# New circular drawing algorithms

# New Circular Drawing Algorithms*

Hongmei He and Ondrej Sýkora

Department of Computer Science, Loughborough University
Loughborough, Leicestershire LE11 3TU, The United Kingdom
{H.He, O.Sykora@lboro.ac.uk }

**Abstract.** In the circular (other alternate concepts are outerplanar, convex and one-page) drawing one places vertices of a $n-$vertex $m-$edge connected graph $G$ along a circle, and the edges are drawn as straight lines. The smallest possible number of crossings in such a drawing of the graph $G$ is called circular (outerplanar,convex, or one-page) crossing number of the graph G. This paper addresses heuristic algorithms to find an ordering of vertices to minimise the number of crossings in the corresponding circular drawing of the graph. New algorithms to find low crossing circular drawings are presented, and compared with algorithm of Mäkinen [4], CIRCULAR+ algorithm of Six and Tollis [5] and algorithm of Baur and Brandes [1]. We get better or comparable results to the other algorithms.

## 1 Introduction

A number of data presentation problems involve drawings of graphs on a two-dimensional surface. Examples include circuit schemas, algorithm animations, software engineering, VLSI pin arrangements, and many others. The primary requirement of graph drawing algorithms is that the output graph should be readable, that is, it should be easy to understand and follow. There are many optimisation goals for variation from one application to another and from one human to another, such as minimising crossings, minimising area, minimising bends (in orthogonal drawings), maximising display of symmetries, etc. In the circular [5] (outerplanar [2], convex [7], or one-page [6]) drawing, one places vertices of a $n-$vertex $m-$edge connected graph $G = (V, E)$ along a circle, and the edges are drawn as straight lines. The common task of circular algorithms is to find an ordering $f : V \rightarrow \{0, 1, ...n - 1\}$ of the vertices, minimising the number of edge crossings. Minimum possible number of crossings of any ordering is called circular [5](outerplanar [2], convex [7], or one-page [6]) crossing number of the graph G. We will follow the notation in [6] i.e. one-page crossing number to avoid possible misunderstandings: $\nu_1$(G). Recently, a lot of research was done for circular graph drawing. In this paper, new algorithms to find low crossing circular drawings are presented. We compare performance of our algorithm

* This research was supported by the EPSRC grant GR/S76694/01 and by VEGA grant No. 2/3164/23

with the well known algorithms, such as Mäkinen's algorithm [4] and CIRCULAR/CIRCULAR+ of Six and Tollis [5] and with Baur and Brandes' algorithm [1] on

- our suite of Random Connected Graphs (RCG) with different densities;
- Rome Graphs, which are from the test suite of GDToolkit and were utilised in [1]. We use the undirected graph sets to test our and other previously published algorithms.
  - RND_BUP: this graph set contains about 200 graphs generated randomly. Each graph in the set is biconnected, undirected and planar.
  - ALF_CU: this graph set contains about 10,000 connected and undirected graphs

We get better or comparable results to the other algorithms.

## 2 Previously published algorithms

There are two basic ways to minimise the number of crossings.

- Minimising the circular length of the graph. The problem was proved to be NP-hard [4]. The circular length is defined as follows:

$$\sum_{(u,v)\in E} min(|f(u) - f(v)|, n - |f(u) - f(v)|)$$

  The circular length of the graph shown in the Fig. 1 a) is 53 and the number of edge crossings is 37. See another drawing in Fig. 1 b) where the circular length is reduced to 39 and number of crossings to 10.
- Maximising the number of edges appearing on the circumference of the embedding circle.

The general problem of placing vertices such that the number of edge crossings is minimum, is the well know NP-hard crossing number problem as well as the more restricted problem of Circular Drawing [3].

### 2.1 Algorithm of Mäkinen

Mäkinen [4] proposed an algorithm attempting to minimise the number of crossings by minimising the circular length of the graph. Algorithm of Mäkinen consists of two steps:

- Two vertices with the highest degrees are placed at positions 0 and $n$-1, which correspond to the first position of the right and left halves in the drawing.

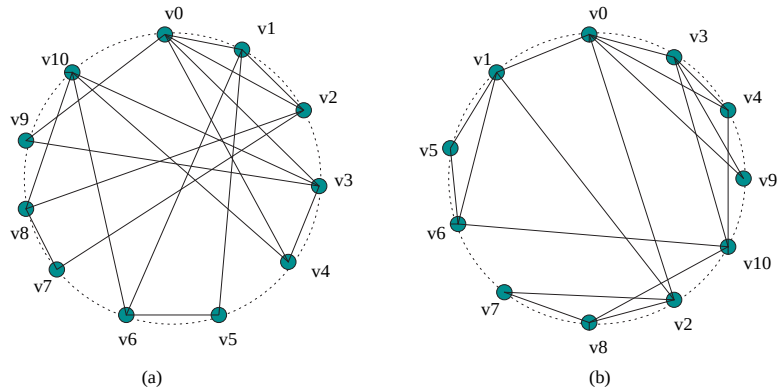


**Fig. 1.** Circular drawings of a graph. Drawing b) was produced by algorithm of Mäkinen

– The other vertices are processed as follows: left and right connectivity arrays for all the vertices not yet placed are maintained, where the connectivity is the number of adjacent vertices already placed on the left or right half in the drawing. The vertex with the highest value of (*right connectivity-left connectivity*) will be placed on the right half. The vertex with the lowest (*right connectivity-left connectivity*) will be placed on the left half.

The running time of the Mäkinen's algorithm is $O(nm)$.

### 2.2 Algorithms CIRCULAR and CIRCULAR+

CIRCULAR [5] algorithm reduces the number of edge crossings by maximising the number of edges appearing on the circumference of the embedding circle. The algorithm first removes one edge from every triangle subgraph in the graph, then places the vertices, which are in the longest path of a Depth-First-Search tree along the embedding circle, and finally builds the corresponding vertex ordering. There is additional phase applied after CIRCULAR and the number of crossings is further reduced by local optimisation of the placement of every vertex. The running time of the algorithm CIRCULAR is $O(mn)$ while the running time of CIRCULAR+ is $O(m^2)$, where $m \geq n - 1$. Fig. 2 shows a drawing produced by the CIRCULAR algorithm for the graph from Fig. 1 a).

### 2.3 Algorithm of Baur and Brandes (BB)

While we were preparing this article a paper of Baur and Brandes [1] was accepted for presentation at WG'04 conference. Therefore we included comparison of our algorithms to their best too. The best algorithm in the Baur and Brandes' article [1] works as follows:

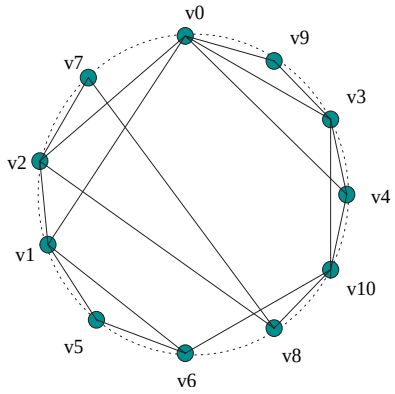


**Fig. 2.** A drawing of the graph from the Fig. 1 a) produced by CIRCULAR algorithm

– Greedy phase: at each step a vertex with the largest number of already placed neighbours is selected, where ties are broken in favour of vertices with fewer unplaced neighbours, and then appended to the end that yields fewer crossings of edges being closed with open edges.
– Sifting phase: Every vertex is moved along a fixed ordering of all other vertices. The vertex is then placed in its (locally) optimal position.

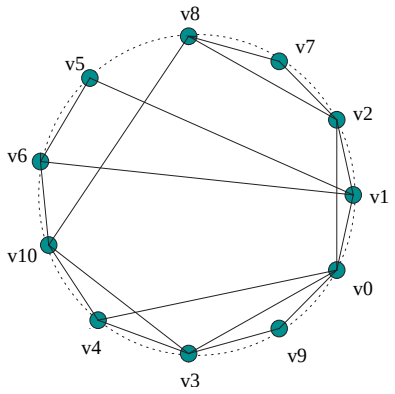


**Fig. 3.** A drawing of the graph from Fig. 1 a) produced by BB-algorithm

We will use BB to denote the greedy phase. The running time of the algorithm BB with sifting is $O(n^2m)$.

# 3 Our Algorithms

## 3.1 Algorithm: Adjacent Vertex with Smallest Degree First (AVSDF)

Depth First Search starts with a vertex as the root, and travels as far as possible along a path emanating from $v$. Then backtracks towards $v$ until it finds a first vertex $u$ from which there is an edge to a vertex $w$ not yet visited. Minimising the circular length of a graph "fuzzily" means making every edge of graph (geometrically a chord of the circle) as short as possible. To achieve this, we change the DFS algorithm so that we first place the vertex with the smallest degree, and then visit the adjacencies of current vertices, which have not been visited yet, such that the smallest degree vertex has highest priority for visiting. Since DFS generates a spanning tree of a graph, our algorithm produces optimal (zero crossing) drawing for any tree. A description of the algorithm AVSDF follows.

**Algorithm 1** Adjacent Vertex with Smallest Degree First

```
1: Initialise an array order[n], and a stack, S.
2: Get the vertex with the smallest degree from the given graph, and push it into S.
3: while (S is not empty) do
4:    Pop a vertex v, from S
5:    if (v is not in order) then
6:       Append the vertex v into order.
7:       Get all adjacent vertices of v, and push those vertices, which are not in order
         into S with descending degree towards the top of the stack (the vertex with
         smallest degree is at top of S).
8:    end if
9: end while
```

The running time of the AVSDF algorithm without counting of crossings is $O(m)$. With counting of crossings its running time increases to $O(n^2)$.
Fig. 4 presents the circular drawing of the graph from Fig. 1 a) produced by AVSDF algorithm.

## 3.2 Preprocessing Phase

For any graph, vertices forming a branch (mutually connected vertices of degree two except one vertex, which is of degree one) do not produce crossing on a circular layout.
We remove these vertices first and then deal with the remaining subgraph using heuristics and finally reinsert the previously removed vertices. For a sparse graphs this preprocessing reduces substantially running time. In comparison of heuristics we first use preprocessing and then apply heuristics AVSDF, BB, and CIRCULAR to the subgraph.

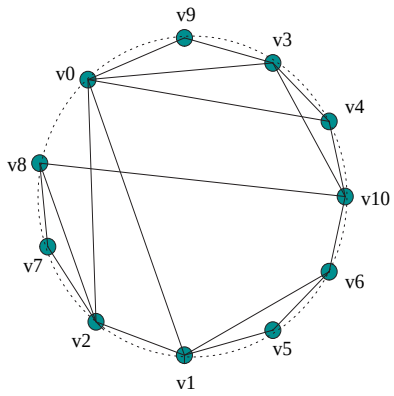


**Fig. 4.** A drawing of the graph from Fig. 1 a) produced by AVSDF-algorithm

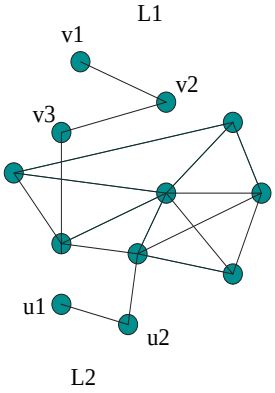


**Fig. 5.** Example of branches, branch $L1$: $v1, v2, v3$, branch $L2$:$u1, u2$.

### 3.3 Postprocessing phase–adjusting

One can further improve the algorithm AVSDF. Taking vertices in the order with descending number of crossings on their incident edges (we start with the vertex whose incident edges create the highest number of crossings), we find its best position among the current one and the ones close to adjacent vertices. We call the procedure adjusting. Its running time is $O(nm)$.

In Fig. 6, the vertex, $v$, whose incident edges create the largest number of crossings, is adjusted first. In this drawing $v$ has three possible positions to try.

In our experiments we combine AVSDF as well as BB with adjusting and sifting.

**Algorithm 2** Local adjusting

1: For every vertex calculate the crossings on edges incident to them.
2: Sort the vertices according to descending number of crossings. Let variable, *currentV*, point to the vertex whose incident edges have the largest number of crossings.
3: **for** (all vertices) **do**
4: Get the positions of adjacent vertices of *currentV* into *pList* array.
5: Try all these positions and calculate the crossing number to find the best location for *currentV*.
6: change the pointer,*currentV* to the next vertex
7: **end for**

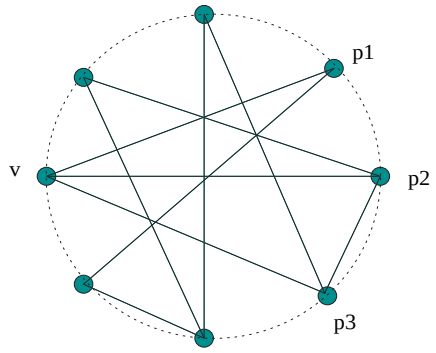


**Fig. 6.** Three possible positions, $p1, p2, p3$, of vertex, $v$, to adjust

### 3.4 Counting of edge crossings

Edges $(i, j)$ and $(k, l)$ intersect if and only if the positions of $i, j$ alternate with the positions of $k, l$. This induces the following formula to calculate the number of crossings.

Given the adjacency matrix $adjMatrix[n][n]$ of the graph $G$, by traversing all edges of the upper triangle of the adjacency matrix, for every edge $e$, we count the number of edges crossing with $e$ :

$$cr(G) = \sum_{i=0}^{n-3} \sum_{j=i+2}^{n-2} \big(adjMatrix[i,j] \sum_{k=i+1}^{j-1} \sum_{l=j+1}^{n-1} adjMatrix[k,l]\big).$$

### 3.5 Generation of the suite Random Connected Graphs

To test our algorithms and compare them with the others we created a suite Random Connected Graphs with different densities. A complete undirected graph on n vertices, has $n(n-1)/2$ edges. We define density $d$ as the ratio of the number of edges in the graph and the number of edges in the complete graph $(n(n-1)/2)$. The algorithm for random connected graph generation is described as follows:

**Algorithm 3** Generation of a random connected graph

{Initialization} :
Create two vertex sets: $uSet[n], vSet[n]$ to store all vertices;
Calculate the number of edges, $eNum = density \times n(n-1)/2$
get a vertex $u \in vSet$ to be stored in $uSet$, and remove it from $vSet$.
{Create a random tree with $n-1$ edges} :
$count = 0$;
**while** $(count < vNum - 1)$ **do**
  Get a vertex $u$, from $uSet$ randomly
  Get a vertex $v$, from $vSet$ randomly
  Set the corresponding element in $adjMatrix$ to 1
  Put $v$ into $uSet$, and remove it from $vSet$
  $count++$
**end while**
{Add remaining edges} :
**while** $(count < eNum)$ **do**
  generate an edge $e$, which is not in the current graph
  $count++$
**end while**

## 4 Experiments with algorithms

### 4.1 Test Suite

– Random Connected Graphs.
We used three densities 1%, 3%, and 5%. For each density, 12 groups of graphs with different number of vertices were tested; and for every group 10 different graphs were generated and average running time and average number of crossings were calculated.

– Rome graphs
Rome graphs are several sets of graphs from GDToolkit. Here we use two sets of graphs, RND_BUP and ALF_CU. RND_BUP is a set of random biconnected undirected planar graphs. ALF_CU is a set of connected undirected graphs. For these graphs we first apply preprocessing phase as described in subsection 3.2 and then on the resulting graph run heuristics.

### 4.2 Comparison of AVSDF, algorithm of Mäkinen, CIRCULAR, and BB

In this subsection we present the results of comparison the algorithms AVSDF, Mäkinen's, CIRCULAR and BB without any postprocessing (adjusting, sifting or local optimisation) improvements.

We carried out experiments on the test suite Random Connected Graphs with densities 1%, 3% and 5%. AVSDF and BB algorithms produced lower crossing drawings than the other two ones. For density 1% (maximal number of vertices

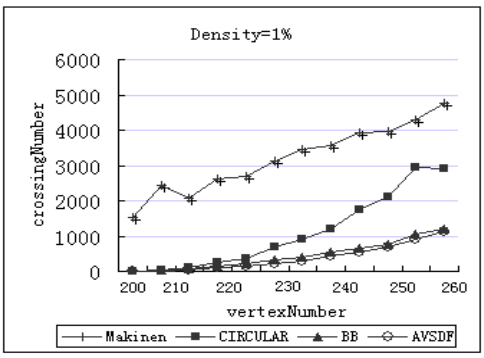


**Fig. 7.** Test results for density 1%

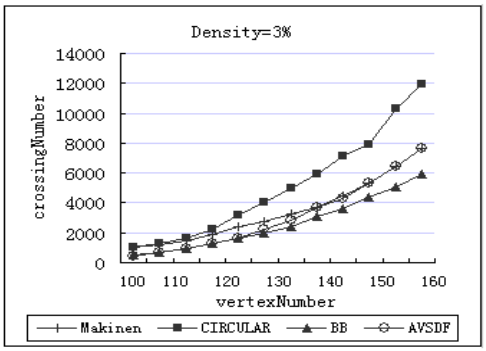


**Fig. 8.** Test results for density 3%

in a graph was 260), for density 3% and $n < 120$, and for density 5% and $n < 50$ AVSDF produced better results than BB.

### 4.3 Comparison of AVSDF and BB with different postprocessing

We combined the algorithm AVSDF and BB with adjusting and sifting to get the combinations: AVSDF with adjusting, AVSDF with sifting, AVSDF with adjusting and sifting, AVSDF with sifting and adjusting, BB with adjusting, BB with adjusting and sifting, BB with sifting and adjusting and compared them to the algorithm BB with sifting which was the best algorithm produced in [1]. We make adjusting and/or sifting round only once in every combination. The results are in Table 1, where each unit has three values. The first (second, third) value is the percentage of graphs where the corresponding combination of algorithms

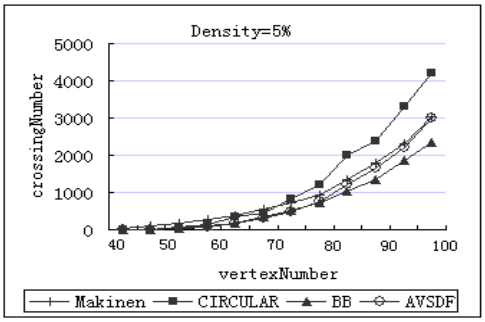


**Fig. 9.** Test results for density 5%

achieved smaller (same, greater) number of crossings than BB + sifting. We also calculate the total percentage for all graphs.
Table 2 displays how many times a combination of algorithms produces the best result.

Table1. (Density given in brackets)

| Graphs | Avsdf +sift | Avsdf +adjust | Avsdf +adjust +sift | Avsdf +sift +adjust | BB +adjust | BB +adjust +sift | BB +sift +adjust |
|---|---|---|---|---|---|---|---|
| RCG (5%) | 32,9,59 | 30,10,60 | 44,9,47 | 43, 9,48 | 20,17,63 | 59,24,17 | 67,33,0 |
| RCG (3%) | 17,0,83 | 19,0,81 | 35,1,64 | 32,0,68 | 39,1,60 | 93,0,7 | 100,0,0 |
| RCG (1%) | 56, 0, 44 | 52,1,47 | 55,0,45 | 56,0,44 | 5,5,90 | 63,24,13 | 63,37,0 |
| RND_BUP | 43,17,41 | 39,17,44 | 45,16,39 | 44,17,39 | 6,46,48 | 29,66,5 | 21,79,0 |
| ALF_CU | 38,24,38 | 33,21,46 | 40,24,36 | 41,24,35 | 10,30,60 | 31,62,7 | 25,75,0 |
| TOTAL | 38,13, 49 | 36,11, 53 | 43,13, 44 | 43,13, 44 | 14,23,63 | 49,45, 6 | 47,53, 0 |

Table2 How many times a combination of algorithms produces the best result

| Graphs | Avsdf +adjust | Avsdf +adjust +sift | Avsdf +sift | Avsdf +sift +adjust | BB +adjust | BB +adjust +sift | BB +sift | BB +sift +adjust |
|---|---|---|---|---|---|---|---|---|
| RCG (5%) | 19 | 39 | 23 | 36 | 14 | 54 | 22 | 48 |
| RCG (3%) | 0 | 16 | 2 | 15 | 0 | 57 | 0 | 34 |
| RCG (1%) | 21 | 51 | 54 | 62 | 0 | 30 | 7 | 33 |
| RND_BUP | 67 | 95 | 86 | 92 | 48 | 91 | 71 | 81 |
| ALF_CU | 93 | 145 | 129 | 146 | 70 | 153 | 121 | 144 |
| TOTAL | 200 | 346 | 294 | 351 | 132 | 385 | 221 | 340 |

### 4.4 Comparison of AVSDF with adjusting and BB with sifting done more than once

The test was similar to the one above, but the adjusting and sifting phases were run more than once - until no improvement was achieved. This number was surprisingly low! Generally it was about 5 and always less than 8. AVSDF with adjusting produced for graphs from RND_BUP in 35% better, in 18% equal and in 47% worse results than BB with sifting. For ALF_CU graphs, AVSDF with adjusting produced in 29% better, in 22% equal and in 49% worse results than BB with sifting. In case of RCG suite with density 1% (3%, 5% ) the percentages were 47, 3, 50 (21, 0,79; 30,10,60).

### 4.5 Running time

In this subsection we compare the average running time of AVSDF, BB, Mäkinen and CIRCULAR on RCG with different densities,5%,3%,1%, RND_BUP, and ALF_CU of Rome Graphs. AVSDF has absolute superiority for all graphs (see table 3).
This can be explained so that the average degree of a random connected graph with density $d$ is $adeg = d(n-1)$, and this is the number of positions which has to be tried in adjusting postprocessing. Especially for sparse graphs, ajusting takes much less running time than sifting, which tries $n$ positions. The high running time of sifting might make it impossible to use for larger graphs and multiple runs.

Table 3 Average running time of AVSDF, Mäkinen, CIRCULAR and BB on different graphs

| Graphs | AVSDF (ms) | Mäkinen(ms) | CIRCULAR(ms) | BB(ms) |
|---|---|---|---|---|
| RCG (density=5%) | 2 | 16 | 17 | 14 |
| RCG (density=3%) | 5 | 24 | 38 | 80 |
| RCG (density=1%) | 13 | 56 | 130 | 295 |
| RND_BUP | 1 | 1 | 2 | 5 |
| ALF_CU | 1 | 1 | 3 | 3 |

## 5 Conclusion

In this work we designed a new algorithm, AVSDF+, to produce circular drawings, and carried out comparisons with previously published algorithms. AVSDF and BB [1] algorithms without postprocessing (adjusting, sifting or local optimisation) produce better results than the well known algorithm of Mäkinen [4] and CIRCULAR of Six and Tollis [5]. For lower densities AVSDF produces better

results than BB.

We combined the algorithm AVSDF with postprocessing adjusting and sifting to get the combinations: AVSDF with adjusting, AVSDF with sifting, AVSDF with adjusting and sifting, AVSDF with sifting and adjusting and compared them with the algorithm BB with sifting which was the best algorithm designed in [1]. The results of experiments show that that AVSDF combined with one or two operations of postprocessing produces approximately same results as BB combined with same operations.

We carried out similar tests on the adjusting and sifting phases run more than once (until no improvement was achieved) for AVSDF with adjusting and BB with sifting. The results show that BB with sifting is slightly better than AVSDF with adjusting but from the the running time point of view the AVSDF with adjusting algorithm is much faster than the algorithm BB with sifting. Another interesting fact was that the number of rounds was low - less than 8 for all tested graphs.

## References


1. Baur, M., Brandes, U., Crossing Reduction in Circular Layouts, accepted to WG'2004.
2. Kainen, P.C., The book thickness of a graph II, *Congressus Numerantium*, **71** (1990), 121–132.
3. Masuda, S., Kashiwabara, T., Nakajima, K., Fujisawa, T., On the *NP*-completeness of a computer network layout problem, in: Proc. *IEEE Intl. Symposium on Circuits and Systems 1987*, IEEE Computer Society Press, Los Alamitos, 1987, 292–295.
4. Mäkinen, E., On circular layouts, *International Journal of Computer Mathematics* **24** (1988), 29-37.
5. Six, J.M., Tollis, I.G., Circular drawings of biconnected graphs, in: *ALENEX'99*, LNCS **1619**, Springer, 1999, 57-73.
6. Shahrokhi, F., Sýkora, O., Székely, L.A., Vrťo, I.: The book crossing number of a graph, *Journal of Graph Theory* **21** (1996), 413-424.
7. Shahrokhi, F., Sýkora, O., Székely, L.A., Vrťo, I., Towards the theory of convex crossing numbers, in: *Towards a Theory of Geometric Graphs*, ed. J. Pach, Contemporary Mathematics, **342** American Mathematical Society, Providence 2003, 249–258.