“Forced” Force Directed Placement: a New Algorithm for Large Graph Visualization

Zakaria Boulouard[1], Lahcen Koutti[1], Anass El Haddadi[2], Bernard Dousset[3]

Abstract – *Graph Visualization is a technique that helps users to easily comprehend connected data (social networks, semantic networks, etc.) based on human perception. With the prevalence of Big Data, these graphs tend to be too large to decipher by the user’s visual abilities alone. One of the leading causes of this problem is when the nodes leave the visualization space. Many attempts have been made to optimize large graph visualization, but they all have limitations. Among these attempts, the most famous one is the Force Directed Placement Algorithm. This algorithm can provide beautiful visualizations for small to medium graphs, but when it comes to larger graphs it fails to keep some independent nodes or even subgraphs inside the visualization space. In this paper, we present an algorithm that we have named "Forced Force Directed Placement". This algorithm provides an enhancement of the classical Force Directed Placement algorithm by proposing a stronger force function. The “FForce”, as we have named it, can bring related nodes closer to each other before reaching an equilibrium position. This helped us gain more display space and that gave us the possibility to visualize larger graphs.*

Keywords*: Big Data, Force Directed Placement, Graphs, Graph Visualization, Large Graphs*

Nomenclature

FDP	Force Directed Placement
FFDP	“Forced” Force Directed Placement
T-FDP	“Temporal” Force Directed Placement

I. Introduction

Graph visualization has imposed itself lately as a blossoming research area. Indeed, it offers the possibility to quickly grasp complex issues such as network analysis, bioinformatics, or transport, based on human’s visual prowess. In general, graphs display abstract data as well as their patterns and connections and design them so that they could make more sense or tell a story. For example, let’s check out this social network {Zakaria – Taha, Zakaria – Amine, Taha – Hamza, Zakaria – Anass, Anass – Amine, Hamza – Zakaria}. We can understand that Zakaria is a friend of Anass and Amine is also a friend of Anass and so on. However, it is more challenging to find a general pattern of the relationships between all of these persons. On the other hand, the graph in Figure 1 can help us quickly understand the relationships between all of the individuals in the social network in order to find out that the graph has two clusters (groups of people) and that Zakaria is the person that connects them.

According to Purchase et al. [1], an excellent graph layout leads to a smooth cognition of the underlying information while a bad layout makes the graph confusing and obliges the reader to spend more time only to understand a part of the information. This urged Purchase to define some esthetic criteria to define a “good” graph layout.

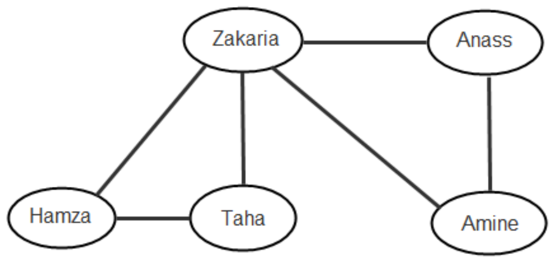


Fig. 1. Graph representing a social network

In order to respect these rules, several graph representations (or layouts) were suggested. Hu and Shi [2] have presented a survey of these graph layouts and categorized them into six distinct models:

- Spring-Electrical Model: In this model, further detailed in Section 2, the links are presented as springs while the nodes are displayed as steel rings. The primary purpose of this model's algorithms is to minimize the attractive-repulsive energy within the nodes.
- Stress Model: The main objective to achieve by adopting this model is to reduce the energy of the springs. For instance, Kamada and Kawai [3] provided an algorithm that reduces the stress energy within the edges by bringing the distance between the nodes to the ideal length of their connecting spring.

https://doi.org/10.15866/irecos.v12i2.12002

- Strain Model (or Classical MDS): This model tries to find the best inner product between the nodes' positions instead of the actual distance between them. The final embedding should be centered on the origin, and ideally the distance between the nodes is equal to the ideal edge length.
- MDS for Large Graphs: This model represents the improvements to the Strain Model, either by multiscaling it [4] or by approximating it [5]-[6]. Some algorithms suggest starting by drawing some reference nodes (called landmarks [7] or pivots [8]).
- High-Dimensional Embedding: This model provides coordinates to the nodes in a k-dimensional space, and then projects them into a regular 2D or 3D space using principal component analysis [9].
- Algorithms Based on the Spectral Information of the Laplacian: This model takes the node positioning problems and transforms them into problems of defining positions with a minimum weighted sum of the squared distances between the nodes [10]. This will make the solution simply become the eigenvector with the smallest positive eigenvalue of the weighted Laplacian matrix.

Hu and Shi's study [2] affirmed that, even if it was more costly than the other models cited earlier, the Spring Electrical Model gave a better graph visualization. Dong et al. [11] confirmed it by stating that, thanks to their ability to produce esthetically beautiful graphs, Spring-Electrical based algorithms, such as Force Directed Placement (FDP), are widely used. This has compelled us to direct our research toward the Spring-Electrical Model (Section 2) and to focus our work on improving the performance of the FDP algorithm to open more possibilities regarding large graph visualization (Section 3).

The improvement that we propose in the present paper is by replacing the function used in the classical Force-Directed Placement with a stronger one. The new function that we chose to call "FForce" will enable us to refine the nodes final equilibrium positions while bringing them closer to each other. As a result, we were able to gain more visualization space to draw more nodes and, therefore, visualize larger graphs.

Moreover, thanks to the dimension depending repulsion term of our function, we were also able to keep the disconnected smaller subgraphs close to each other and to the larger ones, while in other algorithms, they tended to push themselves further away and eventually leave the visualization space (screen). Tests were run, in Section 4, to observe the performances of the improved algorithm, called "Forced Force Directed Placement" (FFDP).

These tests are based on a comparison between FFDP, the standard FDP algorithm, and two other algorithms. The first one is used in the open source graph visualization tool Gephi [12], while the other one is used in a tool developed by previous members of our team, VisuGraph [13]. Section 5 concludes the paper and presents an idea of our future works.

II. Related Works

II.1. Spring-Electrical Model

To visualize graphs, Tutte [14] proposed to lay down at first, some of the nodes, then the later ones on the barycenters of their neighbors. The nodes' positions are easily determined by solving a system of linear functions. However, the final disposition is not always the best. Eades [15] has suggested a model called "Spring Layout" where the nodes are given an initial positioning, and then the edges (represented as springs) recall the nodes back to an equilibrium position corresponding to a global energy minimum. Fruchterman and Reingold later improved this work by introducing an algorithm called "Force Directed Placement" (FDP) [16]. In this algorithm, the attractive force of the spring between two connected nodes is proportional to the squared distance between them. Thus, the attraction force is expressed as:

$$F_a = -\frac{d^2(n_1, n_2)}{K} \qquad (1)$$

K is a parameter related to the nominal edge length of the final layout.

On the other hand, the repulsive force between any couple of nodes is inversely proportional to the distance between them. It is expressed as:

$$F_r = -\frac{K^2}{d(n_1, n_2)} \qquad (2)$$

The attraction force is applied amidst two adjacent nodes while the repulsion force is administered by each node on the rest of the nodes. Once the forces are calculated, the node will shift toward the position where the system attains a state of minimal energy. The algorithm starts by calculating the attraction forces between neighboring nodes; then, it calculates the repulsion forces between each pair of nodes and, finally, limits the total movement using a temperature criterion. For a graph $G = (V, E)$, every iteration requires a computation cost of $O(|E|)$ for calculating the attraction forces and $O(|V|^3)$ for calculating the repulsion forces [17]. Fruchterman and Reingold later considered reducing the complexity of this algorithm by introducing a cell grid into which the drawing space will be split. The objective here is to compute the local repulsive energy between nodes in non-neighboring cells. The problem with this approach is the fact that it may cause several calculation errors. That is because it neglects the repulsive forces that may exist between non-neighboring nodes.

Tunkelang [18] and Quigley [19] were able to remedy this problem by introducing quadtrees. A quadtree is a grouping of nodes presentable as a "Super-Node" with a repulsive force approximately equal to the total repulsive force of the nodes it contains. If a group of nodes is far enough from an individual node, it is safe to consider this group of nodes as a super-node.

Other methods were proposed, such as the multilevel approach, and a suitable metaheuristic was used in solving several common issues, such as graph partitioning [20], traveling salesman [21], and graph drawing [22].

This method, as described by Hu and Shi [2], has three steps: coarsening, coarsest graph layout, and refinement. In the first step, a series of coarser and coarser graphs, $G_O = G, G_1, G_2, \dots, G_n$, is engendered. In this series, every graph G_{k+1}contains a small number of nodes and edges along with information about its parent G_K. The coarsening stops when we reach a graph with the smallest number of nodes. Therefore, the layout of the coarsest graph becomes cheaper, and laying out the other graphs is prolonged and refined recursively.

The algorithms cited earlier tend to fall into local minima when the graphs to be drawn reach a certain size level. In order to answer Big Data related visualization requests, we need an algorithm that is able to draw large and visible graphs.

II.2. Graph Visualization Tools

Several trials for algorithms have been launched in order to have better large graph visualization. In this paragraph, we will focus on the most important tools that have adopted Spring-Electrical Model based algorithms. A more detailed description will be dedicated to Gephi [12] and VisuGraph [13], the tools utilizing the closest algorithms to ours. Gephi is known for being one of the most famous tools available on the market while VisuGraph is a tool that was developed by former members of our research team and the present work is an amelioration of it. A comparison between the visualization provided by our algorithm “FFDP” and the algorithms adopted by Gephi and VisuGraph will be demonstrated in Section 4.

1)OGDF: Developed by Chimani et al. [23], it is an open source C++ library providing graph drawing solutions among other possibilities. It is an algorithmic layer to be used within other programs to be developed.

2)Cytoscape [24]*:* Licensed as an open source software platform. Cytoscape was originally designed for biological research purposes. It later became a general platform for large graph analysis and visualization. It is based on Java, but it also provides a JavaScript library for web oriented development.

3)D3.js: Short for “Data Driven Documents for JavaScript”, it is an open source JavaScript library that allows amazing rendering charts out of diverse data sources using HTML, SVG, and CSS. This library, developed by Bostock et al. [25], is capable of some seriously advanced visualizations with complex data sets and allows for smooth interaction and sharing.

4)Gephi:, It is a free software program for graph visualization and analysis developed by Jacomy et al. [12]. This tool comes both as an executable and as programming APIs, providing the most common graph visualization algorithms.

The most important visualization algorithm provided by Gephi is called “ForceAtlas2” (FA2). It is a Force-Directed Placement algorithm developed by the Gephi team. It presents the graph as a physical system where nodes push each other away using a repulsive force while links attract the connected nodes back using an attractive force.

The basic expression of the Attractive Force (F_a) according to this model is equal to the distance between these nodes:

$$F_a = d(n_1\,, n_2) \tag{3}$$

The Repulsive Force (F_r) between two nodes, however, depends on the degrees of the nodes. It allows centering the highly connected nodes while repelling the less connected ones to the suburbs:

$$F_r = K_r \frac{deg(n_1 + 1) \cdot deg\ (n_2 + 1)}{d(n_1, n_2)} \tag{4}$$

K_r is a coefficient to be fixed by the settings, and the (+1), different from Noack’s expression [26], is a term added so that we can ensure that even the nodes with a 0 degree can have a repulsive force.

Combining these two forces creates a movement that converges to an equilibrium position.

This algorithm may provide a display that eases the visual interpretation of the data structure under study. However, it doesn’t take the nodes attributes into consideration during the positioning process. This can be a problem if the coordinates were to be used in the analysis.

5)VisuGraph: Developed within our team at IRIT [13]. Its approach regarding large graph visualization is particularly interesting with its two principal features:

- Graph Visualization.
- Time Dependent Evolutionary Aspect..

a) *Graph Visualization:* Loubier has proposed a minor modification of the FDP by setting the attraction and repulsion forces as:
The attraction force:

$$F_a(u, v) = \frac{\beta \cdot d_{uv}^{\alpha_a}}{K} \tag{5}$$

“β” is a constant coefficient.
“d_{uv}” is the distance between two nodes u and v.
“α_a” is a coefficient used to alter the attraction between the two nodes by either increasing or decreasing it.
“K” is a coefficient computed according to the dimensions of the drawing space:

$$K = \sqrt{\frac{L \cdot l}{N}} \tag{6}$$

“L” is the length of the drawing space, and “l” is its width.

The repulsion force:

$$F_r(u,v) = \frac{\alpha_r \cdot K^2}{d_{uv}^c} \qquad (7)$$

"c" is a constant coefficient.
"α_r"is a coefficient used to alter the repulsion between the nodes u and v by either increasing or decreasing it.

b) Evolutionary aspect using Time Slices: Loubier [13] has noted that when the analysis is time dependent, a graph can send mixed signals. Therefore, she adopted a time based graph presentation where each "Time Slice" represents a particular period.
Taking the temporal dimension into account in the graph visualization goes in two steps:

- First, a global time independent graph is drawn.
- Second, virtual nodes representing the time slices are scattered in the drawing space, while the graph nodes are positioned close to the virtual nodes representing their correspondent time slice.

III. Forced Force-Directed Placement

The proposed solution is an alteration of the standard FDP. This algorithm is called "Forced Force Directed Placement" (FFDP).

The algorithm itself is similar to the standard FDP but with stronger attraction and repulsion forces. These forces, combined to form one function called "FForce", are applied to the node positions in order to find equilibrium positions where the nodes are brought closer to each other compared with the positions generated by the standard FDP. This approach provides more drawing space and thereby the ability to draw more nodes.

III.1. Attraction

This function represents the attraction force applied on the links represented as coil springs.

It was first expressed as:

$$F_a(d) = \alpha \cdot (d - d_0)^3 \qquad (8)$$

d is the Euclidian distance between two nodes.

$d - d_0$ is the gap between the current position and the equilibrium position.

$d_0$such as $F_a(d_0) = 0$ is the equilibrium length of the spring. It allows adapting the graph drawing to the screen's size. It depends on the screen's size as well as on the graph's density.

It is important to mention that α represents the links' hardness. This makes it an essential factor for good graph visibility by having a direct relationship with the number of links. In other words, the more the number of the links increases, the more the graph will have a tendency to compact. Therefore, we need to release it.

Example: Imagine we have a graph with 4 nodes that are all connected. We will eventually have 6 links and a full graph matrix.

In this case, in order to have a clear visualization, we need to release the nodes, thereby increasing the value of α. Moreover, our α also needs to depend on the number of the graph's links.

Let us now improve our example and suppose that our links do not have the same weight. It is very common to have links weighed 1, and others weighed 10 or even 100.

In this case, the difference between the links' weights will certainly influence the positioning of the connected nodes and consequently the graph's visualization. To remedy this situation, our α also needs to depend on the links' weights.

To sum up, the greater the number of the links is and the stronger they are, the more the graph tends to compact and to bring itself to the screen's center. Therefore, what we need is a coefficient α that is able to remedy these excesses of number and weights of links.

An expression of the value of α was proposed, such as:

$$\alpha = \sqrt{N_L \cdot \overline{W_L}} \qquad (9)$$

with N_L being the number of the links and $\overline{W_L}$ their average weight.

The square root will avoid an overflow of the value of α when the number of links increases.

The chart in Figure 2 shows the evolution of α according to the number of links.

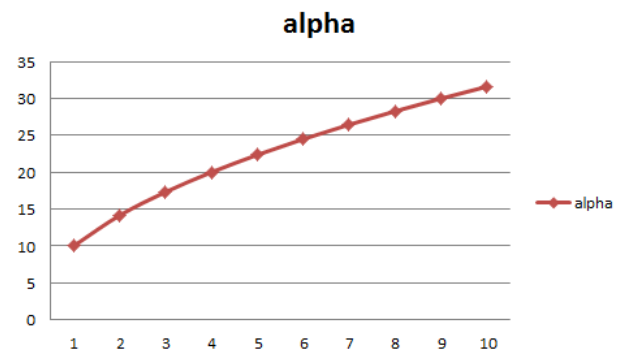


Fig. 2. The evolution of α according to the number of links

The cube term $(d - d_0)^3$ stands for the spring's elasticity. In the same hardness, the spring becomes loose when its length reaches d_0; then, it becomes harder in further values. In other words, the more we pull on the spring, the harder it comes back.

The chart in Figure 3 shows the evolution of the cube term $(d - d_0)^3$ according to the spring's length.

The chart in Figure 3 has a null derivative at one point, in the vicinity of our spring's equilibrium length, in this case, $d_0 = 20$. This would have been entirely correct if we had a perfectly soft spring, which is not the case since we have a coil spring.

The solution would be to add a linear term to our attraction function. This term will allow the attraction function to continue to act even if the spring's length approximates the equilibrium length.

Thus, the expression of the attraction function becomes:

$$F_a(d) = \alpha \cdot (d - d_0) + \alpha \cdot (d - d_0)^3 \qquad (10)$$

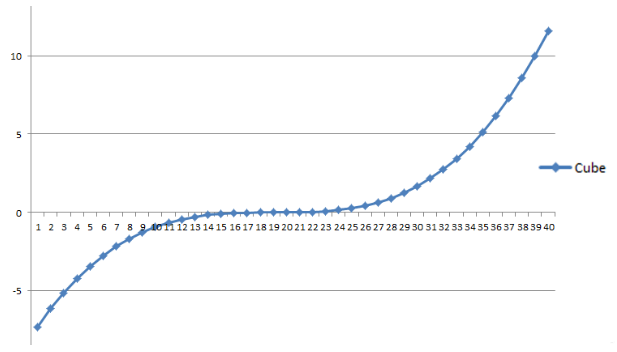


Fig. 3. The evolution of the cube term $(d - d_0)^3$ according to the spring's length

The chart in Figure 4 describes the evolution of the proposed expression of the attraction function according to the spring's length.

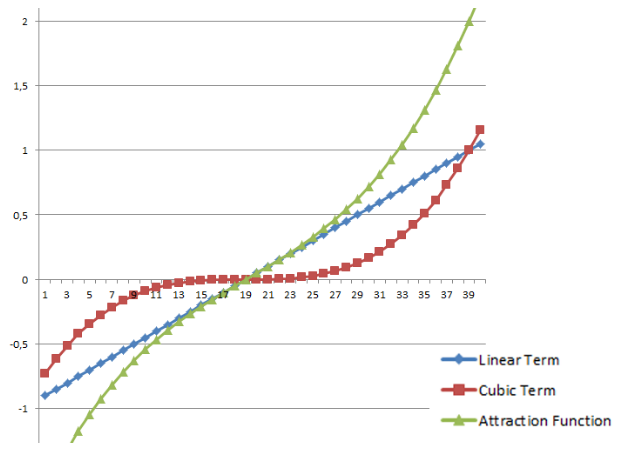


Fig. 4. The evolution of the proposed expression of the attraction function according to the spring's length

In Figure 4, we can see in the chart that we have, on the extremities, a stronger influence of the cube term of the function (in red). This term acts as a restoring function that brings the nodes back when the spring reaches a certain length. Whereas on the vicinity of the spring's equilibrium length, the linear term (in blue) prevents the nodes from colliding.

The overall evolution of the attraction force (in green), therefore, demonstrates more acceptable behavior. However, mathematically speaking, it is still incorrect.

The problem comes from the cube term that is symmetrical and could also, eventually, reach the value 0 or even exceed it. For lengths approximating 0, the spring would eventually break.

What we need is a term that would look similar to the cube term and yet have a vertical asymptote that would prevent our spring from breaking and making it act like a real coil spring that touches, stretches, and then stops pulling at a certain point.

A proposed expression to answer these specifications is the following:

$$F_a(d) = \alpha \cdot (d - d_0) + \frac{\alpha \cdot (d - d_0)^3}{d} \qquad (11)$$

The chart in Figure 5 models the evolution of the corrected expression of the attraction function according to the spring's length.

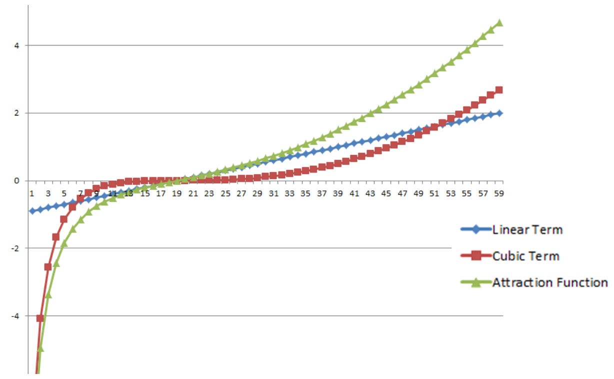


Fig. 5. The evolution of the corrected expression of the attraction function according to the spring's length

This chart displays a much more correct evolution of the attraction function, so that the expression to adopt is the one described in the equation (11).

III.2. Repulsion

It represents the repulsion force exercised by each node on all of the others forcing them and, subsequently, to repel each other.

It is a negative force that depends on a limit distance beyond which it becomes null. This means that at a certain point when the distance between two nodes becomes significant enough, the repulsion force becomes so small that it would be useless to calculate it. Therefore, we can assume that the repulsion force is inversely proportionate to the distance and should not exceed a limit distance beyond which the repulsion force is considered null. The threshold distance chosen is equal to half the screen so that the nodes that have already reached the monitor's border would not be able to repel each other and would remain on the screen.

The repulsion force is expressed by the function:

$$F_r(d) = -\frac{\beta}{d} \qquad (12)$$

However, there is something to pay attention to. This term is only correct in a 2D space.

If our graph were represented in a 3D or 4D space, the expressions of the repulsion force would become respectively, «$-\frac{\beta}{d^2}$» and «$-\frac{\beta}{d^3}$». Why?

The reason is simple; let's take two nodes A and B.

If we double the distance between these nodes, the repulsion force exerted by the node A to the node B, for example, would reduce to half if the nodes were in a 2D space, whereas in 3D, it would shrink to the third, then to the eighth in 4D.

This is essentially due to the force field that spreads throughout the space as the distance increases. In 2D, the force field spreads throughout a surface while in 3D, is spreads throughout a cone and in 4D, throughout a cube.

Thus, the repulsion function would be correct becoming:

$$F_r(d) = -\frac{\beta}{d^{n-1}} \quad (13)$$

with n being the number the dimensions of the space within which our graph is represented.

The chart in Figure 6 models the evolution of the repulsion force according to the distance in 2D (in blue), 3D (in red), then 4D (in green).

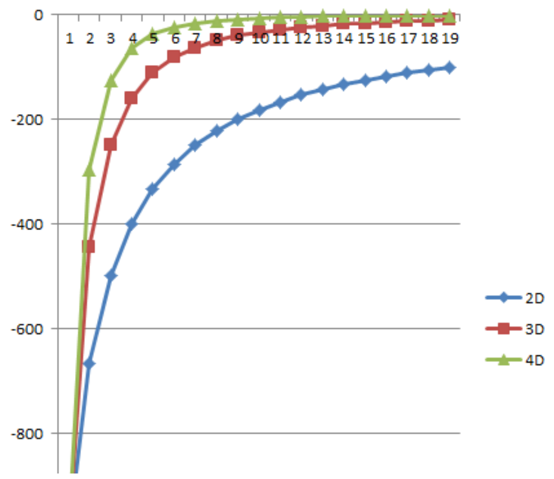


Fig. 6. The evolution of the repulsion force according to the distance

III.3. FForce Function

By combining the attractive and repulsive forces described earlier, we propose a function that represents the overall forces applied by each node of our graph on the other. We chose to call this function "FForce".

It is expressed as follows:

$$f_{ij}(d) = F_a(d) + F_r(d) \quad (14)$$

or in other terms:

$$f_{ij}(d) = -\frac{\beta}{d^{n-1}} + \alpha \cdot (d - d_0) + \frac{\alpha \cdot (d - d_0)^3}{d} \quad (15)$$

« f_{ij} » is the FForce applied between the nodes « *i* » and « *j* ».

« *d* » is the Euclidian distance between the nodes « *i* » and « *j* ».

« d_0 » is the spring's rest length.

« β », and « α » are coefficients that were given the following values:

$$\beta = d_0, \alpha = \sqrt{N_L \cdot \overline{W_L}}$$

The term «$-\frac{\beta}{d^{n-1}}$» expresses the repulsive force.

The terms «$\alpha \cdot (d - d_0) = \sqrt{N_L \cdot \overline{W_L}} \cdot (d - d_0)$» and «$\frac{\alpha \cdot (d-d_0)^3}{d} = \frac{\sqrt{N_L \cdot \overline{W_L}} \cdot (d-d_0)^3}{d}$» represent the linear and non-linear effects of the attraction on the edge, respectively.

III.4. FFDP Algorithm

Applying the "FForce" on a graph's nodes will be done using an algorithm we named "Forced Force Directed Placement" or (FFDP). This algorithm, as outlined in the pseudo-code (Algorithm 1), is inspired from the FDP algorithm as described in the pseudo-code by Hu and Shi [2].

The «break» term expresses the node's displacement increment. The objective of the FFDP algorithm is to find better equilibrium positions for the graph's nodes while gaining more space on the screen. It is important to mention that applying the FFDP algorithm gave us a rendering that is close to the FDP but with better control over the smaller subgraphs that tend to leave the screen in the standard FDP and, therefore, we could keep valuable information from being lost.

```
Algorithm 1 FForce Algorithm(G, x, tol, K)

input: graph, initial positions x, tolerance tol, and nominal edge length K
set break = initial break value
repeat
    $x_0 = x$
    For( i ∈ V ) {
        f = 0    // f is a 2-D or 3-D vector
        f ← f + FForce(i, j)    // Equation (15)
        $x_i \leftarrow x_i + break \quad f$
    }
until ($|x - x_0| < tol \quad K$)
```

The algorithm's complexity is O(n^4). It is true that it will be time-consuming when the numbers of nodes and links get higher, but this problem is easily overcome since the algorithm is supposed to run in a parallelized environment.

The results of running the FDP coupled with FForce will be described in the following section.

IV. Testing and Results

To test the visualizations produced by FFDP, we put it in comparison with the standard FDP, then with the T-FDP used in VisuGraph [13] and ForceAtlas2 used in

Gephi [12]. The first testing sample is a simple graph containing two nodes with a 1-D coordinate each: $x_1(0)$ and $x_2(3)$, connected with one edge. Figure 7 displays the evolution of the nodes' positions during the application of three algorithms (Standard FDP, FFDP, and ForceAtlas2).

Figure 7 made it clear that the FFDP, compared to the two other algorithms, could bring the nodes closer to each other before settling into an equilibrium position.

This will help us gain more display space for larger graphs.

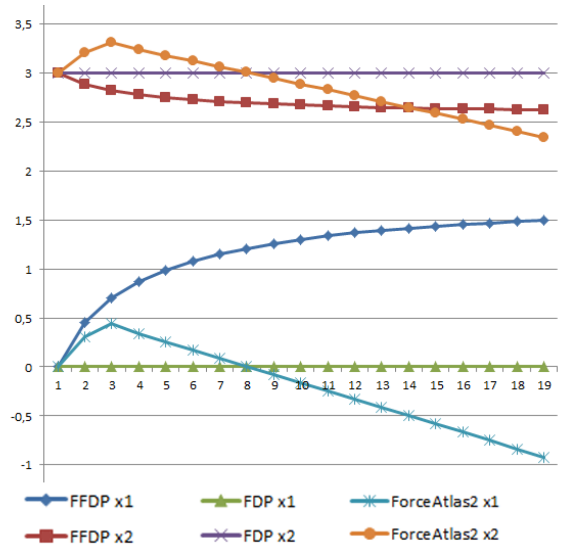


Fig. 7. Evolution of Nodes Coordinates after applying FDP, FFDP, and ForceAtlas2

The second testing sample is composed of 3 graphs with different sizes: the first graph, called "Small World" describes a small network gathering 20 persons represented as nodes connected with 40 links. The second graph, called "Facebook Ego 0" represents a Facebook network connecting 333 individuals represented by nodes with a total of 2,519 friendship relations represented by the links. The third graph, called "Marvel Superheroes", connects all the characters from the comic books by Marvel. They are represented with 104,690 nodes and 178,115 links.

Those graphs can be found among Gephi's test dataset that is available in the following link:

https://github.com/medialab/benchmarkForceAtlas2/blob/master/dataset.zip (last checked 3/17/2017).

The objective of this test is to display the ability of the FFDP to draw large graphs in a 2D space. The visualizations produced by FFDP are compared with those generated by the other two previously mentioned algorithms (Standard FDP and ForceAtlas2). Another algorithm, called "Temporal" Force Directed Placement (T-FDP), was also introduced to this test. It is the algorithm developed by our team in IRIT and implemented in the tool VisuGraph [13].

All these algorithms were implemented in Java and were tested in a computer with a standard configuration (Intel i3 processor and 4Gb RAM).

The next Tables I, II, and III show the results of the comparison that we proposed.

In the "Small World" graph, the best node positioning was provided by FFDP, closely seconded by ForceAtlas2. On the other hand, the visualizations provided by both the standard FDP and T-FDP were way too far from the expected result.

In the "Facebook Ego 0" graph, the displays provided by the standard FDP and T-FDP were practically similar. ForceAtlas2 was able to assemble the nodes in two groupings, but, on the other hand, it lost the smaller subgraphs that were not connected to the main one. FFDP was able to bring out a third less pronounced grouping in the main subgraph while keeping the three independent subgraphs close to the main one.

TABLE I
THE RESULTS OF RUNNING VARIOUS LAYOUT ALGORITHMS ON THE GRAPH "SMALL WORLD" (20 NODES, 40 LINKS)

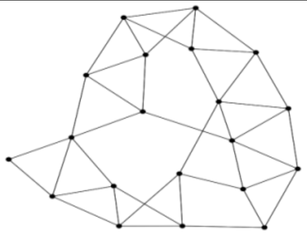
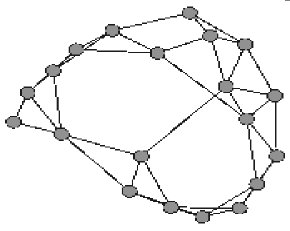
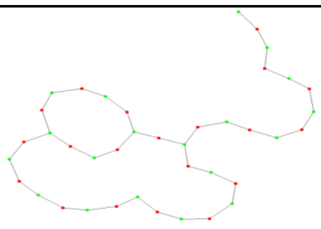
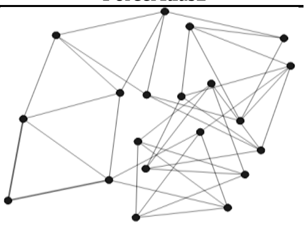
 FFDP	 ForceAtlas2
 T-FDP	 Standard FDP

TABLE II
THE RESULTS OF RUNNING VARIOUS LAYOUT ALGORITHMS ON THE GRAPH "FACEBOOK EGO 0" (333 NODES, 2519 LINKS)

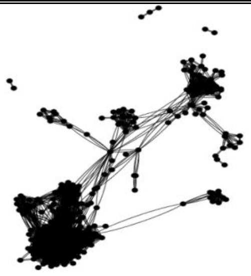
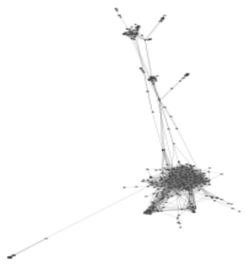
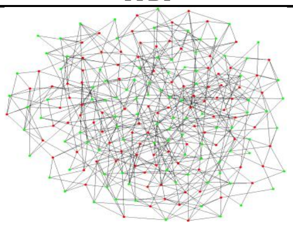
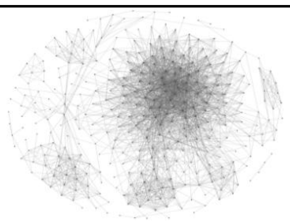
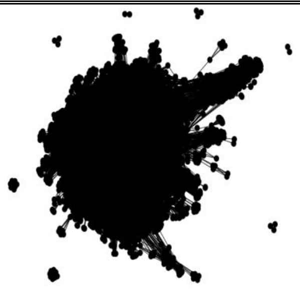
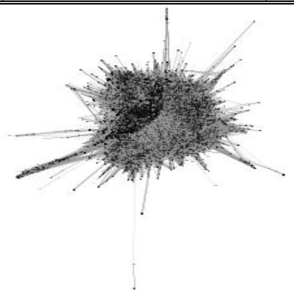
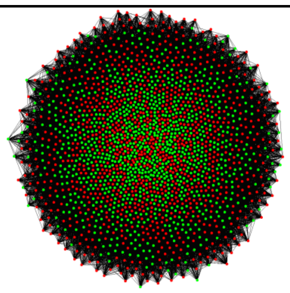
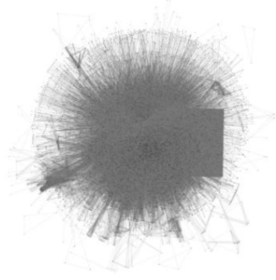
 FFDP	 ForceAtlas2
 T-FDP	 Standard FDP

TABLE III
THE RESULTS OF RUNNING VARIOUS LAYOUT ALGORITHMS ON THE GRAPH "MARVEL SUPERHEROES" (104690 NODES, 178115 LINKS)

	
FFDP	ForceAtlas2
	
T-FDP	Standard FDP

In the "Marvel Superheroes" graph, the standard FDP and T-FDP can barely be seen with a few nodes popping out of the cloud, but it is not enough to have a proper view of the graph. ForceAtlas2 was able to highlight two groupings of nodes and, yet again, lost the independent subgraphs. FFDP, on the other hand, managed to highlight more groupings while keeping the separate subgraphs in the screen.

It is important to mention that even though FFDP was able to draw larger graphs by bringing nodes closer to each other and gaining more space, this particular point can affect the quality of the drawing after a certain threshold. Indeed, in the "Marvel Superheroes" graph, it was nice to have the smaller subgraphs visible and close to the larger one. However, the nodes could have been less close to each other than how they actually were. This can be achieved by adding a limitation term to the attraction force.

V. Conclusion

This paper presented the "Forced Force Directed Placement" (FFDP) algorithm, as an improvement to the classical Force Directed Placement algorithm. FFDP allowed us to refine the nodes final positions and provide better equilibrium positions while bringing the nodes closer to each other. We were able to gain more visualization space to draw more nodes and provide larger graphs as a result. Moreover, thanks to the dimension depending repulsion term, were also able to keep the disconnected smaller subgraphs close to each other as well as to the larger ones, while in other algorithms they tend to push themselves further away and eventually leave the visualization space (screen). FFDP's rendering results were compared to those of the standard FDP algorithm along with two other versions, wherein the first one is used in the open source graph visualization tool Gephi [12], while the other one is used in a tool developed by previous members of our team, VisuGraph [13].

The FFDP algorithm will be integrated into XEWGraph [26], the large graph visualization service of the Competitive Intelligence tool Xplor EveryWhere [27]. The out of the box clustering and categorization provided by XEWGraph's hypergraph approach will give us two advantages. The first one is to be able to draw lighter, web destined graphs with a general view and then have a deeper view of more specific details according to the decision maker's needs. The second advantage is the ability to display these graphs on smaller screens such as smartphones.

FFDP's dimension depending repulsion term will open up the possibility to draw graphs on 3D or 4D spaces while guaranteeing a better convergence of the algorithm. This urges us to propose an expansion to the XEWGraph tool that will provide such visualizations.

References

[1] H. C. Purchase, Performance of Layout Algorithms: Comprehension, not Computation, (1998) *Journal of Visual Languages and Computing*, Elsevier, pp. 647-657.

[2] Y. Hu, L. Shi,. Visualizing large graphs, (2015) *Wiley Interdisciplinary Reviews Computational Statistics*, Wiley Periodicals Inc., pp. 115-136.

[3] T. Kamada, S. Kawai, An algorithm for drawing general undirected graphs, (1989) *Information Processing Letters*, Elsevier, pp. 7-15.

[4] R. Hadani, D. Harel, A multi-scale algorithm for drawing graphs nicely. (2001) *Discrete Applied Mathematics*, Elsevier, pp. 3-21.

[5] E.R. Gansner, Y. Hu, SC. North, A maxent-stress model for graph layout, (2013) *Comput Graph, Transactions on, IEEE*, pp. 927-940.

[6] M. Khoury, Y. Hu, S. Krishnan, CE. Scheidegger,. Drawing large graphs by low-rank stress majorization. (2012) *Comput Graph Forum*.

[7] V. De Silva, J. B. Tenenbaum, J. B., Global versus local methods in nonlinear dimensionality reduction, (2003) *Neural Information Processing Systems, Advances in.*, MIT Press, pp. 721-728.

[8] Brandes, U., Pich, C., Eigensolver methods for progressive multidimensional scaling for large data, *Proceedings of the 14th International Springer Symposium on Graph Drawing* (Page: 285, Year of publication: 2007).

[9] D. Harel, Y. Koren, High-Dimensional Embedding, (2004) *Journal of Graph Algorithms and Applications*, Brown University, pp. 195-214.

[10] K. M. Hall, An r-dimensional quadratic placement algorithm, (1970) *Management Science, Informs Journal on Computing*, pp. 219-229.

[11] W. Dong, F. Wang, Y. Huang, G. Xu, Z. Guo, X. Fu, K. Fu, An advanced pre-positioning method for force-directed graph visualization based on PageRank algorithm, (2015) *Computers & Graphics*, vol. 47, p 24-33.

[12] M. Jacomy, , T. Venturini, , S. Heymann, M. Bastian, *ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software*, (2014) PLoS ONE, vol. 9.

[13] E. Loubier, *Analyse et visualisation de données relationnelles par morphing de graphe prenant en compte la dimension temporelle*, PhD Thesis, IRIT, Paul Sabatier University, Toulouse, France, 2009.

[14] Tutte, W.T., How to draw a graph, *Proceedings of the London Mathematical Society* (Page 743, Year of Publication 1963).
[15] P. Eades, A heuristic for graph drawing, (1984) *Congressus Numerantium* 42, pp. 149-160.
[16] T. M. J. Fruchterman, E.M. Reingold, Graph drawing by force-directed placement, (1991) *Software Practice and Experience*, John Wiley & Sons Ltd, pp. 1129-1164.
[17] SG. Kobourov, Force-directed drawing algorithms, *Handbook of graph drawing and visualization*, (United States: CRC Press, 2013, 388–389).
[18] D. Tunkelang, , *A numerical optimization approach to general graph drawing*, Ph.D. Thesis, Carnegie Mellon University, Pennsylvania, United States, 1999.
[19] A. Quigley, *Large scale relational information visualization, clustering, and abstraction*, Ph.D. Thesis, Department of Computer Science and Software Engineering, University of Newcastle, Australia, 2001.
[20] Gupta, A., Karypis, G., and Kumar, V., Highly scalable parallel algorithms for sparse matrix factorization, *IEEE Transactions on Parallel and Distributed Systems* (Page 502, Year of Publication 1997).
[21] C. Walshaw, A multilevel approach to the traveling salesman problem, (2002) *Operations Research*, vol. 50, pp. 862–877.
[22] C. Walshaw, A multilevel algorithm for force-directed graph drawing, (2003) *Journal of Graph Algorithms and Applications*, vol. 7, pp. 253–285.
[23] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, P. Mutzel, The Open Graph Drawing Framework (OGDF), *Handbook of Graph Drawing and Visualization*, (United States: CRC Press, 2014, Chapter 17).
[24] http://www.cytoscape.org (last visited 3/17/2017).
[25] Bostock, M., Ogievetsky, V., Heer, J., D3: Data-Driven Documents, *Proceedings of the IEEE InfoVis Conference* (Year of Publication: 2011).
[26] Noack, A., 2004. An energy model for visual graph clustering. *Proceedings of the 11th International Springer Symposium on Graph Drawing* (Page 425, Year of Publication: 2003).
[27] Boulouard, Z., Koutti, L., El Haddadi, An., El Haddadi, Am., Fennan, A., XEWGraph: A Tool for Visualization and Analysis of Hypergraphs for a Competitive Intelligence System, *Proceedings of the 6th IEEE International Conference on Information Systems and Economic Intelligence (SIIE)*, (Page 66, Year of Publication: 2015).
[28] A. El Haddadi, *Fouille Multidimensionnelle sur les Données Textuelles Visant à Extraire les Réseaux Sociaux et Sémantiques pour leur Exploitation via la Téléphonie Mobile*, PhD Thesis, IRIT, Paul Sabatier University, Toulouse, France, 2011.

Authors' information

[1]LabSIV, Faculty of Sciences, Ibn Zohr University, Agadir, Morocco.
E-mails: zboulouard@gmail.com,
zakaria.boulouard@edu.uiz.ac.ma
lkoutti@yahoo.fr

[2]Department of Mathematics and IT, ENSA, Mohamed 1st University, Al Hoceima, Morocco.
E-mail: anass.elhaddadi@gmail.com

[3]SIG, IRIT, Paul Sabatier University, Toulouse, France.
E-mail: dousset.bernard@gmail.com

Zakaria Boulouard was born in 1988; he received his engineer's degree in software engineering in 2013. He is currently a PhD student at the Faculty of Sciences, Ibn Zohr University, Agadir, Morocco. His research interests include Big Data Visualization and Analytics, Data Science, Business Intelligence, and Competitive Intelligence.

Lahcen Koutti is currently a Professor at Department of Computer Science at the Faculty of Science, Ibn Zohr University, Agadir, Morocco. He received his PhD degree in Computational Physics in 1999 from University Paul Verlaine, France and the Habilitation degree in 2010, from Ibn Zohr University, Morocco. His research interests include Artificial Intelligence and Computer Vision. He is a member of the Computer Systems and Vision Laboratory.

Anass El Haddadi is a doctor of business intelligence from the University of Toulouse (France) and University Mohammed V of Rabat (Morocco) (2011). He is an associate professor in ENSA of Al-Hoceima. He is a member of the French Research Group in Competitive Intelligence. Since 2014, he is a co-president of Competitive Intelligence Day in Morocco; since 2015, he is the president of VSST Association Chapter Morocco; and since 2016, he is the vice president of ISKO-Maghreb (Morocco). His research interests include decision-support information systems, big data analytics, data visualization, and unstructured data management.

Bernard Dousset is an Emeritus Professor in the IRIT research laboratory in the University of Toulouse (France). He is the honorary president of the International Conference on Scientific and Technological Strategic Intelligence. He is a member of the French Research Group in Competitive Intelligence. He is the honorary president of VSST Association. His research interests include Applied Mathematics, Optimization, Statistics, Data Mining, Text Mining, Strategic Monitoring, and Competitive Intelligence.