Vertex-Neighboring Multilevel Force-Directed Graph Drawing

Farshad Ghassemi Toosi
Department of CSIS
University of Limerick
Email: farshad.toosi@ul.ie

Nikola S. Nikolov
Department of CSIS
University of Limerick
Email: nikola.nikolov@ul.ie

***Abstract*—We introduce a new force-directed graph drawing algorithm for large undirected graphs with at least a few hundreds of vertices. Our algorithm falls into the class of multilevel force-directed graph drawing algorithms. Unlike other multilevel algorithms it has no pre-processing step and it also ignores repulsion forces between pairs of non-adjacent vertices. As a result, our algorithm demonstrably outperforms known multilevel algorithms in terms of running time while keeping the quality of the layout sufficiently good.**

***Index Terms*—Big Graph Visualization, Multilevel Force-Directed.**

I. Introduction

The class of force-directed graph drawing algorithms is one of the most successful approaches to find aesthetically pleasing layouts of general graphs in 2D [3], [10]. The simplicity of this class of algorithms along with reasonable running time for average-sized graphs and the quality of the layout (by placing highly interconnected vertices close to each other and avoiding unnecessarily long edges) make this class of algorithms one of the most popular and successful approaches. Over the last decade, a group of force-directed algorithms which exploit the multilevel paradigm has been introduced.

Classical force-directed algorithms such as Fruchterman-Reingold [4] and Kamada-Kawai [9] are relatively slow with a high chance of getting trapped into a local minimum (in terms of number of edge crossings) for large graphs. It has been empirically demonstrated that the number of edge crossings in a layout is the feature of the layout that most closely correlates to its quality for visual analysis [11]. The more recent multilevel force-directed algorithms improve both the quality of the layout in terms of edge crossings and the running time [2]. Multilevel methods involve using force-directed algorithms over a number of phases in order to reduce the chance of being trapped in a local minimum. These algorithms are not dealing with all vertices at once. At the very first level it starts with one or a few *supervertices* each representing a group of vertices that they totally contain all the vertices in the graph. A several force-directed iterations is applied on the current supervertices in order to refine their positions in the layout and makes the layout ready to progress to the next level. In the next level the supervertices are broken down into smaller supervertices in order to get refined by a several force-directed iterations again. This process continues until supervertices cannot be broken down further and each supervertex is an actual vertex of the input graph.

There have been several multilevel methods proposed so far which differ either in their coarsening method (which is the method for grouping vertices into supervertices), or in the applied force-directed algorithm, or in how to physically break a supervertex into smaller supervertices. For example, the algorithm proposed by Pawel *et al.* [5] has a different coarsening method compared to the algorithm of Yifan [8]. Two main features of the class of multilevel algorithms are *1)* the very short running time and *2)* the reduction in the number of edge crossings.

The layout of this paper is as follows: In Section 2 we review work related to the class of multilevel graph drawings. In Section 3 we introduce our algorithm. In Section 4 we report some layouts produces with our algorithm along with the running time. Finally, in Section 5 we draw some conclusions of this work and discuss future works.

II. Related Work

The class of multilevel force-directed algorithms has gained popularity mostly because of improving the quality of the layout (compared to classical force-directed algorithms) as well as the running time [2]. In 2005 Yifan [8] proposed an algorithm that exploits a multilevel approach with an *octree* technique proposed by Barnes and Hut [1]. The coarsening method used in this algorithm is based on edge collapsing (EC) where pairs of adjacent vertices are selected to be collapsed and create a *supervertex*. This algorithm [8] ignores long forces by applying the force calculation only within a neighborhood. Walshaw [13] proposed a multilevel algorithm for graph drawing in 2002. The force-directed algorithm which has been used in this algorithm is inspired by the force-directed algorithm proposed by Fruchterman and Reingold [4].

One of the biggest challenges in large graph visualisation is how to deal with *vertex overlapping* that decreases the expressiveness of the layout. Yifan proposed a solution [6], however his algorithm requires vertex repositioning. Both algorithms by Yifan [6], [8] are implemented in **Graphviz** [7].

978-1-5090-1897-0/16/$31.00 ©2016 IEEE

III. The Algorithm

A. Overview

The input for our proposed algorithm is a large, undirected and connected graph G with n vertices and m edges. This algorithm has two main parts: *1) vertex placement* and *2) force-directed layout*, and unlike many other multilevel algorithms, it has no coarsening part. At the beginning all the vertices are labeled as unvisited vertices. An initial pair of adjacent vertices with relatively low degree is selected and they become visited vertices. They are placed relatively close to each other, proportional to the size of the graph; this represents the first level l_0. To start building level l_{r+1}, first all unvisited vertices which are adjacent to the visited vertices are selected and get placements according to the *vertex placement* method (it will be discussed later in this section) and then our proposed *force-directed* method is applied over several iterations on the visited vertices. A cooling schedule is then used in order to reduce unnecessary movements.

Once the total energy of the system goes below a threshold the force-directed stops and level l_{r+1} is complete. This process continues until all the vertices of the graph become visited. In the remainder of the paper we use the notation *Un-Ad* for *unvisited adjacent*.

B. Vertex Placement

Let $v(x)$ and $v(y)$ be the x and y coordinates of vertex v, respectively. At the end of each level (phase) the algorithm finds the *Un-Ad* vertices and places them close to the currently visited vertices. Suppose that after completing level l_r a set of *Un-Ad* vertices $\{a_1, a_2, ..., a_j\}$ for the currently visited vertex v_i is found . The placement for each vertex in the *Un-Ad* set is computed as follows:

Algorithm 1 Placement Algorithm

Vertex placement of the v_i's *Un-Ad* vertices.
Initialize a circle with a short *radius* around v_i;
radius $= \frac{10}{\mathbf{n}}$
for $p = 1$ to **j do**
$a_p(x) = v_i(x) + \cos\frac{(2\times\pi\times p)}{j} \times (\mathbf{radius})$
$a_p(y) = v_i(y) + \sin\frac{(2\times\pi\times p)}{j} \times (\mathbf{radius})$
end for

This algorithm places the Un-Ad vertices for a visited vertex v_i on a circle with a relatively small radius centred at v_i. Furthermore, the Un-Ad vertices are evenly spaced on the circle as shown in Figure 1(a).

The vertices in the newly *Un-Ad* set then become visited vertices and will be taken into account by the following force-directed iterations to improve and refine their initial placement.

C. Force-Directed

To start the algorithm an adjacent pair of vertices, usually with a lower degree, is selected and are initiated with a small length between them proportional to the size of the graph; this represents the first level l_0. It continues adding *Un-Ad* set of each visited vertex at the start of any new level and applying forces on all visited vertices. We exploit the general idea of a force-directed algorithm. The key point of our algorithm which makes it different from other force-directed algorithms is applying forces only between adjacent pairs, thus improving the running time while keeping the quality fairly good. Our algorithm exploits the general forces introduced by Fruchterman and Reingold [4]. Two forces (*attraction* and *repulsion*) are applied between any adjacent pair of visited vertices. The combination of two opposite forces is then restricted from causing huge vertex movement. To control the negative effect of big forces we make use of the diagonal of the rectangle that surrounds the layout at the time see Figure1(b).

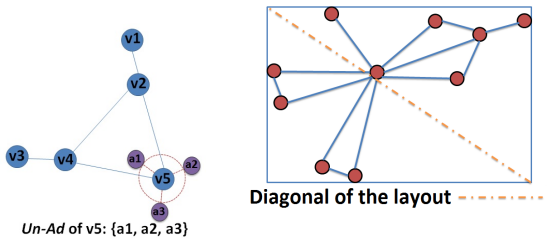


Fig. 1. 1(a) shows the placement method for the set of *Un-Ad* vertices adjacent to *v5* and 1(b) shows the diagonal of a rectangle which surrounds a layout.

At level l_r, forces are applied to the set of visited vertices in a number of iterations until the total energy of the system goes below a certain threshold. (the threshold will be discussed later). The force-directed algorithm that we applied is shown in Algorithm 2.

Algorithm 2 represents the overall picture of our work. Five parameters **diagonal**, **k**, **cooling**, **R-factor** and **A-factor** have to be updated at each iteration of the force-directed algorithm. Once the main loop in the algorithm is finished, **diagonal** is updated as it is shown in Figure 1(b). The parameters **R-factor** and **A-factor** are controlling the **diagonal** from being too small. The other variable that needs to be updated is **temperature**. It controls the rate of the movement of vertices. At the beginning of each level there are new vertices which are just added to the graph based on algorithm 1 and they need to find their correct position over some force-directed iterations. The **temperature** of the system at the start point of each level is initiated at 0.1. The **temperature** at the beginning allows large movements but as the force-directed iterations progresses, the temperature has to be reduced in order to restrict the large movements (i.e. a process of simulated annealing). To control the **temperature** a cooling schema is exploited [8]. The **temperature** is being reduced by **cooling**. The **cooling** factor in our proposed algorithm is 0.99. The **temperature** again is set at 0.1 at the start point of the next level l_r and start getting cooler by **cooling** factor.

Let all visited vertices at level l_r be the current *population* of vertices. The value of **k** is updated at each iteration and it is dependent on the current population of the graph at the time. As it is mentioned earlier, the algorithm starts with a

Algorithm 2 Adopted force-directed at iteration **t**

Force-directed on visited adjacent pairs.
Update **diagonal** of the layout from previous level.
R-factor: $\sqrt{|N|}$
A-factor: **R-factor** $\times 5$
if diagonal < 0.1 **then**
  R-factor: =*R-factor:* $\times 0.1$
  A-factor: =*A-factor:* $\times 0.1$
end if
Update **cooling**.
Update **k**.
Update **temperature**.
function: $f_r(d) = \frac{\mathbf{k}^2}{-d}$ *Calculate attraction force*
function: $f_a(d) = \frac{d^2}{\mathbf{k}}$ *Calculate repulsion force*
for each $e \in E$ **do**
  if $e.v \in visitedvertices$ & $e.u \in visitedvertices$ **then**
    $d = \|e.v.pos - e.u.pos\|$
    $force(e.v) = f_a(d) + f_r(d)$
    $force(e.u) = f_a(d) + f_r(d)$
    $\cos(\theta) = \frac{(e.u_x - e.v_x)}{d}$
    $\sin(\theta) = \frac{(e.u_y - e.v_y)}{d}$
    if $force(e.v) > 0$ & $force(e.v) > \frac{\textbf{diagonal}}{\textit{R-factor}}$ **then**
      $force(e.v) = \frac{\textbf{diagonal}}{\textit{R-factor}}$
    else if $force(e.v) < 0$ & $\|force(e.v)\| > \frac{\textbf{diagonal}}{\textit{A-factor}}$ **then**
      $force(e.v) = \frac{\textbf{-diagonal}}{\textit{A-factor}}$
    end if
    if $force(e.u) > 0$ & $force(e.u) > \frac{\textbf{diagonal}}{\textit{R-factor}}$ **then**
      $force(e.u) = \frac{\textbf{diagonal}}{\textit{R-factor}}$
    else if $force(e.u) < 0$ & $\|force(e.u)\| > \frac{\textbf{diagonal}}{\textit{A-factor}}$ **then**
      $force(e.u) = \frac{\textbf{-diagonal}}{\textit{A-factor}}$
    end if
    $e.v_t(x) = e.v_{t-1}(x) + force(e.v) \times \cos(\theta) \times$ **temperature**
    $e.v_t(y) = e.v_{t-1}(y) + force(e.v) \times \sin(\theta) \times$ **temperature**
    $e.u_t(x) = e.u_{t-1}(x) + force(e.u) \times (-\cos(\theta)) \times$ **temperature**
    $e.u_t(y) = e.u_{t-1}(y) + force(e.u) \times (-\sin(\theta)) \times$ **temperature**
  end if
end for

pair of adjacent vertices which means the size of population of the graph at the very first step is 2; at each level a new set of vertices (*Un-Ad*) is added into the population.

The value of k is $\frac{\alpha}{population}$, where α starts at 0.5 and gets reduced at each iteration by the cooling factor. To progress from level l_r to the next level l_{r+1} the total movement of the vertices at each iteration is evaluated and compared with the total movement at the previous level. If the difference between the total movement of the system for levels l_r and l_{r+1} is less than **k** then the algorithm progresses to the next level and if all the vertices are visited in l_{r+1} then the algorithm ends.

TABLE I
RUNNING TIME FOR MESH-LIKE GRAPHS.

	Graph	$\|N\|$	$\|E\|$	Run-Time(s) Our algorithm	Run-Time(s) sfdp
a	Mesh 100 X 150	15000	29750	7.2	10.2
b	Mesh 100 X 100	10000	19800	4.84	7.79
c	Mesh 60 X 100	6000	11840	1.62	4.7
d	Mesh 70 X 70	4900	9660	1.5	3.1
e	Mesh 50 X 50	2500	4900	0.436	2.13

Algorithm 3 is the skeleton of our proposed graph drawing algorithm.

Algorithm 3 Our proposed Multilevel Algorithm

Initiate an adjacent pair of vertices.
Initiate **population** to 2.
Initiate **k**, **cooling** and **temperature**.
while ((**population** $\neq n$) & (**energy** $>$ **k**)) **do**
  Update **diagonal**
  Perform Algorithm 2
  if (**energy** $\leq$ **k**) **then**
    Perform Algorithm 1
  end if
end while

IV. EXPERIMENTAL RESULTS

The described algorithm 3 has been implemented as a Java application, and the running time has been measured on a system with Intel(R) Core(TM) i5-3470 CPU with 3.20 GHz. After some initial experiments, we discovered that on average our algorithm performs best with graphs of more than 300 vertices. Therefore, a graph in the dataset, we used in our final experiments, typically has more than 300 vertices. We compare our results to the results of the **sfdp** algorithm proposed by Yifan [8] as implemented in GraphViz [7] and run on the same system as mentioned above. The running time that we measured for both our algorithm and **sfdp** includes the time for the reading of the graph along with the actual algorithm. Our dataset includes some well-known graphs provided by **University of Florida** [12] and some other symmetrical graphs.

Our first experiment is with mesh graphs with 2500 vertices ($50X50$), 4900 vertices ($70X70$), 6000 vertices ($60X100$), 10000 vertices ($100X100$), and 15000 vertices ($100X150$), respectively. Table I shows the running time for each graph for both our proposed algorithm and **sfdp**. Figure 2 shows the layouts of the mesh graphs produced by our algorithm.

Figure 2 shows the borders and angles of the graphs according to the topology of the graph. For example Figure 2(b) shows the graph with 10000 vertices which is a mesh with 100 rows and 100 columns. Our algorithm clearly shows that the graph is a mesh with almost straight lines on the borders and minimum vertex overlapping around the borders. In Figure 3 the layouts 3(c) and 3(b) are drawn by **sfdp** using an overlapping removal algorithm and without an overlapping removal algorithm respectively.

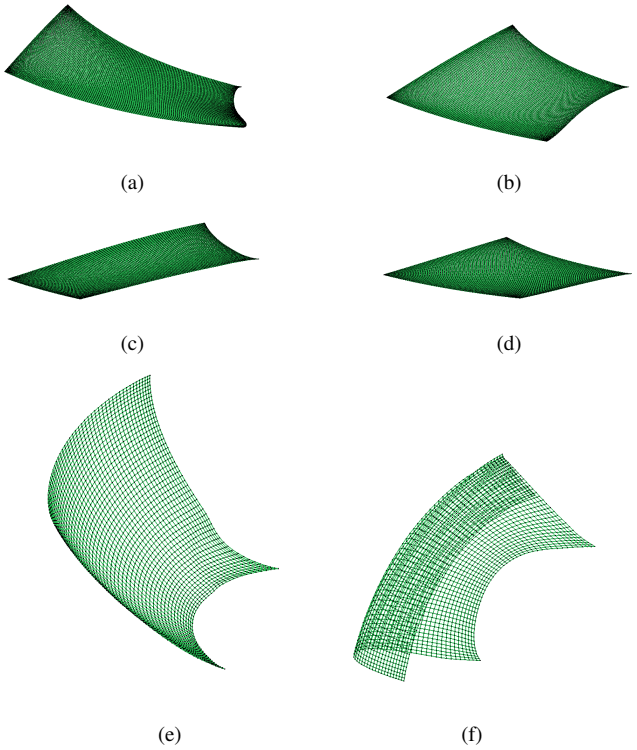


Fig. 2. Five different mesh graphs. Figure 2(a) is a mesh graph with 15000 vertices 100X150; Figure 2(b) is a mesh graph with 10000 vertices 100X100; Figure 2(c) is a mesh graph with 6000 vertices 60X100; Figure 2(d) is a mesh graph with 4900 vertices 70X70; Figure 2(e) is a mesh graph with 2500 vertices 50X50; Figure 2(f) is a mesh graph with 2500 vertices 50X50 with different start point from 2(e)

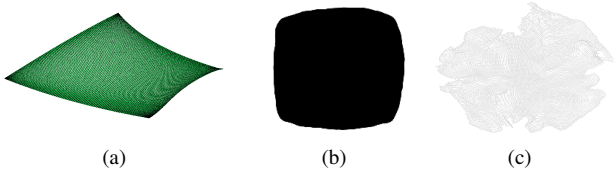


Fig. 3. Three different layouts for 100X100 mesh, 3(a) is drawn by our proposed algorithm, 3(b) is drawn by **fsdp** and 3(c) is drawn by**fsdp** using removal overlapping algorithm.

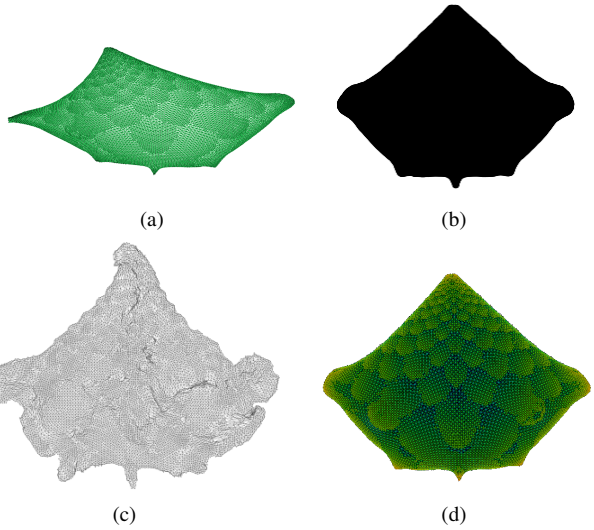


Fig. 4. Four different layouts for the graph called **crack** from [12], 4(a) is drawn by our proposed algorithm, 4(b) is drawn by **fsdp**, 4(c) is drawn by**fsdp** using removal overlapping algorithm and 4(d) is what the graph should look like.

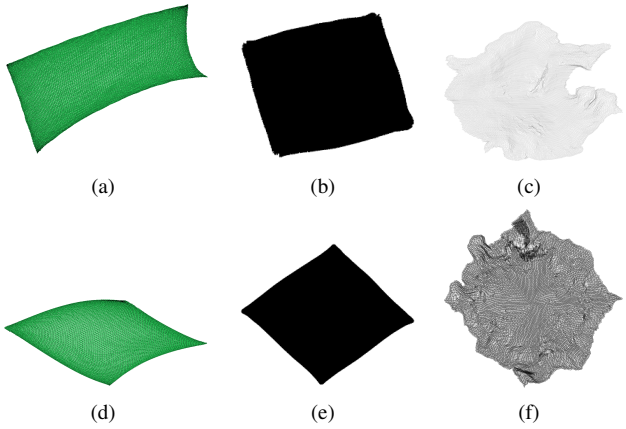


Fig. 5. Two selected graphs [12] each with three different layouts, the layouts in 5(a), 5(b) and 5(c) are drawn by our algorithm, **fsdp** and **fsdp** using removal algorithm respectively and it is called **epb1**, and the layouts in 5(d), 5(e) and 5(f) are drawn by our algorithm, **fsdp** and **fsdp** using removal algorithm respectively and it is called **stokes64**.

The *sfdp* layout without overlapping removal algorithm results in a high rate of vertex overlapping (see Figure 3(b)) and remains messy after applying an overlapping removal algorithm (see Figure 3(c)). The layout produced by our algorithm in Figure 3(a) results in low vertex overlapping rate and no overlapping removal is necessary.

The two layouts in Figure 2(e) and Figure 2(f) are for the same graph (**e** in Table I), and the difference is due to the different selection of the starting vertices for drawing. Figure 2(f) shows a very smooth folding although it still implies the structure of the graph.

In another experiment a collection of selected graphs from [12] is represented. We have selected graphs of different types and different sizes. Table II shows the details of the graphs and Figures 4 to 10 show the corresponding layouts.

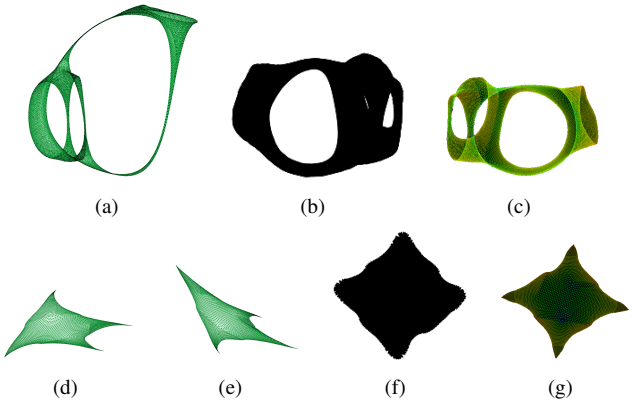
Figure 4 shows three different layouts for a selected graph called *Crack*. Figure 4(d) shows the real structure of the graph. The Figure 4(b) and 4(c) show the layouts by **fsdp** without and with removal overlapping, respectively, while Figure 4(a) shows the layout by our proposed algorithm.

Figure 5 shows three different layouts for two selected graphs [12]. Figure 5(a) and 5(d) show the layouts by our proposed algorithm. The Figures 5(b) and 5(e) show the layouts by **fsdp** and Figure 5(c) and 5(f) show layouts by **fsdp** using removal overlapping respectively. As it is shown in the Figure 5 our proposed algorithm shows the structure of the graph, which has rectangular shape, more clear than the one by **fsdp**.

Figure 6 shows two layouts of the famous graph **dimacs10-data** [12] by our algorithm 6(a) and **fsdp** 6(b). The layout by **fsdp** probably emphasizes the structure of the graph better however the running time by our algorithm is less than half,

(a) (b)

Fig. 6. Two different layouts for the graph called **Dimacs10-Data** from [12], 6(a) is drawn by our proposed algorithm and 6(b) is drawn by **fsdp**.



(a) (b) (c) (d) (e) (f) (g)

Fig. 7. Three different layouts for two graphs called **fe-4elt** and **swang1** from [12], 7(a) and 7(d) and 7(e) are drawn by our proposed algorithm, 7(b) and 7(f) are drawn by **fsdp** and 7(c) and 7(g) are the real structure of the graphs [12].

see Table II.

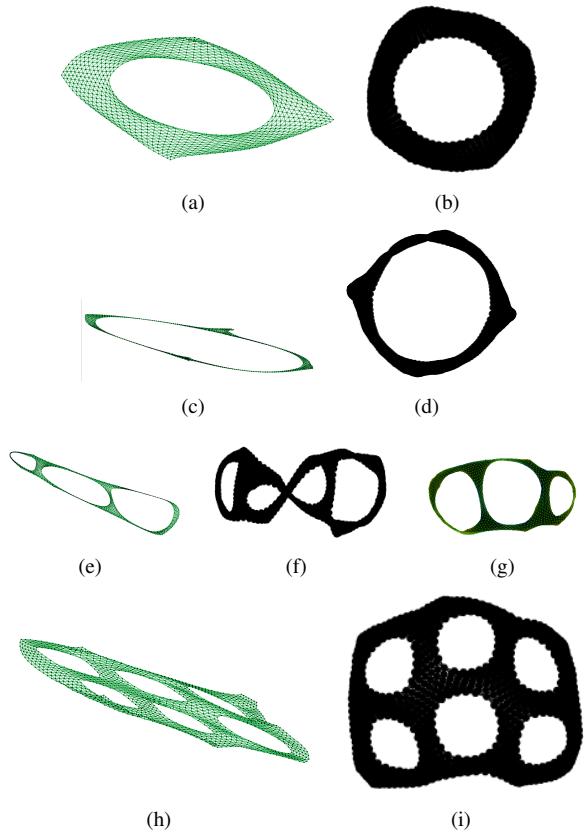
Figure 7 shows different layouts of the two selected graph [12]. Figures 7(a), 7(d) and 7(e) show the layout by our proposed algorithm. The difference in the layouts in Figures 7(d) and 7(e) is due to the different selection of the starting vertices for drawing. Figures 7(b) and 7(f) show the layout by **fsdp** and 7(c) and Figure 7(g) show the real structure of the graph [12]. As you can see in Figures 7(a) and 7(d) the details around the bother are well distinguished according with the real structure of the graph. The running times are almost half of the running time by **fsdp**.

Four graphs 8(a), 8(c), 8(e) and 8(h) [12] are shown in Figure 8 with their details and running time in Table II.

A few more experimental results which confirm the above observations are presented in Figures 10 and 9 with their details in Table II.

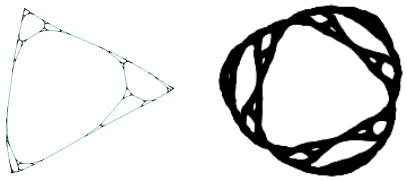
V. Conclusion

We have proposed an early stage algorithm to draw large graphs using a multilevel paradigm with $O(E)$ complexity per iteration, while E is the number of edges between visited vertices. Our algorithm does not use any coarsening step but starts with a selected vertices and at each level adds the neighbors of the currently visited vertices. The forces between all non-adjacent vertices are ignored in this algorithm which makes the running time faster than in similar algorithms, while the quality of the layout is kept at a fairly acceptable stage. The exploited force-directed algorithm in our algorithm has been



(a) (b) (c) (d) (e) (f) (g) (h) (i)

Fig. 8. Four different graphs each with two layouts (our proposed algorithm: 8(a), 8(c), 8(e), 8(h) and **fsdp**: 8(b), 8(d), 8(f), 8(i)). 8(a) and 8(b) jagmesh1, 8(c) and 8(d) jagmesh4, 8(e) and 8(f) jagmesh5 and 8(h) and 8(i) jagmesh8. 8(g) is the real structure layout of the graph jagmesh7.



(a) (b)

Fig. 9. Two different layouts for the graph called **sierpinski**, 9(a) is drawn by our proposed algorithm and 9(b) is drawn by **fsdp**.

taken from the one proposed by Fruchterman and Reingold [4]. Overall, our algorithm has a better running time and has a better layout quality for some graphs, e.g. the mesh graphs in Figures I and 9 show that our algorithm takes a good care of borders to represent them as straight lines, unlike other algorithms which have approximate estimation.

VI. Future Work

The visual results of our algorithm imply a 3D layout with physical orientation. This is due to the selection of a start vertices. As a future work, a more efficient way to select a more meaningful start point can be considered. Alternatively, some transformation methods on the layout can be applied in

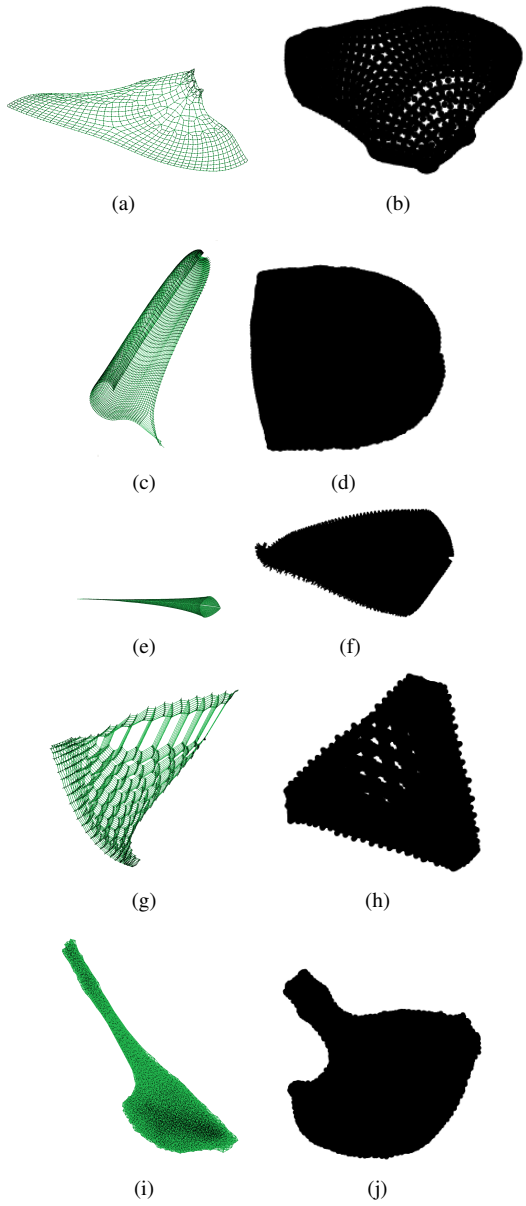


Fig. 10. Five different graphs each with two layouts (our proposed algorithm: 10(a), 10(c),10(e),10(g),10(i) and **fsdp**: 10(b), 10(d),10(f),10(h),10(j)), 10(a) and 10(b) netz4504 [12], 10(c) and 10(d) rajat07 [12], 10(e) and 10(f) rw5151 [12], 10(g) and 10(h) tuma2 [12] and 10(i) and 10(j) wing-nodal [12]. The details of all graphs are represented in Table II

order to rotate the layout. Also, having a variety of 3D views for the same graph can also be beneficial for some applications.

TABLE II
DETAILS OF THE GRAPHS SHOWN IN FIGURES 10, 9, 8, 7, 6, 5 AND 4 .

Graph	$\|N\|$	$\|E\|$	Run-Time(s) Our algorithm	Run-Time(s) sfdp
Crack	10240	30380	4.51	10.12
delaunay-n13	8192	24547	1.98	6.35
Dimacs10-data	2851	15093	1.04	2.35
epb1	14734	80319	11.2	12.6
fe-4elt2	11143	32818	4.213	8.41
jagmesh1	936	2664	0.233	0.53
jagmesh4	1440	4032	0.324	0.99
jagmesh7	1138	3156	0.27	0.64
jagmesh8	1141	3162	0.326	0.59
netz4504	1961	2578	0.363	1.15
rajat07	14842	24571	3.6	11.36
rw5151	5151	20199	2.45	3.3
stokes64	12546	65792	8.18	9.8
tuma2	12992	20925	2.94	9.47
wing-nodal	10937	75488	6.88	9.37
swang1	3169	17672	1.53	2.14
sierpinski	3282	6561	2.36	2.76

REFERENCES

[1] J. Barnes and P. Hut. A hierarchical o(nlogn) force-calculation algorithm. volume 324, pages 446–449. Nature, 1986.

[2] G. Bartel , C. Gutwenger, K. Klein and P. Mutzel *Graph Drawing: 18th International Symposium, GD 2010, Konstanz, Germany, September 21-24, 2010. Revised Selected Papers*, chapter An Experimental Evaluation of Multilevel Layout Methods, pages 80–91. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[3] P. Eades. On the future of graph drawing: Invited talk at the 18th International Symposium on Graph Drawing. http://graphdrawing.org/gd2010/invited.html,

[4] T. M. J. Fruchterman and E. M. Reingold Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

[5] P. Gajer, M. T. Goodrich, and S. G. Kobourov. A multi-dimensional approach to force-directed layouts of large graphs. *Computational Geometry*, 29(1):3 – 18, 2004. Special Issue on the 10th Fall Workshop on Computational Geometry, {SUNY} at Stony Brook.

[6] E. R. Gansner and Y. Hu Graph drawing. chapter Efficient Node Overlap Removal Using a Proximity Stress Model, pages 206–217. Springer-Verlag, Berlin, Heidelberg, 2009.

[7] E. R. Gansner and C.N. Stephen An open graph visualization system and its applications to software engineering. *SOFTWARE - PRACTICE AND EXPERIENCE*, 30(11):1203–1233, 2000.

[8] Y. Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[9] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15, April 1989.

[10] G.S. Kobourov. Force-directed drawing algorithms. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, volume 81 of *Discrete Mathematics and Its Applications*, chapter 12, pages 383–408. Chapman and Hall/CRC, 2013.

[11] R. Tamassia, editor. *Handbook of Graph Drawing and Visualization*, volume 81 of *Discrete Mathematics and Its Applications*. Chapman and Hall/CRC, 1 edition, 2013.

[12] UFL. Sparse graph. From University of Felorida Web Resource. http://www.cise.ufl, Accessed: 2016-03-09.

[13] C. Walshaw. A multilevel algorithm for force-directed graph drawing. In Joe Marks, editor, *Graph Drawing*, volume 1984 of *Lecture Notes in Computer Science*, pages 171–182. Springer Berlin Heidelberg, 2001.