ForceAtlas2, A Continuous Graph Layout Algorithm for Handy Network Visualization

Mathieu Jacomy, Sebastien Heymann, Tommaso Venturini, and Mathieu Bastian

MM. Jacomy and Venturini are with Sciences Po, medialab.

M. Heymann is with LIP6 - CNRS - Universite Pierre et Marie Curie.

M. Bastian is with the Gephi Consortium.

Abstract

We present ForceAtlas2, a force-directed algorithm implemented in the Gephi software. It aims at giving a readable shape to a network (spatialization). We do not claim a theoretical advance but an attempt to integrate different techniques such as Barnes Hut simulation, degree-dependent repulsive force, local and global adaptive temperatures. The algorithm benefits from many feedbacks and is developed in order to provide a fruitful experience to the user. We expose its complete functioning for the users that need a precise understanding of its behavior. We propose a benchmark of our compromise between performance and quality, and we discuss why we integrated its different features.

Index Terms

Force-directed placement, Gephi, Spatialization, Network Visualization, Social Networks Analysis

I. Introduction

This paper addresses two different publics. To Gephi users, we propose a complete description of the ForceAtlas2 algorithm and its settings. To the researchers or engineers interested in the development of spatialization algorithms, we propose to discuss our choices of features and implementation. If developing an algorithm is "research" and implementing it is "engineering", then a specificity of Gephi in general is to come from engineering rather than from research. This is why it looks so different to a software like Pajek. This is also why ForceAtlas2 is more about usability than originality. Our contribution to the mathematics of network spatialization is limited to the benchmark of a specific implementation of adaptive speed (step length selection). This paper focuses more about how classical techniques fit together in the perspective of a rich user experience - and which techniques do not.

We need to explain quickly how the user feedback lead us to the specific orientation of ForceAtlas2 (a continuous algorithm). We will expose the different techniques gathered in this layout, with a little formal language and many illustrations. We will discuss our implementation of step length selection with examples and a benchmark. And at last we will provide a short discussion about the general design of the algorithm.

II. A wild network visualization tool appears

In 2008 we started to develop Gephi [1], a software to visualize and manipulate networks, at the Maison des Sciences de l'Homme in Paris under the direction of Dana Diminescu [2]. Our

goal was to provide some network analysis methods to social scientists, without having them to learn graph theory.

Three reference softwares inspired us: Pajek (CITATION NEEDED), GUESS (CITATION NEEDED) and TouchGraph (CITATION NEEDED). TouchGraph offered a manipulative interface that we loved, but it had strong performance issues and the layout was not adapted to scale-free networks of a hundred nodes or more (high visual cluttering). Pajek is very powerful but not adapted to dynamic exploration (it is designed as a computation software, where visualization is a bonus). GUESS was the most adapted to our needs, being user centric and implementing state-of-the-art spatialization algorithms like GEM (CITATION NEEDED). We do not expose here the reasons why we created Gephi rather than just using GUESS, since it is a much larger discussion. But an important point for this paper is that we wanted a *continuous* layout, that runs homogeneously and that you can display. Visualizing the "live" spatialization is a key feature of Gephi and is much appreciated. It provides a very intuitive understanding of the layout process and its settings. It allows users to have a trial-error approach of the layout, that improves the learning curve of Gephi.

We developed ForceAtlas2 by combining existing techniques. We did it wildly. We did not start from a systematic review of academic papers, and we eventually redeveloped existing techniques. We implemented features when they were needed by users, and we tried to be user-friendly in the design of our settings[1]. We focused ForceAtlas2 on fluency and quality, because fluency is required by Gephi's manipulative user experience, and because researchers prefer quality over performance. The fundamentals of the algorithm are not sophisticated. As long as it runs, the nodes repulse and the edges attract. This pull for simplicity comes from a need for transparency. Social scientists disavow black boxes, and so do we. Our features change the forces or how they are simulated, but keep this model of continuous force directed layout: forces apply continuously as long as the layout is running. We give more details about our reasons at this end of this paper. Developing a continuous algorithm prevented us to implement many powerful techniques. We cite here some techniques that we intentionally avoided for focusing reasons. Simulated annealing (CITATION NEEDED) cannot be fully implemented, nor than any auto-stop feature (like Yifan

[1]When we reworked all the settings, we created a version "2" of "ForceAtlas" into avoiding a too confusing change. Both versions are still available in Gephi even if the first version is obsolete.

Hu [9], also implemented in Gephi). Our layout stops exclusively at the user's request. Phased strategies, used for example by OpenOrd [8], are by definition incompatible, even if in this case it allows OpenOrd to spatialize much larger networks. Graph coarsening (CITATION NEEDED) cannot be implemented for the same reason. The classic optimization of nodes' starting positions (CITATION NEEDED) would also break the continuity. At last, strategies where forces do not apply homogeneously do not fit either. It is notably the case of the old Kamada Kawai (CITATION NEEDED) and more recently GEM (CITATION NEEDED). We abandoned many techniques by keeping ForceAtlas2 continuous. But most of these are actually optimizations, and our performances are still coherent with the size of networks managed by Gephi (as we will see). We were able to implement qualitative features that impact the placement of the nodes, like a degree-dependent repulsion force suggested by Noack (CITATION NEEDED), gravitation (CITATION NEEDED), and more. We also implemented the Lin-Log forces proposed by Noack [6], a great inspiration for us, since his conception of layout quality corresponds to researchers' needs (a visual interpretation of modularity).

III. Anatomy of ForceAtlas2

ForceAtlas2 is a force directed layout: it simulates a physical system in order to spatialize a network. Nodes repulse each other like magnets, while edges attract their nodes, like springs. These forces create a movement that converges to a balanced state. This final configuration is expected to help the interpretation of the data.

The force-directed drawing has the specificity of placing the nodes function of the other nodes. This process depends only on the connexions between nodes. Eventual attributes of nodes are never taken in account. This strategy has drawbacks. The result varies depending on the initial state. The process is not deterministic, and the coordinates of each point do not reflect any specific variable. The result cannot be read as a Cartesian projection. You cannot interpret the position of a node for itself, you have to compare it to the others. Despite this issue, the technique has the advantage to allow a visual interpretation of the structure. Its very essence is to turn structural proximities into visual proximities, facilitating the analysis and in particular the analysis of

social networks. Noack [3] has shown that the proximities express communities[2]. Other types of layouts allow a visual interpretation of the structure, like the deterministic visualization XXX (CITATION NEEDED), but they do not manifest the modular aspect of the structure.

A. Energy Model

Every force-directed algorithm relies on a certain formula for the attraction force and a certain formula for the repulsion force. The "spring-electric" layout [10] is a simulation inspired by real life. It uses the repulsion formula of electrically charged particles ($F_r = k/d^2$) and the attraction formula of springs ($F_a = -k.d$) involving the geometric distance d between two nodes[3]. Fruchterman and Rheingold [7] created an efficient algorithm using custom forces (attraction $F_a = d^2/k$ and repulsion $F_r = -k^2/d$, with k adjusting the scaling of the network).

Sixteen years later, Noack [6] explains that the most important difference among force-directed algorithms is the role played by distance in graph spatialization. In physical systems, forces depend on the distance between the interacting entities: closer entities attract less and repulse more than farther entities and vice versa. The dependence between distance and forces can be linear, exponential or logarithmic. The Spring model for example, replicates precisely the physical forces to which it is inspired, thereby establishing a linear proportionality between the distance and the force (as for the spring attraction) and as a square proportionality between the distance and the force, as for electromagnetic repulsion. Noack defines the energy model or "(attraction,repulsion)-model" of a layout as the exponent taken by distance in the formulas used to calculate attraction and repulsion (the log being considered as the 0^{th} power). For example, the model of the spring-electric layout is $(1, -2)$.

The (attraction,repulsion)-model of ForceAtlas $(1, -1)$ has an intermediate position between Noack's LinLog $(0, -1)$ and the algorithm of Fruchterman and Rheingold $(2, -1)$.

[2]Noack relies on the very intuitive approach of Newman [4], [5]: actors have more relations inside their community than outside, communities are groups with denser relations. Newman proposes an unbiased measure of this type of collective proximity, called "modularity". Noack [3] has shown that force-directed layouts optimize this measure: communities appear as groups of nodes. Force-directed layouts produce visual densities that denote structural densities.

[3]Actually, non-realistic forces were used since the beginning, noticeably by Eades [10] in its pioneer algorithm. Fruchterman and Rheingold were inspired by Eades' work, and they notice that despite using the spring metaphor to explain his algorithm, the attraction force is not the one of a spring.

Noack [3] states that "Distances are less dependent on densities for large $a - r$, and less dependent on path lengths for small a" (the "density" is the ratio of actual edges on potential edges). It means that visual clusters denote structural densities when $a - r$ is low, that is when the attraction force depends less on distance, and when the repulsion force depends more on it. ForceAtlas2's ability to show clusters is better than Fruchterman and Rheingold's algorithm and worst than LinLog (fig 1).

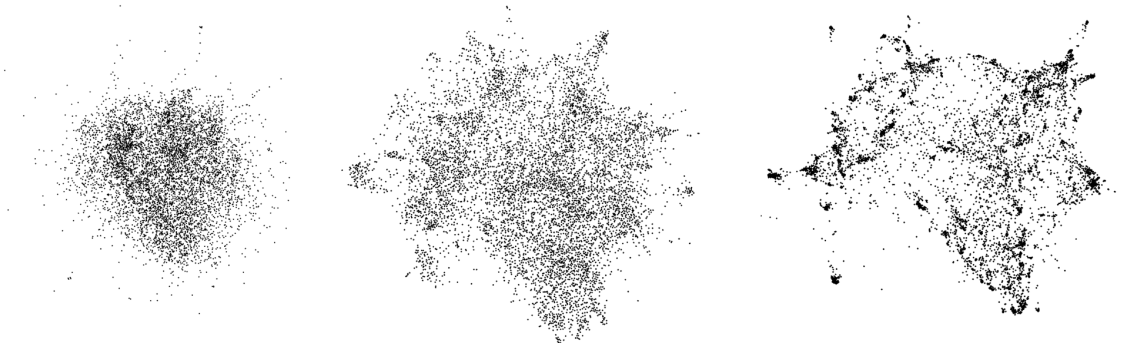

Fig. 1. Layouts with Fruchterman-Rheingold ($a - r = 3$), ForceAtlas2 ($a - r = 2$) and the LinLog mode of ForceAtlas2 ($a - r = 1$).

1) A classical attraction force: The attraction force F_a between two connected nodes n_1 and n_2 has nothing remarkable. It depends linearly on the distance $d(n_1, n_2)$. We will explain later why there is no constant adjusting this force.

$$F_a(n_1, n_2) = d(n_1, n_2) \tag{1}$$

2) Repulsion by degree: A typical use case of ForceAtlas2 is the social network. A common feature of this type of network is the presence of many "leaves" (nodes that have only one neighbor). This is due to the power-law distribution of degrees that characterizes many real-world data. The forests of "leaves" surrounding the few highly connected nodes is one of the principal sources of visual cluttering. We take in account the degree of the nodes in the repulsion so that this specific visual cluttering is reduced.

The idea is to bring poorly connected nodes closer to very connected nodes. Our solution is to tweak the repulsion force so that poorly connected nodes and very connected nodes repulse less. As a consequence they will end up being closer in the balanced state (fig 2). Our repulsion

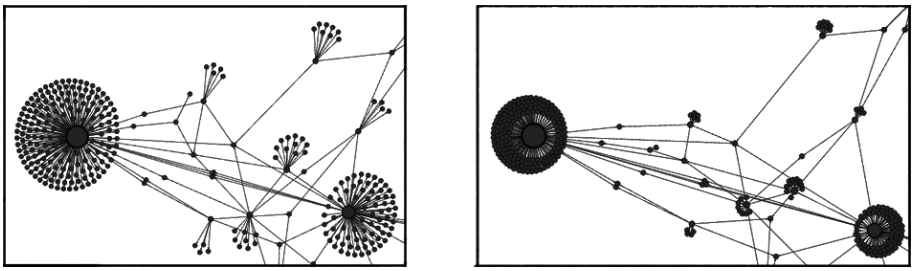

Fig. 2. Fruchterman-Rheingold layout on the left (regular repulsion) and ForceAtlas2 on the right (repulsion by degree). While the global scheme remains, poorly connected nodes are closer to highly connected nodes.

force F_r is proportional to the produce of the degrees plus one ($deg + 1$) of the two nodes. The coefficient k_r is defined by the settings.

$$F_r(n_1, n_2) = k_r \frac{(deg(n_1) + 1)(deg(n_2) + 1)}{d(n_1, n_2)} \tag{2}$$

This formula is very similar to the edge repulsion proposed by Noack [6] except that he uses degree and not the degree plus one. The $+1$ is important as it ensures that even nodes with a degree of zero still have some repulsion force. We speculate that this feature has more impact on the result and its readability than the (attraction,repulsion)-model.

B. Settings

We detail now the settings proposed to the user, what they implement, and their impact on the layout. Most of these settings allow the user to affect the placement of nodes (the shape of the network). They allow the user to get a new perspective on the data and / or to solve a specific problem. They can be activated while the layout is running, thus allowing the user to see how they impact the spatialization.

1) LinLog Mode: Andreas Noack did a great job on placement quality measures (CITATION NEEDED). Its LinLog energy model arguably provides the most readable placements, since it results in a placement that corresponds to Newman's modularity (CITATION NEEDED), a widely used measure of community structure. The LinLog mode just uses a logarithmic attraction force.

$$F_a(n_1, n_2) = log(1 + d(n_1, n_2)) \tag{3}$$

This formula is different from Noack's since we add 1 to the degree to manage disconnected nodes ($log(0)$ would produce an error). We have already seen that this energy model has a strong impact on the shape of the graph, making the clusters tighter (fig 1). We also observed that it converges slowly in some cases. Switching from regular mode to LinLog mode needs a readjustment of the scaling parameter.

2) Gravity: Gravity is a common improvement of force-directed layouts. This force $F_g(n)$ prevents disconnected components (islands) to drift away. It attracts nodes to the center of the spatialization space. Its main purpose is to compensate repulsion for nodes that are far away from the center. In our case it needs to be weighted like the repulsion:

$$F_g(n) = k_g(deg(n) + 1) \tag{4}$$

k_g is set by the user.

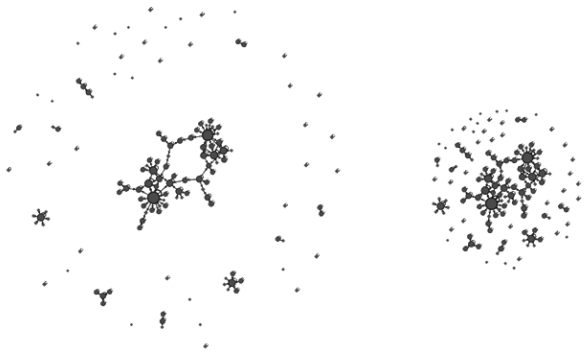

Fig. 3. ForceAtlas2 with gravity at 2 and 5. Gravity brings disconnected components closer to the center (and affects a little the shape of the components as a side-effect).

The "Strong gravity" option sets a force that attracts more the nodes that are distant from the center ($d(n)$ is this distance). This force has the drawback of being so strong that it is sometimes stronger than the other forces. It may result in a biased placement of the nodes. But its advantage is to force a very compact layout, which may be useful for certain purposes.

$$F'_g(n) = k_g(deg(n) + 1)d(n) \tag{5}$$

3) Scaling: A force-directed layout may contain a couple of constants k_a and k_r playing an opposite role in the spatialization of the graph. The attraction constant k_a adjusts the attraction force, and k_r the repulsion force. Increasing k_a reduces the size of the graph while increasing k_r expands it. In the first version of ForceAtlas, user could modify the value of both variables. For practical purposes, however, it is better to have only one single scaling parameter. In ForceAtlas2, the scaling is k_r while there is no k_a. The higher k_r, the larger will be the graph.

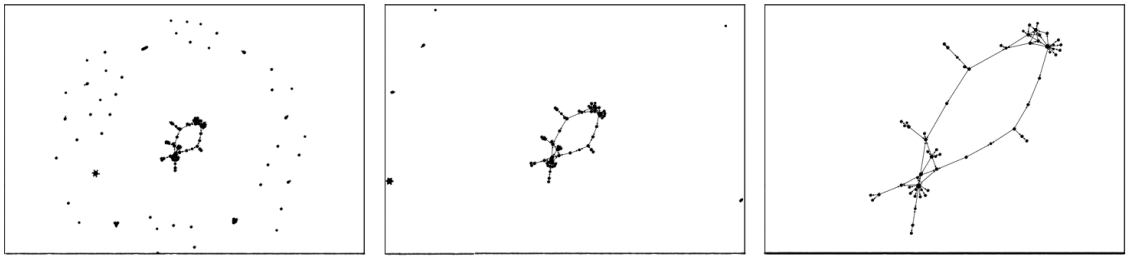

Fig. 4. ForceAtlas2 with scaling at 1, 2 and 10. The whole graph expands as scaling affects the distance between components as well as their size. Note that the size of the nodes remains the same ; scaling is not zooming.

4) Edge weight: If the edges are weighted, this weight be taken into consideration in the computation of the attraction force. If the setting "Edge Weight Influence" δ is set to 0, the weights are ignored. If it is set to 1, then the attraction is proportional to the weight. Values above 1 emphasize the weight effects. This parameter is used to modify the attraction force according to the weight $w(e)$ of the edge e:

$$F_a = w(e)^{\delta} d(n_1, n_2) \tag{6}$$

5) Dissuade Hubs: ForceAtlas2 has a "Dissuade Hubs" mode that, once activated, affects the shape of the graph by dividing the attraction force of each node by its degree plus 1 for nodes it points to. When active, the attraction force is computed as follows:

$$F_a(n_1, n_2) = \frac{d(n_1, n_2)}{deg(n_1) + 1} \tag{7}$$

This mode is meant to grant authorities (nodes with a high indegree) a more central position

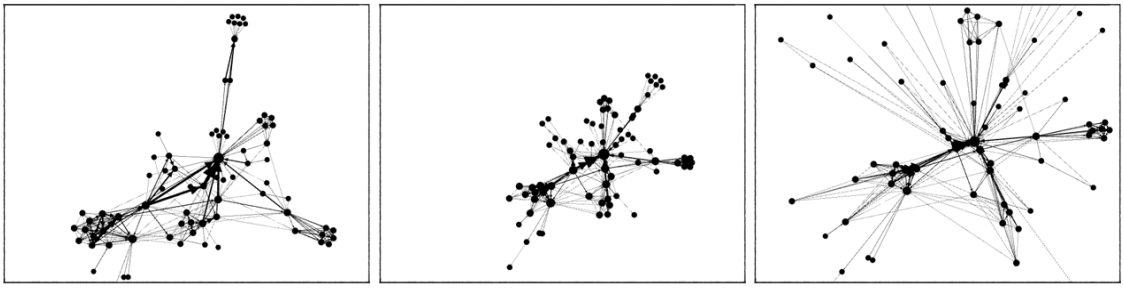

Fig. 5. ForceAtlas2 with Edge Weight Influence at 0, 1 and 2 on a graph with weighted edges. It has a strong impact on the shape of the network.

than hubs (nodes with a high outdegree)[4]. This is useful for social networks and web networks, where authorities are sometimes considered more important than hubs. "Dissuade Hubs" tends to push hubs to the periphery while keeping authorities in the center.

6) Prevent Overlapping: With this mode enabled, the repulsion is modified so that the nodes do not overlap.

The idea is to take in account the size of the nodes $size(n)$ in computing the distance $d(n_1, n_2)$ both in the attraction force and in the repulsion force.

- $d'(n_1, n_2) = d(n_1, n_2) - size(n_1) - size(n_2)$ is the "border-to-border" distance preventing overlapping.
- if $d'(n_1, n_2) > 0$ (no overlapping) then we use d' instead of d to compute forces:
  $F_a(n_1, n_2) = d'(n_1, n_2)$
  $F_r(n_1, n_2) = k_r \frac{(deg(n_1)+1)(deg(n_2)+1)}{d'(n_1, n_2)}$
- if $d'(n_1, n_2) < 0$ (overlapping) then no attraction and a stronger repulsion:
  $F_a(n_1, n_2) = 0$
  $F_r(n_1, n_2) = k'_r(deg(n_1)+1)(deg(n_2)+1)$
- if $d'(n_1, n_2) = 0$ then there is no attraction and no repulsion

In Gephi's implementation k'_r is arbitrary set to 100. Note that the swinging measure is biased due to this option, that's why we also implemented a diminishing factor on the local speed (dividing it by 10). It is important to notice that this mode adds a considerable friction in the

[4]Here we improperly use the concepts of Hub and Authority defined by Kleinberg [12]. We do not actually compute the HITS algorithm for performance issues.

convergence movement, slowing spatialization performances. It is necessary to apply it only after the convergence of graph spatialization.

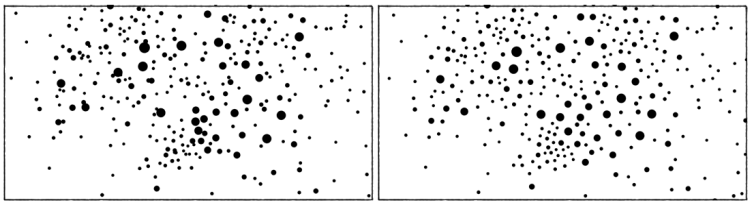

Fig. 6. ForceAtlas2 without and with the nodes overlapping prevention.

7) Approximate Repulsion: In order to improve spatialization performances on big graphs, we implemented the optimization of Barnes Hut [13]. Relying on an approximate computation of repulsion forces, such optimization generates approximation and may counter-productive on small networks, thus we allow the user to disable it. Out of the side effects of the approximation, it does not impact the shape of the layout. Without the Barnes and Hut optimization, the complexity time is $o(n^2)$ where n is the number of nodes.

IV. Performance optimization

A. The issue of speed

When employing a force-based layout, users have to deal with a speed/precision trade-off. This issue is a consequence of using a simulation of the forces. It appears in any force-directed algorithm as well as in other types of simulations. The time is not something continuous in the simulation, because it is computed step by step. If you use many computing steps, you have a precise simulation but it takes longer to compute: it is slow. If you choose few steps it is computed quickly but the simulation is imprecise. Reducing the number of steps is making a rougher approximation of the system. The proper term to discuss this would be “step length”, since it is the mathematical variable that we actually use. But we will prefer here the term of “speed”, because it is closer to the experience of users. The speed of the simulation is just like the step length: a high speed means long steps (less precision), a low speed means short steps (more precision). In a force-directed algorithm, increasing the speed makes the precision drop. We cannot have speed and precision at the same time. The effect of the approximation is that

some nodes become unable to find a stable position and start oscillating around their balancing position (fig 7).

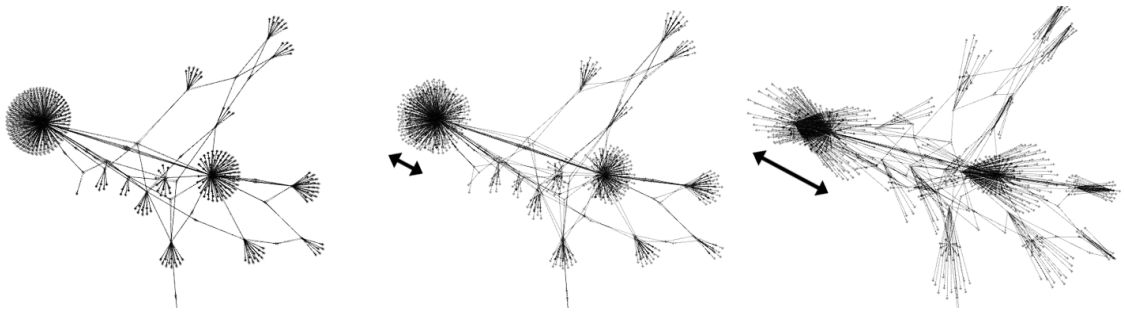

Fig. 7. Fruchterman-Rheingold layout at speeds 100, 500 and 2500 (superposition at two successive steps). The oscillation of nodes increases with speed.

This oscillation problem is known as a problem of "temperature", because we can assimilate the movement of a node to the temperature of a molecule. Different solutions exist: local temperatures as featured in GEM (CITATION NEEDED), adaptive cooling as featured in Yifan Hu [9] or simulated annealing (CITATION NEEDED). ForceAtlas2 features its own implementation of local temperatures as well as adaptive cooling, but in the perspective of a continuous layout. In terms of "speed vs. precision", since users are more comfortable with these concepts, we compute an optimal speed for each node as well as for the whole graph. Our strategy is to measure oscillations and to compute a speed that allows only a certain amount of oscillation. This amount is set by the user as "Tolerance (speed)". In Gephi's implementation, we set three default values: 0.1 under 5000 nodes, 1 up to 50000 nodes and 10 above. We now describe how this feature works.

B. Adapting the local speed

We implemented a strategy aiming at optimizing the convergence. Researchers often visualize scale-free networks where some nodes gather a huge amount of edges. Highly connected nodes have a high temperature. They tend to oscillate quickly, and need a high precision, thus a low speed. Poorly connected nodes are very stable and thus can go high speed. If we have different speeds for different nodes, we can have a much better performance. Our strategy is to determine

the speed of each node by observing its oscillation, like in GEM [**?**]. But our implementation is actually quite different.

Our version of oscillation is based on the forces applying to each node, and we call it "swinging" (oscillation is about distances). We define the swinging $swg(n)$ of a node n as the divergence between the force applied to n at a given step and the force applied n at previous step. Intuitively, the more the node is asked to change direction, the more it swings. $F_{(t)}(n)$ it the result force applied to n at step t.

$$swg_{(t)}(n) = |F_{(t)}(n) - F_{(t-1)}(n)| \tag{8}$$

For a node moving towards its balancing position, $swg(n)$ remains close to zero. A node that is diverging, on the contrary, has a high swinging and its movement needs to be slowed down to make it converge. The speed $s(n)$ of a node n determines how much displacement $D(n)$ will be caused by the resultant force $F(n)$: $D(n) = s(n)F(n)$. The resultant force is the sum of all forces applied to each node (attraction, repulsion and gravity: $F = F_a + F_r + F_g$). So in ForceAtlas2 the speed is different for every node, and computed as follows:

$$s(n) = \frac{k_s s(G)}{(1 + s(G)\sqrt{swg(n)})} \tag{9}$$

$s(G)$ is the global speed of the graph (see below). k_s is a constant set to 0.1 in Gephi's implementation.

The more a node swings, the more it is slowed. If there is no swinging, the node moves at the global speed. As a protection, we implemented an additional constraint that prevents the local speed from being too high, even in case of very high global speeds.

$$s(n) < \frac{k_{smax}}{|F(n)|} \tag{10}$$

$k_{smax} = 10$ in Gephi's implementation.

C. Adapting the global speed

At each step, two global values are computed and used to set the global speed: the global swinging and the global effective traction.

The global swinging $swg(G)$ represents the quantity of erratic movement present in the global movement of the graph. It is the sum of local swinging values, weighted by the degree of each node as in our repulsion force (degree+1).

$$swg(G) = \sum_n (deg(n) + 1) swg(n) \tag{11}$$

The effective traction $tra(n)$ of a node is the amount of "useful" force applied to that node. Effective traction is the contrary of swinging: forces that contribute to the convergence. It is defined as an average:

$$tra_{(t)}(n) = \frac{|F_{(t)}(n) + F_{(t-1)}(n)|}{2} \tag{12}$$

If a node keeps its course, then $tra(n) = F(n)$. If it goes back to its previous position (a perfect swinging) then $tra(n) = 0$.

The global effective traction $tra(G)$ is the weighted sum of effective tractions of nodes:

$$tra(G) = \sum_n (deg(n) + 1) tra(n) \tag{13}$$

The global speed $s(G)$ keeps the global swinging $swg(G)$ under a certain ratio τ of the global effective traction $tra(G)$ and is defined as follows:

$$s(G) = \tau \frac{tra(G)}{swg(G)} \tag{14}$$

The ratio τ represents the tolerance to swinging and is set by the user.

NB: During our tests we observed that an excessive rise of the global speed could have a negative impact. That's why we limited the increase of global speed $s_{(t)}(G)$ to 50% of the previous step $s_{(t-1)}(G)$.

D. Details on this strategy

Our initial idea was to get the optimal speed under every circumstance, and avoid a "speed" setting that users do not manage easily. We did not succeed, and we still have it under the name of "Tolerance". But this perspective explains the strategy we adopted. An optimal global speed is a similar idea to simulated annealing (CITATION NEEDED). But is not the same because

we have to prevent the freezing of the network. Simulated annealing is to find the right way to "cool down" the network, to reduce its speed so that it convergences more efficiently. Intuitively, the network can go faster in the beginning, since it is about its general shape, and needs more precision in the end, for the details. In more scientific terms, simulated annealing is about shortening the steps in the end for refining the spatialization, and stopping it. Yifan Hu [9] uses this technique in the end, during a refining phase. However he remarks that "for an application of a force-directed algorithm from a random initial layout, an adaptive step length update scheme is more successful in escaping from local minimums. [...] Step length can increase as well as reduce, depending on the progress made". Yifan Hu remarks that out of the refining phase, an adaptive speed is about "heating" as well as "cooling", because escaping a local minimum may need a speed up. This applies to us, since there is no refining phase in a continuous algorithm. Yifan Hu evaluates the convergence of the network and adapts its speed in consequence. Our "global speed" plays the same role, but we evaluate the convergence differently. Yifan Hu relies of the variation of global energy, while we rely on the regularity of applied forces (effective traction). The reason is that we have also a local speed optimization, like GEM [**?**], and that we need some homogeneity between the global speed and the local speed.

We explained that the local speed aims at providing more precision to nodes that fail at converging. Like GEM, we try to minimize swinging (oscillations). The local speed can slow the nodes down, but cannot speed them up. Even if the node deserves more speed, it is limited by the global speed. The global speed determines the global movement, it is an "adaptive heating" rather than an "adaptive cooling". It is as high as possible, in the limit of a certain amount of global swinging determined by the "Tolerance" setting. The local speed regulates the swinging while the global speed regulates the convergence. But the regulation of convergence is indirect, since we just compare the global effective traction with the swinging. We rely here on the assumption that oscillation denotes a lack of convergence. This assumption is reasonable, even if we know that it is false under certain circumstances (the swinging of a node propagates to its neighbors). GEM also relies on this assumption.

E. Comparison with other algorithms

We compare here ForceAtlas2 to the recent algorithm of Yifan Hu and to the old and classic layout of Fruchterman and Rheingold. We did not compare to OpenOrd, which is very efficient,

but is not a continuous layout. We did not compare to GEM because it is not implemented in Gephi (that we used as a benchmarking tool). We also pictured the LinLog variant of ForceAtlas2 because we had no other implementation (they are very close).

We analyze how each layout impacts the quality of a network. We use the "$\mathrm{normalized}^{\mathrm{endv}}$ atedge length" measure proposed by Noack [11] to evaluate the quality. It measures that edges are the shortest possible, being normalized to the size and density of the graph. The lower it is, the better is the spatialization. Noack's LinLog layout is designed to minimize this measure. This measure is formally, for a set of positions p where N are the nodes and E the edges:

$$Q(p) = \frac{\sum_{\{n_1;n_2\}\in E} distance(p(n_1), p(n_2))}{|E|} / \frac{\sum_{\{n_1;n_2\}\in N^2} distance(p(n_1), p(n_2))}{|N^2|} \tag{15}$$

At each step of the tested algorithm, we compute a quality for current positions. All the layouts, by definition, improve the quality of the spatialization (the quality decreases). We compare the best quality they reach, and how much steps are needed (performance). The figure 8 uses this protocol to picture the impact of the adaptive local speed feature. We compare the actual implementation to variants where we fixed the local speed at different values. We observe different scenarios. If the speed is too low (0.001), the convergence is slow and we do not have enough steps to see the reached quality. If the speed is too high (0.1) the convergence is quick in the beginning but the quality stagnates early (because of oscillations). A medium value of 0.01 has a good convergence and a good final quality, but the adaptive local speed achieves even better on convergence as well as on final quality.

As we see in figures 8 and 9, the spatialization process has 3 different stages. In the early iterations, quality improves slowly (this stage is sometimes very short). Then in the following iterations, the spatialization reaches its best rate. Finally the quality stops improving and slowly reaches its final state. The second stage characterizes the balance between performance and quality. We can detect this stage thanks to the inflexion point. We rely on the tangent at this point to measure the performance and quality of the curve. The figure 9 illustrates this method. To detect a global inflexion, we smooth the curve (we average from -10 to +10 iterations), then we detect the inflexion point (minimal tangent) (fig 9.B). The inflexion point defines the moment where the quality improves the most. The tangent defines the best quality that can be obtained at a single step. Our performance indicator is the iteration where the best quality (0) would be

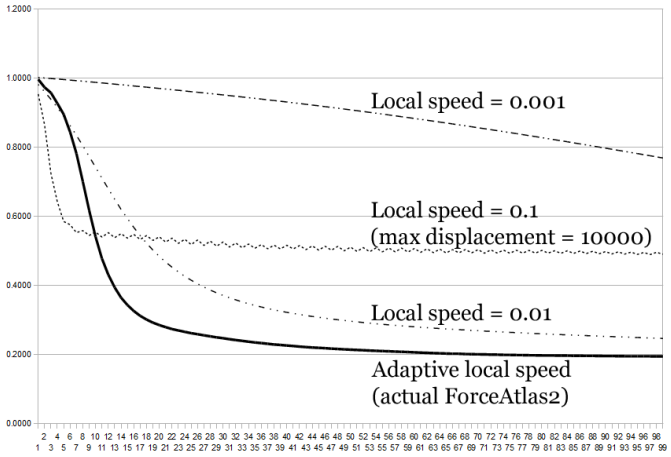

Fig. 8. Evolution of the quality of the ForceAtlas2 layout at each iteration (the lower the better), with different values of the local speed. The adaptive local speed achieves the best compromise between performance and quality.

achieved at the best rate. It is the point where the tangent crosses the X axis (fig 9.C). We call this step the "ideal convergence". It corresponds to the number of steps that would be needed to reach an ideal quality at the best possible rate. It does not depend on the best quality reached by the algorithm. The actual quality reached at this moment is called "short-term quality", since it corresponds to the quality you obtain just after the most efficient phase of the layout. To represent what you get if you have time, we measure the quality reached after 750 iterations ("long-term quality").

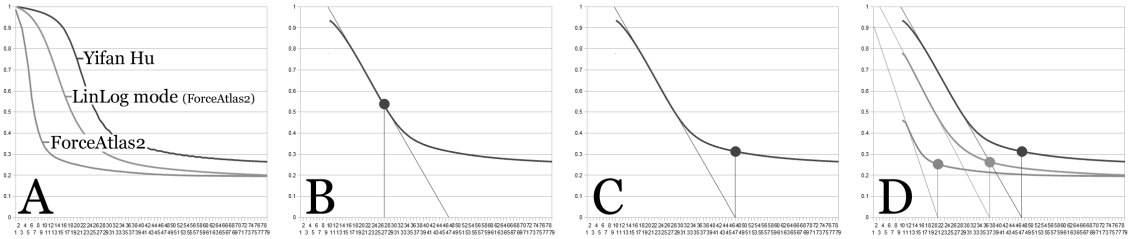

Fig. 9. Evolution of the quality of 3 different algorithms during spatialization (A). The tangent at the inflexion point (on smoothed curves) (B) defines the most efficient rate of convergence. When this tangent crosses the X axis, the actual quality reached on the curve (C) defines what you get if you stop the algorithm after its most efficient phase. This points allows to compare quality as well as performance between algorithms (D). The earlier it happens, the most efficient the algorithm.

We applied this protocol to 10 different networks ranging from 34 to 23000 nodes. The result

is presented as charts in figures 9, 10 and 11. The figures are available in an appendix, with details on the datasets used. Note that the settings are adjusted so that spatialized networks always have the same size. For ForceAtlas2 and Fruchterman Rheingold, the gravity was set to zero. The speed was set to 20 for Fruchterman Rheingold (the best empirical value for these datasets). These indicators measure performance and quality:

- "Iterations performance" measures how much iterations the layout would need to reach the ideal quality (0) if it was progressing at its best rate.
- "Short-term quality" is the quality reached just after the most efficient phase of the layout. This moment is the one used to measure iterations performance.
- "Long-term quality" is measured at the 750th iteration. This value was empirically chosen so that all layouts have time to fully improve quality.

ForceAtlas has both a better "iterations performance" and a better "short-term quality" than Yifan Hu and Fruchterman Rheingold algorithms for 7 datasets over 10. It has a better "long-term quality" for all but one dataset with default settings, and achieves an even better "long-term quality" in all cases with the LinLog mode enabled. We empirically observed that ForceAtlas2 is at its best with strongly clustered networks.

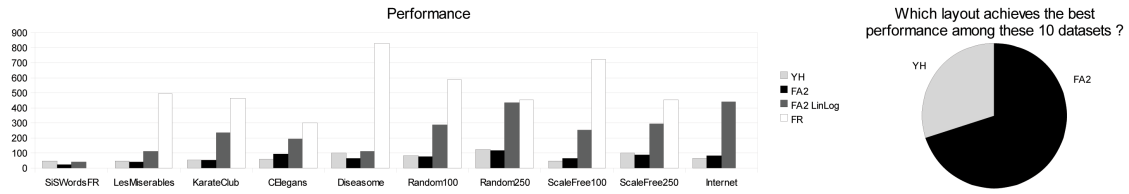

Fig. 10. Performance of the layouts on various datasets: Yifan Hu (YH), ForceAtlas2, by default (FA2) and with LinLog mode enabled (FA2 LinLog) and Fruchterman Rheingold (FR). The lower the better. ForceAtlas2 needs most of the time a little less iterations than Yifan Hu.

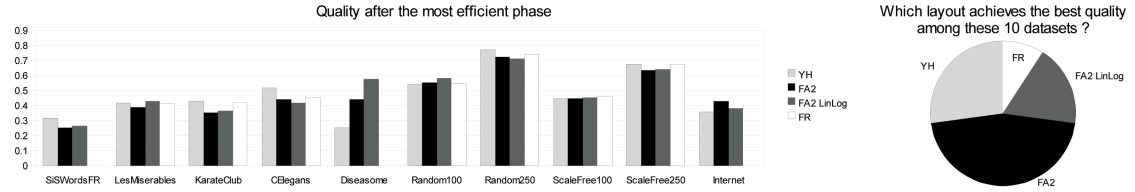

Fig. 11. Short-term quality of the layouts on the datasets. The lower the better. ForceAtlas2 has often the best quality.

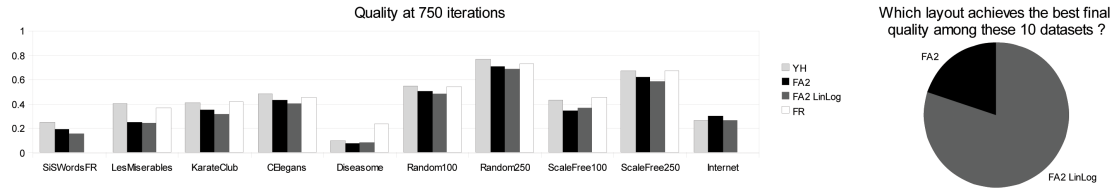

Fig. 12. Long-term quality of the same layouts for the same datasets. The lower the better. ForceAtlas2 in LinLog mode is clearly the best in the long-term. Noack's LinLog was designed to optimize this quality measure. Even in normal mode, FA2 has a better long-term quality than YH.

ForceAtlas2 is compares to Yifan Hu in terms of performance and quality. The LinLog option converges slowly but provides a better quality.

V. Discussion: designing a generic, continuous layout

The visualization of a network involves design choices. We think users have to be aware of their consequences. The strategy we adopt in Gephi is to let users see in real time the consequences of their choices, learning from their trials and errors. Interaction, we believe, is the key to understanding. While developing the Gephi user experience, we strongly desired a "live" spatialization process. Hiding it may lead users to believe that the placement is unique or optimal. Non-expert users need to observe the spatialization process and even to interact with it. Manipulating a graph while it spatializes helps to understand the difference between a graph layout and a Cartesian projection. The effect of the settings can be observed and understood. It helps to figure out that spatialization is a construction that involves the responsibility of the user.

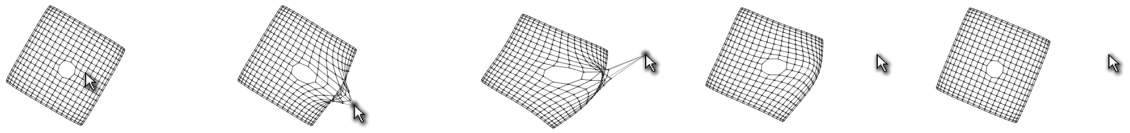

Fig. 13. A sample network manually stretched and released while ForceAtlas2 is running. The goal is to experience what is "the equilibrium state".

Users can act on the network by changing the ranking of the nodes, or filtering nodes and edges, even creating new entities. ForceAtlas2 passes on modifications in real time, re-computing forces and continuously updating the placement of nodes. It is possible to "play" with the network.

Since it is intuitive for users, developers can integrate other features on top. For instance we integrate the visualization of a dynamic network just as a particular case of dynamic filtering: the real-time layout updates the structure according to the specified time span[5].

Data monitoring is a basic use case of network visualization. With Gephi we intend to foster advanced uses: data exploration and map making. These uses are more demanding. Exploring the data may require to search for an adapted layout: a satisfying algorithm with satisfying settings. We cannot discuss here how and why some algorithms are better choices for certain networks, but we can give basic examples cases. ForceAtlas2 is not adapted to networks bigger than 100000 nodes, unless we let it work during several hours. On the contrary, OpenOrd [8] is not adapted to networks of less than 100 nodes, because its side effects are too visible at this scale. Certain algorithms are more adapted to certain sizes, as well as certain densities, or certain community structure. Certain energy models provide a better depiction of certain network types. Alternative energy models are relevant features to diversify the algorithm's applications. The *LinLog*, *edge weight* and *gravity* settings are such options, fostering a better exploration of the structure. Map making requires different features. Its issue is to make the network fit in a limited graphic space. *Scaling* and *gravity* settings help users to bring have a more compact network. The *overlapping prevention* provides more readability to the result. At last, some features are implemented just for performance, like Barnes Hut's optimization (*approximate repulsion*) and adaptive speeds. But even in this case we try to provide explicit settings to the user (*Tolerance (speed)*).

Integrating varied features obliges us to adapt some of them. We bring homogeneity in the different forces we implement. First we weight the nodes by degree plus one instead of just the degree (we cannot ignore nodes of degree 0). Secondly we adapt the *gravity* energy model to the repulsion force to limit its side effects. When repulsion is weighted in a certain way (for instance with the *dissuade hubs* setting) then the gravity is weighted the same way. We also normalized certain features to provide a smoother user experience. When *dissuade hubs* is activated, we compute a normalization to ensure that the total energy with the alternative forces is the same to the reference forces. Thanks to this trick, the network keeps a comparable spreading in the

[5]for an example see the dynamic visualization of a Twitter conversation, http://gephi.org/2011/the-egyptian-revolution-on-twitter

graphic space. Not that the *LinLog* energy model does not benefit from such a normalization, so you have to adjust the scaling when you activate it.

VI. Conclusion

As more and more people deal with relational data, network visualization assumes a key importance. ForceAtlas2 is our practical contribution to network sciences. It is not based on a new conception of force-directed layouts and it implements many features from other famous layouts [9] [6] [**?**]. However, by its design and features, it aims at providing a generic and intuitive way to spatialize networks. Its implementation of adaptive local and global speeds brings good performances for network of less than 100000 nodes, while keeping it a *continuous* layout (no phases, no auto-stop), fitting to Gephi user experience. Its code is published in Java as a part of Gephi source code[6].

References

[1] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: an open source software for exploring and manipulating networks," in *International AAAI Conference on Weblogs and Social Media*. Association for the Advancement of Artificial Intelligence, 2009.

[2] D. Diminescu, "The connected migrant: an epistemological manifesto," *Social Science Information*, vol. 47, no. 4, pp. 565–579, 2008.

[3] A. Noack, "Modularity clustering is force-directed layout," *Physical Review E*, vol. 79, no. 2, 2009.

[4] M. E. J. Newman, "Analysis of weighted networks," 2004, arxiv:cond-mat/0407503.

[5] ——, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[6] A. Noack, "Energy models for graph clustering," *J. Graph Algorithms Appl.*, vol. 11, no. 2, pp. 453–480, 2007.

[7] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Softw: Pract. Exper.*, vol. 21, no. 11, pp. 1129–1164, Nov. 1991.

[8] S. Martin, W. M. Brown, R. Klavans, and K. W. Boyack, "OpenOrd: an open-source toolbox for large graph layout," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, ser. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 7868, Jan. 2011.

[9] Y. F. Hu, "Efficient and high quality force-directed graph drawing," *The Mathematica Journal*, vol. 10, pp. 37–71, 2005.

[10] P. Eades, "A heuristic for graph drawing," *Congressus Numerantium*, vol. 42, pp. 149–160, 1984.

[11] A. Noack, "Unified quality measures for clusterings, layouts, and orderings of graphs, and their application as software design criteria," PhD thesis, Brandenburg University of Technology, Cottbus, Germany, 2007.

[12] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, vol. 46, pp. 604–632, September 1999.

[6]https://github.com/gephi/gephi/tree/master/LayoutPlugin/src/org/gephi/layout/plugin/forceAtlas2

[13] J. Barnes and P. Hut, “A hierarchical o(n log n) force-calculation algorithm,” *Nature*, vol. 324, no. 6096, pp. 446–449, Dec. 1986.