A circular layout algorithm for attributed hierarchical graphs with ports

V N Kasyanov, A M Merculov and T A Zolotuhin

A.P. Ershov Institute of Informatics Systems SB RAS, Ac. Lavrentiev ave., 6, Novosibirsk, 630090, Russia
E-mail: kvn@iis.nsk.su

Abstract. Information visualization based on graph models is a key component of support tools for many applications in science and engineering. The Visual Graph system is intended for visualization of big amounts of complex information on the basis of attributed hierarchical graph models. In this paper, a circular layout algorithm for attributed hierarchical graphs with ports and its effective implementation in the Visual Graph system are presented.

1. Introduction

Graphs are widely used to describe and structure information in subject areas, where it is necessary to model connections between objects. Graph visualization techniques which embed graphs onto a two or three-dimensional surface where vertices are represented by points, boxes or ellipses and arcs are represented by curves connecting them visualization, have many areas of application [1, 4, 6].

For visual analysis of data and connections between them, it is required to build graphical representations vary greatly according to the application domain and taking into account a variety of so-called aesthetic criteria. For example, planarity and the display of symmetries are often highly desirable in graph visualization for many applications. In general, in order to improve the readability of graph drawings, it is important to keep the number of arc-arc crossings low as possible. Also, to avoid wasting of space on screen (or page), it is important to keep area of the graph drawing as small as possible. Trying to satisfy all criteria is generally infeasible if not impossible, as they are inherently conflicting, and some trade-offs are usually used [1].

In some application areas the organization of information is too complex to be modeled by classical graphs, and some more powerful graph formalisms and new methods for hierarchical visualization have been introduced [2, 3, 4, 5, 10, 13]. A practical and general graph formalism called hierarchical graphs and graph models can handle many applications with complex information, and is amenable to graph drawing [5, 7, 9]. It forms a basis for natural “abstraction” and “reduction” techniques that can be applied by researchers in order to reduce the visual complexity of a large graph. It divides the graph into parts (called fragments) so that all elements of each fragment are similar in some sense and all considered fragments form a hierarchy of nested graph parts. This hierarchy provides a good means of handling size complexity for those applications which are obliged to deal with large amounts of data.

The Visual Graph visualization system created at the A.P. Ershov Institute of Informatics Systems SB RAS is based on attributed hierarchical graph models and allows exploring complex structured large data through their visual representations [8].

In many applications, the objects modeled by the vertices of the graph contain disjoint logical locations (so-called ports [14]), through which they (objects) are in the relationship that is modeled by
arcs. For example, in program graphs that represent the flow of data in a program, the vertices represent program statements, the ports of the vertices represent the operands of the statements (their arguments and results) and arcs connected ports represent data flow between results and arguments of statements [8].

The readability of the resulting graph image strongly depends on the way its elements (vertices and arcs) are laid on the plane. A circular graph drawing is an embedding of a graph on the plane such that all vertices are placed onto the circumference of a circle and each arc is drawn into the circle usually as a straight line [12], but the problem of producing a circular drawing with a minimum number of arc crossings is NP-Complete [11]. Circular layout finds its application in areas where the objects represented by the graph vertices are of equal priority, and none of them occupies a privileged position. Circular graph layouts are used to visualize ring and stellar network topologies, biological and social networks, and smaller clusters in a large graph. Since these applications deal with big data, the problem of developing a circular layout algorithm for general hierarchical graphs is very relevant, but at present only for clustered graphs being some simple hierarchical graphs without ports a circular layout algorithm has been developed [15].

In this paper an algorithm for circular layout of attributed hierarchical graphs with ports and an extension for the Visual Graph system that implements this algorithm are described. Two heuristics are considered to minimize arc-arc intersections. To solve the arc-to-vertex intersection problem, rounded arc inserts are used.

The rest of the paper is organized as follows: Section 2 considers the attributed hierarchical graphs with ports and their drawings, Section 3 describes our circular layout algorithm for the attributed hierarchical graphs, Section 4 presents the algorithm implementation in the Visual Graph system, and Section 5 concludes the paper and highlights the future research directions.

2. Attributed hierarchical graphs with ports and their drawings
To better describe the algorithm, we first introduce some notations from [5]. Let $G$ be a graph of some type, e.g. $G$ can be an undirected graph or a directed graph. A graph $G$ is defined by two finite sets $V$ and $E$, where the elements of $V$ are the vertices of $G$, and the elements of $E$ are the arcs of $G$. $G$ is a trivial graph if $|V| = 1$ and $|E| = 0$.

A graph $C$ is called a fragment of $G$, denoted by $C \subseteq G$, if $C$ includes only elements (vertices and arcs) of $G$. A set of fragments $F$ is called a hierarchy of nested fragments of the graph $G$, if the following two properties hold: (1) $F$ contains the graph $G$ and all vertices $V$, and the elements of $E$ are the arcs of $G$. $G$ is a trivial graph if $|V| = 1$ and $|E| = 0$.

A hierarchical graph $H = (G, T)$ consists of a graph $G$ and a rooted tree $T$ that represents an immediate inclusion relation between elements of a hierarchy $F$ of nested fragments of $G$. $G$ is called the underlyng graph of $H$. $T$ is called the inclusion tree of $H$.

A hierarchical graph $H$ is called a connected one, if each fragment of $H$ is connected graph, and a simple one, if all fragments of $H$ are induced subgraphs of $G$.

It should be noted that any clustered graph [2] can be considered as a simple hierarchical graph $H = (G, T)$ such that $G$ is an undirected graph and any fragments $C \in F$ is considered as the set of vertices of $C$.

Let $P$ be a finite set of ports. A port $p \in P$ is a special vertex of a hierarchical graph which is attached to a fragment $C \in F$ and serves to designate the point where the arc enters the anchored fragment $C$. It is assumed that each arc of the hierarchical graph with ports is an arc between two ports.

Let $P_i(C)$ be an ordered set of ports of a fragment $C$ and $P_i(C)$ be the port of $C$ with number $i$.

An arc $u = (P_i(w), P_j(z))$ breaks a fragment $C$ ($u$ is a breaking arc) if $C$ includes an element from the set $\{w, z\}$ and does not include the other element.

A hierarchical graph $H$ with ports is a structural one if it has no breaking arc.
Let \( u = (P_i(w), P_j(z)) \) be a breaking arc of a fragment \( C \). Transformation of removal from \( H \) of breaking arc \( u \) is defined as replacing in \( H \) the arc \( u = (P_i(w), P_j(z)) \) by two arcs \( (P_i(w), P(v)) \) and \( (P(v), P_j(z)) \) where \( P(v) \) is a new dummy port added to the fragment \( C \).

\( H_1 \) is a structural extension of a hierarchical graph \( H_2 \) if \( H_1 \) is a structural hierarchical graph which can be constructed from \( H_2 \) by iterative application of breaking arc removal transformation.

![Visual Graph system.](image)

**Figure 1.** Visual Graph system.

A drawing (or layout) \( D \) of a hierarchical graph \( H \) is a representation of \( H \) in the plane such that the following properties hold.

1. Each port, vertex and fragment of \( G \) is represented by a simple closed region (a cycle in our case). The region is defined by its boundary (a simple closed curve in the plane) which divides the plane into two parts: inside face and outside face.
2. For any \( C_1, C_2 \in F \) the intersection of regions of \( C_1 \) and \( C_2 \) is empty if and only if \( C_1 \cap C_2 = \emptyset \), and the region of each fragment \( C \in F \) includes all regions of all fragments nested into it.
3. Any two ports of each fragment \( C \) are represented by two nonintersected regions. The region of each port of \( C \) is divided into three nonempty sets of points: points of the inside face of region of \( C \), points of its boundary and points of its outside face.
4. Each arc of \( H \) is represented by a simple curve between the drawings of its endpoints (ports).
5. All arcs of any fragment \( C \) of \( H \) are located inside the region of \( C \).

A drawing \( D \) of a hierarchical graph \( H \) is a circular one if for any fragment from \( F \) all its sons in \( T \) (or all fragments \( C \) directly nested into it) are placed onto the circumference of a circle.

### 3. Circular layout algorithm

Below it is assumed that the input graph is a hierarchical graph \( H = (G, T) \) with ports, and the algorithm output is a circular drawing \( D \) of a structural extension of \( H \).
The algorithm performs traversing the nesting tree $T$ in depth. At each step of traversing, a certain fragment $C$ being considered as a vertex of $T$ is drawn using the sizes and positions of the fragments directly nested into it. Note that after building an image for a fragment, it is possible to calculate its sizes and use them when drawing a fragment in which it is nested will be performed.

The algorithm of the drawing of a fragment is consists of following three stages.
1. Calculation of the fragment sizes.
2. Ordering fragments relative to each other.
3. Layout of the arcs.

To construct a circular image, it is proposed to divide the algorithm into three passes through all levels of the hierarchy represented by $T$. This is due to the fact that the calculation of the fragment sizes and the laying of the arcs must be done in a different order since the calculation of the sizes should starts with the leaves of the nesting tree $T$, but the layout of the arcs should starts with its root.

![Figure 2. Drawings of simple graphs.](image)

3.1. Calculation of the fragment sizes
The size of all trivial fragments (or leaves of $T$) is already specified at the input to the algorithm. For other fragments, the sizes can be calculated only taking into account the sizes of the fragments lying inside. It follows from this that the first stage should be carried out from a low level of the hierarchy to a high one, since first you need to calculate the sizes of the inner fragments. At this stage, the following three steps should be performed for a given set of fragments.

1. The radius of the circle on which the given set of fragments will be located is calculated. This is possible because the sizes of all given fragments are already known. The minimum radius of a circle that could cover any given fragment is calculated and used.

2. All given fragments are evenly spaced around the circumference of the calculated radius.

3. If the given fragments are internal to a fragment distinct from $G$, then the size of this fragment is calculated. After that, real (or nondummy) ports, if any, are evenly placed around the circumference of the fragment.

If we consider dummy ports, then an arc must connect with an outside port, because this is a condition for the existence of a dummy port. In the case of real ports, this is not the case; they can only connect to the inner ports of the fragment. Accordingly, it is required to set positions for ports for which there are no external arcs. But defining external relations is a rather expensive operation. Therefore, the positions are set for all ports, but later (at the stage of laying the arcs) some ports can be interchanged.
3.2. Ordering fragments relative to each other

The task of this stage is to find such an order of the fragments on the circle that the number of intersections of arcs is minimal. It should be noted that the number of intersections of arcs in the circular graph drawing does not depend on the final coordinates of the fragments, but depends only on their relative position on the circle (their ordinal number). Thus, this problem is combinatorial, not geometric. However, this problem is NP-complete [11]. Therefore, below we will give two heuristics that find the order of the fragments in polynomial time, but do not guarantee the best result on any graph. Let's designate them as permutation and grouping heuristics, based on the peculiarities of each of them.

![Figure 3. Drawings of graphs with ports.](image)

The grouping heuristic is based on the assumption that the number of intersections between arcs will decrease if the most connected fragments are placed side by side. This location of the fragments leads to a decrease in the number of arcs lying close to the centre of the circle. As a rule such arcs cause a large number of intersections, because their length is close to the maximum possible in the given drawing. The grouping heuristic consists of the following six steps.

1. Let $S$ be a set of all given fragments.
2. Let $C$ be the most connected fragment from $S$.
3. A group of all given fragments directly connected with $C$ is selected.
4. All fragments of the group are removed from $S$.
5. The fragments of the group are sorted by the number of links between them and placed so that the least connected fragments are located “outside”.
6. If the set $S$ is not empty, then the algorithm continues from step 2.

It should be noted that this approach leads to an increase in the number of arcs lying on the boundary of the drawing circle and has a quadratic time complexity.

The algorithm of the permutation heuristic uses cyclic permutation of neighbouring fragments to minimize intersections. The permutation heuristic consists of the following four steps.

1. Let $S$ be a sequence of all given fragments.
2. The algorithm considers step by step all adjacent pairs of fragments from $S$ beginning from $(C_1, C_2)$ and reforms step 3 for each considered pair. The last pair under consideration consists of the last and first elements of $S$.
3. Incident arcs are considered for the current pair of fragments. The number of their intersections with other arcs is calculated. If the permutation of the fragments in the considered pair leads to a decrease in their intersections, then the fragments are swapped.
4. If the fragments of some pair have been swapped, then the algorithm continues from step 2.

This algorithm is inefficient in time, but in many cases it reduces intersections as much as possible. It is suitable for use on small graphs where the lack of speed is not noticeable.
3.3. Algorithm for arc layout
Arcs are defined as polylines passing through so-called bend points. At this stage, a set of these points is calculated for each arc and includes the beginning point and the end point of the arc. These points can be calculated on the base of the relative positions of the incident ports and the angle between the arc and the abscissa.

Table 1. Comparison of two algorithms with respect to number of intersections of arcs.

| Simple graph          | Our algorithm with the grouping heuristic | The circular graph drawing algorithm of the yEd system |
|-----------------------|------------------------------------------|-------------------------------------------------------|
| 8 vertices, 6 arcs    | 0                                        | 0                                                     |
| 10 vertices, 10 arcs  | 5                                        | 3                                                     |
| 14 vertices, 22 arcs  | 33                                       | 20                                                    |
| 40 vertices, 39 arcs  | 1                                        | 13                                                    |

If the arc is associated with a port at a lower level, then its position is also determined. The layout process goes from a high level of hierarchy to a low one, because to lay arcs, it is needed to know the coordinates of the ports, which are determined on the base of the arcs of a higher level. At this stage, the following six steps should be performed.

1. There is an empty list at the input, to which control points will be added.
2. The point “start of the arc” is calculated and added to the beginning of the list.
3. If there is a point at which the arc approaches to a non-incident fragment on a given distance, then points projecting the arc onto a circle of some radius centred at this fragment are added to the list.
4. Design ends when the arc leaves the circle. The “end of arc” point is calculated and added to the end of the list.
5. If the fragment from which the arc originates has a corresponding port, then it is assigned the “start of arc” position.
6. Similarly, if the fragment to which the arc arrives has a corresponding port, then it is assigned the “end of arc” position.

When drawing arcs, there may be a problem with crossing a fragment by the arc at steps 3 and 4. To solve this problem, rounded arc inserts were chosen. Arcs are defined by a set of bend points.
through which they must pass. If the arc approaches to a non-incident fragment on a given distance, then at this place bend points are added, projecting the arc onto a circle with a centre at the fragment and a radius. These points are added until the arc leaves the circle. This solution does not increase the area of the graph. For graphs with a large number of intersections, this does not lead to an aesthetic improvement in the image, since the general arrangement of arcs is chaotic and local improvements are not noticeable. In addition, it is time consuming. Thus, for sufficiently large graphs, steps 3 and 4 should be removed from the arc packing algorithm.

Figure 5. Graph drawings without grouping heuristic (left) and with grouping heuristic (on right).

4. Implementation of the algorithm

The presented algorithm was implemented as an extension to the Visual Graph system. For a better understanding, it is necessary to say a few words about this system. Visual Graph is a universal system for visualization of attributed hierarchical graphs, which is distributed under the BSD license and is intended for visualization of large-scale complex-structured information based on graph models. The nomenclature of these graph models may differ from one design system to another. Moreover, the representations of the same graph models can differ among themselves in different systems and even in different versions of the same system. However, all these graph-theoretic models are special cases of attributed hierarchical graphs. Therefore, the following scenario for using the Visual Graph system is assumed. The Visual Graph system reads a graph model from a file, one of the supported formats (GraphML, DOT or GML), and then provides the user with navigation tools, as well as algorithms for visualizing its parts.

The Visual Graph system has a mechanism for adding extensions, with the help of which a circular drawing extension was added. Figures 2, 3 and 4 shows a few graph drawings which have been obtained by this extension. Some typical graph constructions were selected for these examples: simple graphs of varying complexity, graphs with ports, and hierarchical graphs.

Some comparisons of our grouping heuristic algorithm with the circular graph drawing algorithm of the yEd system [15] for some simple graphs and with our algorithm without heuristic are represented in table 1 and figure 5. As you can see from results, there is no clear superiority of one algorithm, and to visualize simple graphs, it is need to select an algorithm based on their individual characteristics.
5. Conclusion
The paper describes the created circular layout algorithm that draws attributed hierarchical graphs with ports in quadratic time and its effective implementation in the Visual Graph system. The algorithm takes into account the criterion for minimizing intersections and uses rounded arc inserts for solving the arc-to-vertex intersection problem. Its ability to draw graphs with ports is very important for many applications and is new for circular layout algorithms.

In the future, the algorithm can be improved in the following directions. First, we plan to improve sorting heuristic. For example, it is possible to make by adding sorting for adjacent levels or by improving intersection minimization. Second, we plan to take into account and preserve the initial order of fragment ports. We plan also to reduce the area of the drawing graph due to the economical distribution of fragments of hierarchical graphs on circles and extend the algorithm by adding a component which focuses on drawing acyclic fragments.

References
[1] Di Battista G, Eades P, Tamassia R and Tollis I G 1999 Graph Drawing: Algorithms for Visualization of Graphs (Prentice Hall)
[2] Feng Q W, Cohen R F and Eades P 1995 Planarity for clustered graphs Lecture Notes in Computer Science 979 213–226
[3] Harel D 1988 On visual formalism Comm. ACM 31(5) 514–530
[4] Herman I, Melançon G and Marshall M S 2000 Graph visualization and navigation in information visualization: a survey IEEE Trans. on Visualization and Computer Graphics 6 24–43
[5] Kasyanov V N and Kasyanova E V 2013 Information visualization based on graph models Enterprise Information Systems 7(2) 187–197
[6] Kasyanov V and Kasyanova E 2014 Information visualization on the base of graph models Scientific Visualization 6(1) 31–50
[7] Kasyanov V N and Kasyanova E V 2015 Graph- and cloud-based tools for computer science education 2015 Lecture Notes of Computer Science 9395 41-54
[8] Kasyanov V N and Zolotuhin T A 2018 A system for visualization of big attributed hierarchical graphs Intern. Journal of Computer Networks & Communications 10(2) 55–67
[9] Kasyanov V N, Zolotuhin T A and Gordeev D S 2019 Visualization methods and algorithms for graph representation of functional programs Programming and Computer Software 45(4) 156–162
[10] Macquisten A, Smith A M and Fernstad S J 2020 Evaluation of hierarchical visualization for large and small hierarchies 2020 24th Intern. Conf. Information Visualisation (IV) pp 166-173
[11] Masuda S, Kashiwabara T, Nakajima K and Fujisawa T 1987 On the NP-completeness of a computer network layout problem Proc. IEEE 1987 International Symposium on Circuits and Systems (Philadelphia: PA) pp 292–295
[12] Six J M and Tollis I G 1999 A framework for circular drawings of networks Lecture Notes in Computer Science 1731 107–116
[13] Sugiyama K and Misue K 1999 Visualization of structured digraphs IEEE Trans. on Systems, Man and Cybernetics 21(4) 876–892
[14] The GraphML file format, http://graphml.graphdrawing.org
[15] The yEd system, http://www.yworks.com/