Technical Section

An advanced pre-positioning method for the force-directed graph visualization based on pagerank algorithm

Wenqiang Dong 1, Fulai Wang 1,*, Yu Huang, Guangluan Xu, Zhi Guo, Xingyu Fu, Kun Fu

Department of Key Laboratory of Technology in Geo-spatial Information Processing and Application System, Institute of Electronics, Chinese Academy of Sciences, No. 19, North 4th Ring Road West, Haidian District Beijing 100190, PR China

ABSTRACT

Graph visualization techniques aim to help users obtain deep insights about the structure and evolution of graphs through minimizing visual confusion and real-time updating of the drawing. This demands high effectiveness of the drawing algorithms to produce layout of graphs with few edge crossings. However, the classical force-directed algorithms fail to satisfy this requirement. In this paper, we solve this problem by using a novel and heuristic pre-positioning method based on a variant of the PageRank algorithm. In our algorithmic framework, five models are proposed to calculate the importance degree of each vertex, the relation matrix of the vertices, the layer each vertex belongs to, the available placing coordinates and the matching coefficient between a vertex and a position. After the pre-positioning procedure of each layer, the classical force-directed algorithm is used to calculate the final position of each vertex. Experimental results have shown that our approach can reduce the computing time as well as the number of edge crossings.

1. Introduction

Graphs have become one of the most widely used models to represent structural information as diagrams. In order to communicate information contained in the graph clearly, graph visualization has become a fast growing research area. Typically, graph visualization aims at visualizing the abstract data to reinforce human cognition. It has a broad application domain including scientific computing, network analysis, bioinformatics and network security.

As Purchase et al. described in their literature, the process of human understanding was in close relation to the quality of the final layout [1,2]. A good layout of graph can show the underlying relationships between the entities clearly, so that the readers can understand the information contained in the graph quickly. In contrast, a bad layout can make the structure of the graph confusing, and the readers will get a poor graph reading experience with more time spent. Empirical experiments performed by Purchase has shown that reducing the number of edge crossings is the key to getting a good layout as well as increasing the understandability of the graph [3].

In order to map the relational data structure as graphs, many graph layout algorithms have been proposed. Due to the ability to produce graphs in an aesthetically pleasing way, the force-directed graph drawing algorithms [4,5] are the most widely used. A typical force-directed graph drawing algorithm, like Eades’ spring embedder, treats a graph as a mechanical system in which vertices are represented as steel rings and edges are represented as springs [6]. In this system, repulsive forces exist between any two vertices and attractive forces exist between two connected vertices. When the spring length is longer than the natural length, vertices are pushed apart. In this way, the repulsive forces nullify the attractive forces so that two connected vertices maintain perfect distance to each other. After the vertices are placed at randomly chosen positions, the repulsive and attractive forces act on these vertices, and they are forced to move in the direction of the total force. However, the force-directed algorithms are very computing expensive because iterations are needed to find a minimal energy state. In addition, the final drawings often have lots of edge crossings which will reduce the readability [7]. Thus, the demand for high effectiveness of the graph drawing algorithms to produce comprehensible graphs in a short time cannot be satisfied.

In addition to the existing online visualization software [8–11], many algorithms have been proposed to improve the performance of the force-directed algorithms. Koren used the spectral approach to reduce the time consumption and optimize the layout within a
small vector space [12], Tominski et al. combined dynamic filtering, graph lenses, and edge-based navigation to support different visualization tasks so as to improve the interactivity [13]. Shamir and Stolpnik presented the notion of visual queries as a graphical interface to visualize and interact with multivariate graphs [14]. They displayed the result in a pop-up view to promote view consistency. To visualize small world graph, van Ham and van Wijk proposed a method combined with both semantical and geometrical distortions [15]. For large graphs, Hosob used high-dimensional approach to allow users to focus on interesting parts of graphs [16]. In order to reduce the edge crossings, Six and Tollis proposed an algorithm to produce drawings with low edge crossings first, then they used a postprocessing step to visit each vertex and determine whether the crossings could be reduced by moving that vertex next to one of its neighbors [17].

Exploring the convergence of the algorithms is also an important aspect of the force-directed methods. Gansner et al. introduced the majorization approach in the framework of graph layout [18]. They combined three extensions to the stress function of Kamada and Kawai [19] and speeded up the convergence rate by monotonically decreasing the stress until convergence. Hua et al. proposed a pre-positioning method to reduce the convergence time [20]. They precalculated the position of all the vertices according to the number of edge crossings each vertex had with other vertices before applying a force-directed algorithm.

In this paper, we propose an advanced pre-positioning method to reduce the time consumption and the number of edge crossings. In our algorithmic framework, we use a variant of the PageRank link analysis algorithm, which is called VertexRank, to calculate the weight of the vertices in the graph. The center vertices, which have local maximal weight, are positioned first. The other vertices are classified into different layers depending on the number of connecting edges to the upper layer. Then, the relation matrix of the vertices is calculated. After that, the available positions for the vertices in each layer are calculated and the layer vertices are placed according to their matching coefficient with these positions. At last, the classical force-directed algorithm is applied to determine the final position of the vertices in each layer. Experimental results have shown that our approach can reduce both the number of edge crossings and time consumption. In addition, the number of edge crossings of our approach is more stable than the classical method.

The rest of this paper is organized as follows. In Section 2, we briefly review the related work about the graph drawing algorithms. Section 3 describes our pre-positioning approach, followed by Section 4, where experimental results are shown. In Section 5, we present the conclusion of this paper with topics for future work.

2. Related work

The force-directed algorithm has a long history of development. In addition to its simplicity, it has good flexibility as well as strong theoretical foundations.

The first graph drawing algorithm was proposed by Tutte [21]. The objective of this algorithm is to produce crossings-free drawings for graphs. Though the positions for vertices can be easily found by solving a system of linear functions, the resulting drawings often have poor vertex resolution. After that, Eades proposed a more powerful method based on spring forces in a mechanical system [6]. This algorithm is simple, conceptual intuitive and it can produce graphs with uniform edge length while keeping the symmetry as much as possible. However, as repulsive forces which act on all vertices are calculated in each loop to find the minimum of the energy function, this method is computationally expensive. Different from Eades’ spring-embedder model, Kamada and Kawai introduced a new spring system in which every two vertices were connected by a spring of desirable length which was proportional to the graph theoretic distance between them in the corresponding graph [19]. Though it requires more time to calculate the shortest path between vertices, the convergence of the energy function is improved. To improve the speed and complexity of the spring-embedder model, Fruchterman and Reingold proposed a new method which followed two rules: (1) vertices connected by an edge should be drawn near each other; (2) vertices should not be drawn too close to each other [22]. This method treats vertices in the graph as atomic particles or celestial bodies, exerting attractive and repulsive forces from one another. Though the time complexity remains the same, it uses “temperature” to control the displacement of vertices so that the layout can get better. In order to make the drawing satisfy different aesthetic criteria, Davidson and Harel proposed a method to build the energy function according to vertex distribution, nearness to the borders, edge length and number of edge crossings [23]. Simulated annealing is used to minimize the energy function. In this method, users have to set the control parameter empirically and decide when to change the temperature and how to change it as well. In addition, simulated annealing has poor performance in finding the minimal energy state of the system, so, it is unfeasible for real-time graph drawing. In addition to the vertex–vertex repulsion methods described above, Lin and Yen proposed a new method based on edge–edge repulsion [24]. They use the attractive forces and the repulsive forces among edges, derived from potential fields, to adjust the position of each vertex. Although their method can overcome the problem of zero angular resolution, keep the symmetry and uniform edge length, they need to take graph layouts generated by classical force-directed algorithms as the input to produce the final graphs.

A typical force-directed graph layout algorithm is explained as follows [22]. Given a graph \( G = (V, E) \), let \( n \) be the number of vertices, \( p_1, \ldots, p_n \) be the particles in a drawing plane and \( S \) be the area of the drawing plane. The attractive and repulsive forces are denoted as

\[
f_a(d) = \frac{d^2}{k}
\]

\[
f_r(d) = -\frac{k^2}{d}
\]

where \( d \) denotes the distance between two vertices and

\[k = C \sqrt{\frac{S}{n}}\]

denotes the optimal distance between vertices. The attractive forces exist between two adjacent vertices and repulsive forces exist between all pairs of vertices. After the attractive and repulsive forces are calculated, the vertex will iteratively move in the direction of the total force until the system reaches the minimal energy state.

The main steps of the above algorithm can be summarized as follows:

1. Calculate the attractive forces between adjacent vertices.
2. Calculate the repulsive forces between all pairs of vertices.
3. Limit the total displacement by the temperature.

In each iteration, the above procedure is computationally expensive requiring \( O(|E|) \) time to compute the attractive forces and \( O(|V|^2) \) time to compute the repulsive forces [25].

In order to improve the quality of the resulting layout, many initial placement techniques have been proposed. Baur et al. describe in their literature that vertices should be iteratively added...
to either end of a linear layout to reduce the edge crossings [26]. They proposed four rules for determining the vertex insertion order and the position at which to append the next vertex, respectively. The essential object is to keep the number of open edges low, which result in crossings later on. To reduce the running time, Hadany et al. proposed a multi-scale approach to drawing graphs [27]. First, a pre-processing technique is used to obtain the coarser-scale representations of the graph. Then, the globally related properties are optimized on coarser graphs and locally related properties are optimized on finer graphs. To improve Hadany’s algorithm, Harel et al. proposed a pre-processing method to obtain the coarse graphs by shrinking nodes that were drawn close to each other into a single node [28]. In addition, Walshaw proposed a multilevel algorithm for drawing large graphs [29]. He first groups vertices to form clusters, which are used to define new graphs. This process is repeated until the graph size falls below some threshold. Then the coarsest graph is given an initial layout and the layout is successively optimized for all the graphs. To further improve the result of [29], Hu combined the multilevel approach with the octree technique [30]. However, for a graph in which vertices connect to the same central vertex, the graph coarsening process in [29,30] often assign the vertices to too many levels. In 2004, Gajer et al. proposed a method of drawing graphs with pre-placing the vertices intelligently, close to their final positions [31]. Different from [28], they add vertices to the current drawing plane one at a time. Their method not only reduces the time consumption, but also makes the drawings smoother and more symmetric. However, because there is no statistical evaluation of the aesthetic criteria, the aesthetic properties of the results are unclear. In addition, their method needs to produce a graph embedding of high dimension first because the merits of their method results from projecting the high-dimensional embedding into two or three dimensions. In 2012, Hua et al. proposed a new pre-positioning method to reduce the running time and edge crossings, similar to [31], they also placed vertices to the drawing plane one at a time [20]. Their principle is that vertices with the most edge crossings should be placed first. Although they show that their method can produce graphs with fewer edge crossings, it is only suitable for small tree graphs with 15–70 vertices.

In terms of the application of the PageRank algorithm in the graph visualization domain, Chung et al. proposed an algorithm which used PageRank vectors to determine a set of graph clusters [32]. Their goal was to capture local communities by assigning edges with weight inversely proportional to the PageRank vector. The force-directed algorithm was used to determine the coordinates for each vertex.

Our approach is different from the above algorithms in the sense that we use a variant of the PageRank algorithm to pre-calculate the positions of the vertices, which takes the vertices’ importance degree into consideration. In addition, we classify the vertices into different layers according to their importance degree and the number of connecting edges to the upper layer. Also, we provide a new method to calculate the relation matrix of the vertices and the matching coefficient is used to assign the vertices to the available positions. Our goal is to reduce the computing time and edge crossings of the resulting drawing.

3. Our approach

This section describes the details of our approach. Fig. 1 illustrates our algorithmic framework. First, we calculate the importance degree for each vertex in the Weight Calculating Model. Then, in the Center Vertices Selecting Model, we sort the center vertices by the weight scores. Next, in the Layer Vertices Selecting Model, the other vertices are classified into different layers, and their relationships are calculated in the Relation Matrix Calculating Model. At last, the final positions for the vertices are determined in the Layer Vertices Positioning Model.

3.1. Weight Calculating Model

The classical PageRank algorithm calculates the numerical weight to represent the importance degree of a vertex in directed graphs [33]. It considers that a vertex which has larger degree is more important within the vertex set. PageRank can also be applied on undirected graphs, in which case the out-degree of a vertex is equal to the in-degree, and convergence is usually achieved after fewer number of iterations [34]. For undirected graphs, if we still consider PageRank to be proportional to the degree of the vertices, it does not help in choosing more important vertices [35]. In this section, we propose a new method for our situation to calculate the weight for vertices in undirected graphs.
The classical PageRank algorithm [36] for directed graphs can be described as

\[
\text{PageRank}(p_i) = \frac{(1-d)}{N} + d \sum_{p_j \in \text{M}(p_i)} \text{PageRank}(p_j)/\text{L}(p_j)
\]

where \(d\) denotes the damping factor, \(p_1 \ldots p_N\) are the vertices of the graph, \(\text{M}(p_i)\) is the set of vertices that link to \(p_i\), \(\text{L}(p_i)\) is the number of outbounds links from \(p_i\), and \(N\) is the total number of vertices.

For undirected graphs, if we treat edges both as the incoming and outgoing links, the PageRank algorithm can be described as

\[
\text{PageRank}(p_i) = \frac{(1-d)}{N} + d \sum_{p_j \in \text{C}(p_i)} \text{PageRank}(p_j)/\text{N}(p_i)
\]

where \(\text{C}(p_i)\) is the set of vertices that connect with \(p_i\), and \(\text{N}(p_i)\) is the number of links that connect with \(p_i\). This rough description may cause troubles in the situation illustrated in Fig. 2. Although vertices \(b_1, b_2, b_3\) and \(b_4\) have the most incident edges, it is vertex \(a\) that occupies the most important position. Because the PageRank algorithm divides each vertex’s importance degree by the degree of the vertices, it will make the final importance degree of vertex \(a\) smaller than that of vertex \(b_1 \ldots b_4\). To solve this problem, our VertexRank algorithm is described by

\[
\text{VerRank}(p_i)_{\text{new}} = (1-d)\text{VerRank}(p_i)_{\text{old}} + d \sum_{p_j \in \text{C}(p_i)} \text{VerRank}(p_j)
\]

The result of these two algorithms for the graph in Fig. 2 is presented in Table 1. We can see that VertexRank outperforms PageRank in evaluating the importance degree of the vertices obviously.

### 3.2. Center Vertices Selecting Model

**Definition 1 (Center vertex).** Center vertex is the top layer vertex which has local maximal importance degree, like vertex \(a\) in Fig. 3 shows.

![Fig. 2. A typical layout of graph on which the classical PageRank algorithm cannot work well.](image)

<table>
<thead>
<tr>
<th>Vertices</th>
<th>(a)</th>
<th>(b_1 \ldots b_4)</th>
<th>(c_1 \ldots c_4)</th>
<th>(d_1 \ldots d_8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VertexRank</td>
<td>1.8543</td>
<td>1.5285</td>
<td>0.9272</td>
<td>0.6654</td>
</tr>
<tr>
<td>PageRank</td>
<td>1.3407</td>
<td>1.7892</td>
<td>1.1909</td>
<td>1.1909</td>
</tr>
</tbody>
</table>

### 3.3. Layer Vertices Selecting Model

This model is used to classify the vertices into different layers. Assume \(i \in I\) is the set of center vertices, \(C\) is the set of center vertices, \(L(i)\) represents the layer of each vertex and \(D\) represents the relation matrix of the graph, the vertices classifying procedure is illustrated in Algorithm 1.

**Algorithm 1.** Vertices classifying algorithm.

```plaintext
for each \(i\) in \(I\) do
    \(L(i) = -1\);
end for
for each \(c\) in \(C\) do
    \(L(c) = 1\);
end for
\(\text{max}L = 1\)
\(\text{finished} = 0\)
while \(\text{finished} \neq 1\) do
    for each \(i\) that \(L(i) = \text{max}L\) do
        for each edge that \(D(i,j) = 1\) and \(L(i) = -1\) do
            \(L(i) = \text{max}L + 1\);
        end for
    end for
    \(\text{max}L = \text{max}L + 1\)
end for
if \(\forall L(i) \in L, L(i) \neq -1\) then
    \(\text{finished} = 1\);
end if
end while
```
We classify the vertices into different layers according to their shortest path to the center vertices. Compared with the method of Bellman–Ford and Dijkstra, whose distance between two vertices is flexible and time complexity is \(O(|E||V|)\) and \(O(|V|^2)\), respectively [37], in Algorithm 1, we assign the distance between two connected vertices as 1, and the time complexity is \(O(|E|)\), where \(|V|\) is the number of vertices and \(|E|\) is the number of edges.

3.4. Relation Matrix Calculating Model

Different from the traditional relation matrix which uses \(\{0, 1\}\) to represent if two vertices are connected, we adopt a new method to calculate the relation matrix. Traditionally speaking, attractive forces in the force-directed system only exist between vertices which are connected directly, such as vertices connecting to the same vertex, as Fig. 4 shows.

Fig. 4(a) shows the ideal position of a graph. If we use the traditional relation matrix to calculate the position of each vertex, it may produce graph as Fig. 4(b) shows, in which \(p_1\) and \(p_2\) are far apart. This is because it neglects the possible attractive force between \(p_1\) and \(p_2\). In our relation matrix, we consider that these two vertices are connected indirectly. Different from the traditional relation matrix which will assign 0 to represent \(p_1\) and \(p_2\) is not connected, we assign 0.5 in this situation to indicate that these two vertices are connected indirectly, as Fig. 4(c) shows.

Our new relation matrix is used in the force-directed algorithm in the Layer Vertices Positioning Model because it can produce graphs more similar to the ideal one. In this way, it reduces the computing time. Detailed information of this procedure is described in Algorithm 2.

Algorithm 2. Relation matrix calculating algorithm.

Define the Layer set as \(L = L_1, L_2, \ldots, L_N\);
\(L_i = v_1, v_2, \ldots, v_m\) represents the vertex set on the \(i\)th Layer;
\(N\) represents the size of the Layer set \(L\);
\(R\) represents the relation matrix of the \(i\)th layer;
\(D\) represents the original relation matrix of the graph;

for \(i = N - 1\) do

for each vertex \(v_k\) in \(L_i\) do

the point relation set of \(v_k\) \(pr_k = \{v_k\}\);

end for

define \(R(\cdot, \cdot) = \alpha R(\cdot, \cdot) + \beta R_i(\cdot, \cdot)\)

where \(pr_k = [av_k, bv_k]\)

if \(i = N\) then

the relation matrix of \(L_N\) \(R_N = D\);

for each edge \(e\) connecting the vertices \(v_k, v_j\) in \(L_N\) do

\(pr_k = pr_k \cup 0.5v_j, pr_j = pr_j \cup 0.5v_k\);

end for

else

\(R_i = R_{i+1}\);

for each edge \(e\) connecting the vertices \(v_k \in L_i, v_j \in L_{i+1}\) do

\(pr_k = pr_k \cup 0.5pr_j\);

end for

end for

From Algorithm 2, assume that \(|E|\) is the number of edges and \(|V|\) is the number of vertices, the time complexity for this procedure is \(O(|E||V|)\).

3.5. Layer Vertices Positioning Model

According to the connection with the upper layer vertices, the other vertices are classified into different layers. Fig. 5 shows the four major steps of the Layer Vertices Positioning Model. First, the vertices in the same layer are subdivided in the vertices selecting procedure. Next, the position calculating procedure offers the available positions of these vertices. Then, the available positions are assigned to the layer vertices in the position assigning procedure. At last, the force-directed algorithm is used to determine the final position.

3.5.1. Vertices Selecting Procedure

The Vertices Selecting Procedure is used to subdivide the vertices of the same layer into different groups. Assume that \(I\) is the set of layer vertices, \(N\) is the set of vertices that have already been assigned positions, and \(C(i)\) is the set of vertices in \(N\) which connect to vertex \(i\). Our vertices selecting procedure is described in Algorithm 3.


sort \(I\) by VertexRank value in descending order;
for each \(i\) in \(I\) do

select the layer vertices \(g_1, g_2, \ldots, g_n \in I\) which makes \(C(g_1) = C(g_2) = \ldots = C(g_n) = C(i)\); output the vertex sequence \(i, g_1, g_2, \ldots, g_n\) denoted as \(G_n\) to Group \(G\);
delete \(i, g_1, g_2, \ldots, g_n\) from \(I\);
end for

3.5.2. Position Calculating Procedure

In this procedure, we calculate the available positions for the vertices selected in the Vertices Selecting Model. As described in
Algorithm 3. \( N \) is the set of vertices that have already been assigned positions, \( C(i) \) is the set of vertices in \( N \) which connect to vertex \( i \). The detail of our Position Calculating Procedure is sketched in Algorithm 4.


for each \( G_i \) in \( G \) do
  find the connection set \( C \) of Group \( G_i \), \( C = C(g_i), g_i \in G \);  
  compute the set \( Q = N \) which makes \( C(q_i) \subseteq C, q_i \in Q \);  
  calculate the size of \( C, G \) and \( Q \), which is denoted as \( S_x, S_y \) and \( S_q \);  
  if \( S_x = 1 \) then
    find the largest angle \( \theta \) formed by edge \( c(q_i), c_i \in C, q_i \in Q \);  
    calculate the radius \( r = \max \{ c(q_i), 1 \} \), \( c_i \in C, q_i \in Q \);  
    the available position set \( P \) consists of the positions that divide the longest arc into \( S_x + S_y \) equal parts, each angle \( \theta_i = \theta / (S_x + 1) \), as Fig. 6(a) shows;  
  end if  
  if \( S_x > 1, S_y = 0 \) then
    compute the coordinates \( (X_o, Y_o) \) of the center \( O \), \( X_o = \text{mean}(X_c), Y_o = \text{mean}(Y_c), c \in C \);  
    calculate the radius of the circle \( r = \max \{ o(c), 1 \} \), \( o \) is the center of the circle and \( c \);  
    the available position set \( P \) consists of the positions that divide the whole circle into \( S_x + S_q \) equal parts, each angle \( \theta_i = 2\pi / (S_x + S_q) \), as Fig. 6(b) shows;  
    for each \( c \) in \( C \) do
      find the closest position \( p, p \in P \);  
      delete \( p \) from \( P \);  
    end for  
  end if  
  if \( S_y > 1, S_x = 0 \) then
    compute the coordinates \( (X_o, Y_o) \) of the center \( O \), \( X_o = \text{mean}(X_c), Y_o = \text{mean}(Y_c), c \in C \);  
    calculate the radius of the circle \( r = \max \{ o(q_i), 1 \} \), \( o \) is the center of the circle and \( q_i \);  
    find the largest angle \( \theta \) formed by edge \( o(q_i), q_i \in Q \);  
    the available position set \( P \) consists of the positions that divide the arc into \( S_x + S_y \) equal parts, each angle \( \theta_i = \theta / (S_x + S_y) \), as Fig. 6(c) shows;  
    for each \( q \in Q \) do
      find the closest position \( p, p \in P \);  
      delete \( p \) from \( P \);  
    end for  
end if

From Algorithm 4, assume that \( |E| \) is the number of edges and \( |V| \) is the number of vertices, the time complexity for this procedure is \( O(|E| \log |E|/|V|) \).

3.5.3. Position Assigning Procedure

This subsection describes how to match a vertex with an available position. In our approach, we call the Euler distance between two vertices graph distance. Our position assigning procedure follows two rules:

1. Vertex with the largest VertexRank value should be placed first.
2. The Euler distance should match the relation distance as much as possible.

Assume that \( D_{ij} \) is the Euler distance matrix between \( P \) and \( N, D_{crank} \) is the relation matrix between \( G \) and \( N, D_{vertexrank} \) is the VertexRank matrix for \( G \). The matching coefficient \( R \) is calculated by

\[
R = D_{vertexrank} / D_{crank} \times D_{crank}^{-1}
\]

The bigger the \( R \) is, the closer the connection between a position and a vertex is. So if

\[
R_{\text{max}}(i,j) = R(i,j)
\]

we call the vertex \( i \) and the position \( j \) is matched. In this way, we find the positions for all the layer vertices.

Assume that \( |E| \) is the number of edges and \( |V| \) is the number of vertices, the time complexity for this procedure is \( O(|E||V|) \).

4. Experimental results

This section contains several examples that demonstrate the advantages of our pre-positioning method. Experiments are carried out using Windows 7 on an Intel Core i5-2400 at 3.10 GHz and 2.00G RAM. We use the source code provided by the University of Chicago\(^2\) to present the results of the classical method [22]. Our purpose is to produce layout of graphs with less edge crossings in short time. In our experiment, first, we will give some examples to show the final layout of our method against the classical method. After that, we will compare the statistics between these two experimental results.

4.1. Test data

The testing graphs are generated from the “Rome Graphs”\(^3\),\(^4\) the relationships between commodities on the social network\(^4\) and the dataset provided by [38]. The “Rome Graphs” is intended to provide benchmark data for various classes of graph drawing algorithms. The size of the graphs ranges between 10 vertices

---


\(^3\) More information can be found on [http://www.graphdrawing.org/data](http://www.graphdrawing.org/data).

\(^4\) More information can be found on [https://github.com/TBEDP/dataavjs](https://github.com/TBEDP/dataavjs).
and 100 vertices. The relationships between different entities are stored in the graphml format files.

4.2. Results

Some of the experimental results for small graphs using the classical method [22] and our method are shown in Table 2, and their corresponding statistical results are shown in Table 3. We use the normalized standard deviation of edge lengths (NormStdLen), angular resolution (AngResl), average of the angular resolution (AvgAngResl), running time, number of iterations (# of Iterations) and the number of edge crossings (# of Crossings) as the evaluation criteria.

Figure (a)–(g) in Table 2 are the results on “Rome Graphs” and Figure (h) is the result on the commodity relationships. The initial position of each vertex is randomly generated. Both the classical

<table>
<thead>
<tr>
<th>Graph</th>
<th>Classical</th>
<th>Our</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>(h)</td>
<td>31</td>
<td></td>
</tr>
</tbody>
</table>

![Fig. 7. # of edge crossings between our method and the classical method.](image)

Table 2

<table>
<thead>
<tr>
<th>Graph</th>
<th>Classical</th>
<th>Our</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>0.21</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>0.27</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td>0.30</td>
<td></td>
</tr>
<tr>
<td>(h)</td>
<td>0.31</td>
<td></td>
</tr>
</tbody>
</table>

Table 3

<table>
<thead>
<tr>
<th>Graph</th>
<th>Classical</th>
<th>Our</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>11.84</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>1.75</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td>4.27</td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td>13.45</td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td>3.51</td>
<td></td>
</tr>
<tr>
<td>(h)</td>
<td>9.82</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph</th>
<th>Classical</th>
<th>Our</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>77.56</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>107.59</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>96.10</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>109.45</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td>86.31</td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td>106.81</td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td>103.30</td>
<td></td>
</tr>
<tr>
<td>(h)</td>
<td>94.72</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph</th>
<th>Classical</th>
<th>Our</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>45.52</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>50.29</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>66.09</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>31.92</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td>48.35</td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td>103.76</td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td>234.37</td>
<td></td>
</tr>
<tr>
<td>(h)</td>
<td>77.54</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph</th>
<th>Classical</th>
<th>Our</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>(h)</td>
<td>47</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph</th>
<th>Classical</th>
<th>Our</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>(b)</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>(c)</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>(d)</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>(e)</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>(f)</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>(g)</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>(h)</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
method and our method use the same initial drawing as the input. From Table 2, we can see that our method can produce comprehensible layout of graphs with less edge crossings, especially in Figure (a), (d), (e), (g) and (h). In addition, our method can magnify the angular resolution while the classical method may not, especially in Figure (b)–(d) and (g) (the statistics can be found in Table 3). Note that in Figure (a), although the classical method has larger angular resolution, our method has larger average angular resolution while reducing the edge crossings.

From Table 3, we can see that our method can produce drawings with more uniform edge length than the classical method, except for Figure (b) and (c) (see NormStdLen in the table). While the final drawings produced by the classical method may have small angular resolution, our method can improve the angular resolution better, except for Figure (a) (see AngRes in the table). Note that in Figure (a), although the classical method has larger angular resolution, its average angular resolution is smaller than our method (see AvgAngRes in the table). Observing # of

Table 4
Experimental results of our method and the classical method.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Classical</th>
<th>Our</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td><img src="image" alt="Classical Drawing" /></td>
<td><img src="image" alt="Our Drawing" /></td>
</tr>
<tr>
<td>(b)</td>
<td><img src="image" alt="Classical Drawing" /></td>
<td><img src="image" alt="Our Drawing" /></td>
</tr>
<tr>
<td>(c)</td>
<td><img src="image" alt="Classical Drawing" /></td>
<td><img src="image" alt="Our Drawing" /></td>
</tr>
<tr>
<td>(d)</td>
<td><img src="image" alt="Classical Drawing" /></td>
<td><img src="image" alt="Our Drawing" /></td>
</tr>
</tbody>
</table>
Crossings, the number of the edge crossings produced by our method is less than the classical method. Also, the time consumption and the iteration number of our method are less than the classical method, except for Figure (f) (see *Running Time* in the table).

To further demonstrate the fluctuation of the number of edge crossings, we ran the classical method and our method in Figure (h) both 500 times. The distribution of the number of edge crossings with the number of experiments is shown in Fig. 7. In each experiment, the initial positions of vertices are randomly generated in order to eliminate human intervention. Compared to conduct the experiment with only one same randomly generated initial layout, we can get rid of the situation that is always favorable or unfavorable for the classical method. The time of the experiment 500 is chosen to make the final result representative. From Fig. 7, we can see that the number of edge crossings by the classical method varies in a wide range between 0 and 20, while ours only between 0 and 6. It also shows that the classical method’s number of edge crossings is more often between 4 and 8, while ours between 0 and 2. In this way, our method is more stable in producing graphs with less edge crossings than the classical force-directed method.

For the visualization of large graphs, the experimental results produced by the classical method [29] and our method are shown in Table 4.

From Table 4, we can see that our method can distribute “the branches” of the graphs more evenly. In addition, from Figure (a) to (d), we can find that our method can display the cluster of vertices more rounded. These advances make our results more appealing than the classical method. From Table 5, we can see that the running time of our method is shorter than the classical method. Note that the time consumption for both our method and the classical method here is longer than that described in [29]. This is because we use Matlab to build the algorithm while [29] uses C++ and C++ averages a processing speed that is over 500 times faster than Matlab code [39].

### 5. Conclusions and future work

In this paper, we have presented an advanced pre-positioning approach for the force-directed graph layout algorithm, and it can be used by visual analysis tools to visualize graphs. First, we use a variant of the PageRank algorithm to pre-calculate the position for each vertex, then, the classical force-directed graph layout algorithm is used to produce the final layout. The experimental results have proved that our approach can reduce the computing time as well as the number of edge crossings. In addition, our method can produce graphs with a stable number of edge crossings. In the future, we will carry out experiments on more larger graphs. In addition, a rigorous justification will be provided to support the use of the weight assignments calculated by Vertex-Rank, together with the pre-positioning model, allows a faster convergence in the use of a force-directed approach. In order to further reduce the computing time, we will use the pruning algorithm for reference to cut off the unnecessary relationships between vertices in advance.

### Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61163059 as well as the National High Technology Research and Development Program of China under Grant 2012AA010105. The authors would like to thank the reviewers for their helpful comments and constructive suggestions.

### References