move each single node to the barycenter of its neighbors, and then project it back to the circle:

$$(\mathbf{1})\ \ (x_i, y_i) \leftarrow \frac{\sum_{j \in N(i)} (x_j, y_j)}{\|N(i)\|} \qquad (\mathbf{2})\ \ (x_i, y_i) \leftarrow \frac{(x_i, y_i)}{\|(x_i, y_i)\|}$$

A known problem with the mean iteration is that the global minimum of (1) is attained when all nodes are positioned at the same location. Since we are looking for more useful local minima, we avoid such a collapse of the layout by interfering with the process after each few tens of iterations and making the gaps between consecutive nodes uniform. That is, we preserve the current angular order of the nodes, but impose a uniform distribution along the circle. Additionally (or, alternatively), we adopt the anchoring mechanism suggested by Tutte, fixing the positions of three nodes, which prevents the collapse of the layout. During the process we change the anchors to avoid bias toward specific nodes.

While the mean iteration addresses squared edge lengths, a similar *median iteration* addresses non-squared edge lengths. The only difference is the use of the median instead of the mean. Therefore, in this algorithm, the coordinates of a node are iteratively determined by the component-wise median of its neighbors' coordinates, projected back onto the circle. We experienced slightly better results using median iteration over mean iteration in terms of crossing minimization.

The complexity of a single iteration is $O(n+|E|)$. The number of required iterations is less clear. We regularly use $O(n)$ iterations.

2.2 Local refinement through dynamic programming

The median (or mean) iteration is a continuous approximation to the circular ordering problem. We derive the circular order by sorting the nodes according to their angular coordinates. The resulting circular order can be refined by utilizing an algorithm that explicitly considers the discrete nature of the problem. At this stage, we hope that the median iteration already gave us an adequate global positioning of the nodes. Therefore, we opt for using a localized refinement procedure. This refinement procedure considers every sequence of k nodes, and reorders the sequence in a way that minimizes the total edge length.

More formally, assume that the circle contains n equally spaced points named $0, 1, \ldots, n-1$, where point i is located at $(\cos \frac{2\pi i}{n}, \sin \frac{2\pi i}{n})$. In addition, each of the n nodes is uniquely associated with one of the n circle points via the bijection $\mathrm{p}(i) : V \rightarrow \{0, 1, \ldots, n-1\}$. The (angular) distance between two nodes i and j is defined as:

$$d_{ij} = \min\left(\mathrm{p}(i) - \mathrm{p}(j) \mod n,\ \mathrm{p}(j) - \mathrm{p}(i) \mod n\right)$$

Given k nodes $\mathcal{V} = \{v_1, v_2, \ldots, v_k\}$, located consecutively at $\mathrm{p}(v_1), \mathrm{p}(v_1)+1, \ldots, \mathrm{p}(v_1)+k-1$, we would like to reorder $\mathcal{V}$ to minimize $l(\mathcal{V})$, the total length of the edges adjacent to $\mathcal{V}$, which is defined as:

$$l(\mathcal{V}) = \sum_{\langle i,j\rangle \in E, i \in \mathcal{V}} d_{ij}$$

Minimization of $l(\mathcal{V})$ is done by a dynamic programming algorithm which rearranges increasingly larger subsets of $\mathcal{V}$. The pseudocode is given in Figure 2. The complexity of the algorithm is $O(2^k + |E(\mathcal{V})|)$, where $E(\mathcal{V})$ is the set of edges connected to $\mathcal{V}$. Typical values of k are between 5 and 10 (our default is 6). We iteratively run it on each of the n (overlapping) subsequences of length k, so the running time of a full sweep optimizing each subsequence is $O(n(2^k + |E|))$. We run a few sweeps until the total edge length cannot be further reduced. Typically, a very low number of sweeps (10 or less) is required for convergence.

Figure 1(b) illustrates the application of these techniques to the initial layout of Figure 1(a).

3 Exterior routing

Node ordering, using the method described in Section 2 (or one of the methods described in the literature [1, 4, 12, 17]), improves the readability of the layout by removing edge crossings and shortening edges. At this stage, further readability improvement can be achieved without altering the node positions. This is accomplished by taking a subset of the edges from the interior of the circle, and routing them around the exterior of the circle, as depicted in Figure 1(d). Importantly, this can be done in an optimal way which maximizes the number of extracted edges or minimizes the number of crossings.

Since exterior routing of an edge is inherently longer than interior routing, we should utilize the exterior routing carefully, and make sure that edges routed externally are readable. Therefore, we do not allow any edge crossing within the external face. Notice that two edges cross in the external face if and only if they cross internally.

We associate weights with the edges (as explained below), and strive to maximize the total weight of the extracted edges. This is carried out using a dynamic programming algorithm. Before describing the algorithm, we make an observation about "edge flipping". Each exterior edge $\langle i, j\rangle$ can be drawn in two ways: either along the short arc connecting i and j, or along the complementary long arc connecting i and j. Therefore, we assume that all exterior edges are flipped so that no edge is passing over the length-1 arc connecting point $n-1$ with point 0 on the circle. Note that this flipping will not introduce any crossing into a crossing-free layout. As a consequence, we can cut the circle between point $n-1$ and 0, where no edge passes, and solve an equivalent problem on a line starting at 0 and ending at $n-1$. By solving the problem on a line, we determine which edges should be extracted. Then, each of these edges will be drawn on the exterior of the circle along the shorter of the two possible arcs.

The intuition behind the algorithm for solving the problem on the line is based on likening each edge to parentheses, where the left endpoint of the edge opens the parenthesis, and the right endpoint closes it. Accordingly, a non-crossing set of edges is equivalent to a valid sequence of nested parentheses. This induces the following recurrence relation, where p_{ij} is the maximal weighted sum of edges that can be legally routed between i and j:

Function MinCA_DP ($G(V,E)$, p, $\mathcal{V} = \{v_1, v_2, \ldots, v_k\} \subset V$, $ordering$)
% Given a graph (G), circular node positioning (p), and a subset of consecutive nodes ($\mathcal{V}$)
% ordered from v_1 (leftmost) to v_k (rightmost)
% compute an ordering of $\mathcal{V}$ ($ordering$) that minimizes total edge length
% Data structure: A table T whose entries are indexed by subsets of $\mathcal{V}$
% The function $\mathrm{Cut}(i, S)$ returns the number of edges between i and $S \subset \mathcal{V}$.

for each $i \in \mathcal{V}$ compute
$\mathrm{left}(i) = \{\langle i,j\rangle \in E \mid d(j,v_1) < d(j,v_k), j \notin \mathcal{V}\}$
$\mathrm{right}(i) = \{\langle i,j\rangle \in E \mid d(j,v_k) < d(j,v_1), j \notin \mathcal{V}\}$
end for

% Initialize table:
for every $S \subseteq \mathcal{V}$ **do**
$table[S].cost \leftarrow \infty$
end for
$table[\emptyset].cost \leftarrow 0$
$table[\emptyset].cut \leftarrow= \sum_{i \in \mathcal{V}} |\mathrm{left}(i)|$

% Fill table:
for $i = 1$ to k **do**
for every $S \subset \mathcal{V}$, $|S| = i - 1$ **do**
$cut^S \leftarrow table[S].cut$
$new\_cost \leftarrow table[S].cost + cut^S$ % total edge length is a sum of cuts
for every $j \in \mathcal{V} - S$ **do**
if $table[S \cup \{j\}].cost > new\_cost$ **then**
$table[S \cup \{j\}].cost \leftarrow new\_cost$
$table[S \cup \{j\}].right\_vtx \leftarrow j$
$table[S \cup \{j\}].cut \leftarrow cut^S - |\mathrm{left}(j)| + |\mathrm{right}(j)| - \mathrm{Cut}(j,S) + \mathrm{Cut}(j, \mathcal{V} - S)$
end if
end for
end for
end for

% Retrieve optimal ordering:
$S \leftarrow \mathcal{V}$
for $i = k$ to 1 **do**
$v \leftarrow table[S].right\_vtx$
$ordering[i] \leftarrow v$
$S \leftarrow S - \{v\}$
end for
end

Fig. 2. A dynamic programming algorithm for reordering a sequence of nodes in order to minimize total edge length

$$\begin{array}{ll} p_{i,i+1} = w_{i,i+1} & i = 0, \ldots, n-2 \\ p_{i,j} \quad = w_{i,j} + \max_{i<k<j}\{p_{ik} + p_{kj}\} & i = 0, \ldots, n-3,\ i+1 < j < n \end{array} \tag{3}$$

Here, w_{ij} is the weight of $\langle \mathrm{p}^{-1}(i), \mathrm{p}^{-1}(j)\rangle \in E$. Also, $w_{ij} = 0$ if $\langle \mathrm{p}^{-1}(i), \mathrm{p}^{-1}(j)\rangle \notin E$. The target value, $p_{0,n-1}$, is computed in time $O(n^2)$ by dynamic programming. This value indicates the maximal weighted sum of edges that can be extracted. The edges themselves are easily recovered using an auxiliary data structure which enables tracking the computation of $p_{0,n-1}$.

The choice of edge weights (w_{ij}) allows flexibility in the optimization goal. Our default is to pick the weights in a way that ensures minimizing the number of edge crossings. To this end, we set w_{ij} to the number of crossings involving $\langle \mathrm{p}^{-1}(i), \mathrm{p}^{-1}(j)\rangle$. In this way, the maximized value $p_{0,n-1}$ is exactly the number of saved edge crossings. Note that there is no problem of double counting, since two extracted edges cannot cross each other.

Our experience shows that exterior routing is a very effective technique, which can remove a significant portion of the edge crossings. The effect is shown in Figure 1(d) and studied in Section 5.

An additional pleasing outcome of exterior routing is that it tends to extract many of the short edges, such as edges of length 2. These edges are often hard to read when drawn as straight lines, as they are almost collinear with the adjacent length-1 edges. Furthermore, collinearity issue of specific edges can be explicitly addressed by increasing their weights, thus encouraging the algorithm to pick them for exterior routing.

4 Edge bundling

After node places are computed and possibly some edges are extracted to be drawn outside the circle, we can further improve the clarity of the drawing by using *edge bundling*. The essence of this technique is a controlled deformation of the edges, such that groups of edges share long common segments, thereby improving the utilization of the drawing area by saving ink. Put differently, while the most economical way to draw a single edge is by using a straight line, when displaying of group of edges, there might be more efficient ways. For illustration, consider Figure 1(c,e).

The idea of bundling edges is related to the work on confluent drawing [3], where edge crossings are eliminated by grouping edges in tracks. Newbery [16] applied bundling to Sugiyama-style layouts to reduce clutter. Additionally, we were inspired by a recent work by Holten and van Wijk [11] that suggested bundling edges based on hierarchical structure associated with the nodes. Our approach is based on a different technique for bundling edges. In the following, we split the description of the technique into two parts. First, we describe how to bundle together a given set of edges in a way that maximizes area utilization and readability. Second, we describe the algorithm for computing the sets of edges that will be bundled.

Consider the case where we are given a set of m lines ("edges"), $Q = \{e_1 = (v_1, u_1), e_2 = (v_2, u_2), \ldots, e_k = (v_m, u_m)\}$, where $v_i, u_i \in \mathbb{R}^2$. In the accompanying example, given in Figure 3, this set includes the 4 edges (A, E), (B, F), (C, G) and (D, H). Our first step is to divide the $2m$ endpoints of the edges into two equally sized sets – S ("sources") and T ("targets") – such that for each $(v_i, u_i) \in Q$, either $v_i \in S$, $u_i \in T$, or $u_i \in S$, $v_i \in T$. The intention here is to produce two compact sets, minimizing Euclidean distances between nodes belonging to the same set. We achieve

this by a variant of the K-means algorithm, where we iteratively assign each point to the set with the closer mean while continually updating the means. Accordingly, in the given example we would choose $S = \{A, B, C, D\}$, $T = \{E, F, G, H\}$.

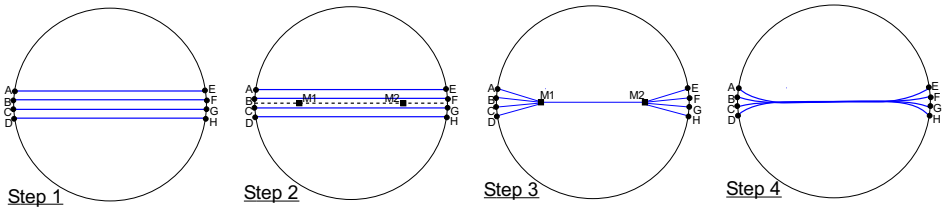

Fig. 3. Non-crossing edges are bundled together thereby freeing up drawing area

The next step is to compute the centroids of S and T, denoted as $\bar{S}$ and $\bar{T}$, respectively. We denote by L the line containing $\bar{S}$ and $\bar{T}$. The prospective bundling should pass along this line. More specifically, we compute two points – M_1 and M_2 – on L such that the bundling is carried out by replacing the original line segments by the following line segments: First, a line from each node of S to M_1, the meeting point of the "sources". Then, a line from M_1 to M_2. Finally, a line from each node of T to M_2, the meeting point of the "targets". See Step 3 in Figure 3. Since we want to reduce the use of ink, the exact positions of M_1 and M_2 minimize the total line length:

$$(M_1, M_2) = \underset{M_1, M_2}{\operatorname{argmin}} \sum_{p \in S} \|M_1 - p\| + \|M_1 - M_2\| + \sum_{p \in T} \|M_2 - p\|$$

We solve this using a numerical method.

At this stage, we can infer if bundling the lines of Q is profitable, as the ink potentially saved is exactly the difference:

$$\sum_{(v_j, u_j) \in Q} \|v_j - u_j\| - \left(\sum_{p \in S} \|M_1 - p\| + \|M_1 - M_2\| + \sum_{p \in T} \|M_2 - p\| \right)$$

If this difference is positive, we know that we gain area by bundling.

If bundling is Q worthwhile, we recommend depicting each line (v_i, u_i) using a Bézier spline with M_1 and M_2 as control points. See Step 4 in Figure 3, and Figure 1(c,e). Our experience is that by incorporating Bézier splines, the drawing is smoother and more readable. Also, the readability of edge bundles is improved when each of them is uniquely colored, as can be seen in Figure 1(c,e).

A possible problem when bundling edges is that we might lose the information about which "source" is connected to which "target". For example, in the final picture of Figure 3, it is unclear whether A is connected to E or maybe to F, G, or H. We adopt a simple rule to address this problem: *crossing edges can never be bundled together*. The edges exit the bundle at the same order they entered it, thus avoiding any ambiguity when each source is connected to a unique target.

Now, we turn to the problem of identifying the sets of edges to be bundled. We pick the sets of edges such that, by bundling them, we minimize the amount of ink used. Our choice is to use a bottom-up, agglomerative approach. The process starts with multiple sets, each of which contains a single edge. Then, sets are merged as long

Function BundlingGain $(Q_1, Q_2 \subset E)$
% Return the ink gain by bundling two edge sets (negative value means no gain)
% The function $\mathrm{Ink}(S)$ returns total ink needed for most efficient drawing of $S \subset E$
  if EdgeCrossing(Q_1,Q_2) **then**
    return -1
  else
    return $\mathrm{Ink}(Q_1) + \mathrm{Ink}(Q_2) - \mathrm{Ink}(Q_1 \cup Q_2)$
  end if
end

Function AgglomerativeBundling $(E = \{e_1, e_2, \ldots, e_m\})$
% Iteratively, grow edge bundles that improve drawing area utilization
  $sets \leftarrow \{\{e_1\}, \{e_2\}, \ldots, \{e_m\}\}$
  while profitable bundling is possible **do**
    % Pick two sets generating most gain:
    $(Q_1, Q_2) \leftarrow \mathrm{argmax}_{Q_1, Q_2 \in sets} \mathrm{BundlingGain}(Q_1, Q_2)$
    $sets \leftarrow sets \cup \{Q_1 \cup Q_2\} - \{Q_1, Q_2\}$
  end
  return $sets$
end

Fig. 4. Agglomerative edge bundling algorithm

as the corresponding bundling improves drawing area utilization; see pseudocode is given in Figure 4.

Concerning computational complexity, this algorithm is essentially a hierarchical clustering algorithm performed on the edges, and therefore it has $O(|E|^2)$ time and space complexity (counting "bundlingGain" calculations), according to Eppstein [5]. The practical situation, however, is better here. First, only a tiny fraction of edge pairs are mergeable since, for most pairs, there is no gain from bundling or the edges cross. Therefore, $O(|E|^2)$ space is unnecessary in practice, and we use a sparse data structure holding only profitable edge pairs. Moreover, when bundling two sets Q_1 and Q_2, we would consider for potential bundling with $Q_1 \cup Q_2$ only sets that could be bundled with Q_1 or Q_2. Finally, the $O(|E|^2)$ time complexity needed for evaluating the bundling gain of all possible edge pairs can be significantly alleviated if we initially consider only bundles involving two nearby edges; other bundles can be considered later by transitivity. Here, two edges e_1, e_2 are considered "nearby" if one of e_1's endpoints is sufficiently close to one of e_2's endpoints in the given circular ordering.

5 Experiments

We evaluated the performance of our methods on the known benchmark set of Rome graphs [2], which contains 11,534 real-life, sparse graphs with 10–109 nodes. In addition, we tested our algorithms on a set of pseudo-random graphs characterized by their average degrees; all these graphs contain 100 nodes.

As a reference, we picked the CIRCULAR algorithm by Six and Tollis [17]. This algorithm finds a circular ordering in two steps. The first step creates an initial ordering based on the largest outerplanar subgraph. Then, the second step iteratively reduces

the number of crossings by carefully moving nodes. We used the publicly available implementation *circo*, which is part of the Graphviz package [8].

Another circular ordering algorithm is that of Baur and Brandes [1]. They also explicitly address edge crossings using a two phase process. The reported numbers of crossings are better – by up to 20% – compared with the aforementioned CIRCULAR algorithm. We did not have an implementation of this algorithm, so no direct comparison was performed.

The quality of the drawings was assessed using two aesthetic criteria: number of crossings and total used ink.[1] The results are given in Tables 1 and 2.

The evaluated algorithms are coded in the tables as follows: **C**=CIRCULAR; **M** = Median iteration followed by fine-tuning, as described in Section 2; **MC** = Median iteration followed by fine-tuning and then by the second step of CIRCULAR.

		No exterior routing			Exterior routing		
Name	#graphs	C	M	MC	C	M	MC
Rome, 10–19 nodes	1407	2.61	3.19	2.11	0.16	0.15	0.09
Rome, 20–29 nodes	839	7.18	8.01	5.51	0.83	0.68	0.46
Rome, 30–39 nodes	2037	21.42	22.17	16.42	4.29	3.33	2.48
Rome, 40–49 nodes	1802	41.49	41.06	31.68	11	8.77	6.66
Rome, 50–59 nodes	1045	66.46	65.16	51.16	20.66	16.67	12.8
Rome, 60–69 nodes	1172	92.76	91.3	72.51	32.4	26.93	21.33
Rome, 70–79 nodes	1008	123.47	120.94	96.23	47.43	39.46	31.04
Rome, 80–89 nodes	788	167.29	161.84	130.41	69.84	58.53	46.73
Rome, 90–99 nodes	1296	209.12	205.64	165.4	92.64	80.24	64.28
Rome, 100–109 nodes	140	230.1	229.52	183.83	103.45	92.74	72.97
Random, avg. deg. 3	100	383.23	357.68	302.29	195.22	166.18	139.12
Random, avg. deg. 4	100	1337.68	1186.50	1048.19	838.42	714.08	627.06
Random, avg. deg. 5	100	2709.35	2489.69	2230.24	1858.20	1678.77	1487.14
Random, avg. deg. 6	100	4437.51	4252.31	3843.23	3192.80	3043.01	2719.80
Random, avg. deg. 7	100	6979.42	6843.71	6210.86	5216.34	5126.31	4594.56
Random, avg. deg. 8	100	9931.27	9808.96	8992.90	7646.76	7545.26	6865.73

Table 1. Comparing number of crossings across different circular ordering options, with and without exterior edge routing

We begin with observations about the circular orderings. In terms of crossings minimization, there is no marked difference between our method (M) and CIRCULAR (C) for the Rome graphs, while M could produce fewer edge crossings than C for the random graphs. As for the edge lengths (Table 2), M consistently achieves better results, which is not surprising as CIRCULAR does not address edge lengths but crossings. Since M does not directly deal with edge crossings, we tried to make it more "crossings aware", by integrating it with the second step of CIRCULAR, obtaining the method coded by MC. As the table shows, MC is consistently the best performer in terms of crossing minimization, outperforming both C and M.

So far, we have compared plain circular orderings. Interestingly, all differences, in terms of number of crossings, are dwarfed by the effect of exterior routing (Section 3).

[1] We prefer the term "total used ink" over the more common "total edge length", since when edge bundling is activated they are no longer equivalent.

Name	#graphs	No edge bundling			Edge bundling		
		C	M	MC	C	M	MC
Rome, 10–19 nodes	1407	12.94	12.34	12.33	10.33	10.11	10.11
Rome, 20–29 nodes	839	17.16	15.49	15.50	13.23	12.52	12.54
Rome, 30–39 nodes	2037	23.12	20.38	20.34	17.46	16.26	16.25
Rome, 40–49 nodes	1802	29.08	25.26	25.19	22.26	19.73	19.68
Rome, 50–59 nodes	1045	34.51	29.59	29.34	24.59	22.71	22.57
Rome, 60–69 nodes	1172	39.04	33.53	33.19	27.39	25.38	25.10
Rome, 70–79 nodes	1008	43.57	37.28	36.58	30.30	27.98	27.56
Rome, 80–89 nodes	788	49.39	42.16	41.33	33.62	31.02	30.58
Rome, 90–99 nodes	1296	53.99	46.21	45.14	36.31	33.63	33.03
Rome, 100–109 nodes	140	56.19	48.68	47.11	37.51	35.08	34.31
Random, avg. deg. 3	100	72.44	62.72	61.99	46.07	43.56	42.59
Random, avg. deg. 4	100	124.08	109.29	107.96	72.04	68.20	66.77
Random, avg. deg. 5	100	171.85	156.19	153.97	93.73	90.08	88.52
Random, avg. deg. 6	100	220.81	206.14	202.87	114.39	111.47	109.45
Random, avg. deg. 7	100	273.35	260.37	254.79	207.10	198.32	194.59
Random, avg. deg. 8	100	325.10	312.25	306.18	243.18	234.58	230.81

Table 2. Comparing total used ink (total length of edges) across different circular ordering options, with and without edge bundling

As can be seen in the right columns of Table 1, exterior routing is capable of eliminating a significant portion of the edge crossings. Also, when exterior routing is activated, our method (M) produces fewer crossings than CIRCULAR (C) even for the Rome graphs, whereas the combined method – MC – is still superior. Apparently, our method can better benefit from exterior routing because, by producing shorter edges, it allows more non-crossing edges to be routed externally. In fact, for the Rome graphs, M allows an external routing of 23% of the edges (on average, surprisingly uniform for all graph sizes), while C allows external routing of 18% of the edges and MC routes 19% of the edges externally.

We see that M has an advantage in reducing the total used ink. When allowing edge bundling (Section 4), a further significant improvement in drawing area utilization is achieved, as shown in the right columns of Table 2. Our experience shows that this ink saving is helpful in conveying a clearer layout. Notice that a further reduction of drawing density could be obtained by exterior routing, but it was not considered in Table 2, as our intention is to isolate the effect of circular ordering and bundling on ink usage.

Finally, as to running time, the average measured running time on the 100-node graphs is around 1 second on a Pentium 4 machine. This is comparable with the running time of the CIRCULAR algorithm. Almost all running time is dedicated to the computation of the circular ordering. The time needed for computing the edge bundling is 50–200ms (depending on the number of edges), whereas the time for computing the external edges is insignificant.

6 Summary

Circular layouts are a rather restrictive layout scheme, offering a simple and highly regularized picture of the graph where nodes cannot be occluded. The limiting nature of circular layouts makes it very important to capitalize on all available degrees of

freedom. In this work, we explored new ways for positioning nodes and routing edges in order to maximize the readability of the layouts. In particular, the density of the drawing is alleviated by shortening edge lengths, moving part of the edges to the exterior of the circle, and bundling some edges together. In addition, shortening edges and exterior routing significantly reduce the number of edge crossings.

References

1. M. Baur and U. Brandes, "Crossing Reduction in Circular Layouts", *Proc. Graph-Theoretic Concepts in Computer-Science (WG '04)*, 332-343, 2004.
2. G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari and F. Vargiu, "An Experimental Comparison of Four Graph Drawing Algorithms", *Comput. Geom. Theory Appl.* **7**, 303–325, 1997.
3. M. Dickerson, D. Eppstein, M. T. Goodrich and J. Meng, "Confluent drawings: Visualizing Non-Planar Diagrams in a Planar Way", *Proc. Graph Drawing (GD'03)*, 1–12, 2003.
4. U. Doğrusöz, B. Madden and P. Madden, "Circular layout in the Graph Layout Toolkit", *Proc. Graph Drawing (GD 96)*, 92–100, 2003, 1996
5. D. Eppstein, "Fast Hierarchical Clustering and Other Applications of Dynamic Closest Pairs", *Journal of Experimental Algorithmics* **5**, 1–23, 2000.
6. M. K. Ganapathy and S. Lodha, "On Minimum Circular Arrangement", *Proc. 21st Annual Symposium on Theoretical Computer Science (STACS'04)*, 394–405, 2004.
7. E. R. Gansner, Y. Koren and S. North, "Graph Drawing by Stress Majorization", *Proc. Graph Drawing (GD'04)*, 239–250, 2004.
8. E. R. Gansner and S. North, "An Open Graph Visualization system and its Applications to Software Engineering", *Software - Practice & Experience* **30**, 1203–1233, 2000. Also, www.graphviz.org.
9. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP Completeness*, Freeman, 1979.
10. K. M. Hall, "An r-dimensional Quadratic Placement Algorithm", *Management Science* **17**, 219–229, 1970.
11. D. Holten and J. J. van Wijk, "Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data", *Proc. IEEE Information Visualization (InfoVis'06)*, to appear.
12. G. Kar, B. Madden and R. S. Gilbert, "Heuristic Layout Algorithms for Network Management Presentation Services", *IEEE Network* 29–36, 1988.
13. M. Kaufmann and R. Wiese, "Maintaining the Mental Map for Circular Drawings", *Proc. Graph Drawing (GD'02)*, 12–22, 2002.
14. V. Liberatore, "Multicast Scheduling for List Requests", *Proc. IEEE INFOCOM 2002*, 1129–1137, 2002.
15. S. Masuda, T. Kashiwabara, K. Nakajima and T. Fujisawa, "On the NP-Completeness of a Computer Network Layout Problem", *Proc. IEEE International Symposium on Circuits and Systems*, 292–295, 1987.
16. F. J. Newbery, "Edge Concentration: A Method for Clustering Directed Graphs", *Proc. 2nd Intl. Workshop Software Configuration Management*, 76–85, 1989.
17. J. M. Six and I. G. Tollis, "Circular Drawings of Biconnected Graphs", *Proc. Algorithms Engineering and Experimentation (ALENEX'99)*, 57–73, 1999.
18. J. M. Six and I. G. Tollis, "A Framework for Circular Drawings of Networks", *Proc. Graph Drawing (GD'99)*, 107–116, 1999.
19. K. Sugiyama, S. Tagawa and M. Toda, "Methods for Visual Understanding of Hierarchical Systems", *IEEE Trans. Systems, Man, and Cybernetics* **11**, 109–125, 1981.
20. A. Symeonidis and I. G. Tollis, "Visualization of Biological Information with Circular Drawings", *Proc. Biological and Medical Data Analysis (ISBMDA'04)*, 468–478, 2004.
21. W. T. Tutte, "How to Draw a Graph", *Proc. London Mathematical Society* **13**, 743–768, 1963.