2013 17th International Conference on Information Visualisation

One Graph, Multiple Drawings

M. Nadal, G. Melanon
CNRS UMR 5800 LaBRI
Talence, France
Email: nadal@labri.fr, melancon@labri.fr

***Abstract*—Being able to produce a wide variety of layouts for a same graphs may prove useful when users have no preferred visual encoding for their data. The first contribution of this paper is a enhanced force-directed layout capable of producing different layouts of a same graph. We turn a well known force-directed algorithm (GEM) into a highly parametrizable layout and control it from a genetic algorithm framework.**

The genetic algorithm allows to efficiently explore the parameter space of this highly parametrisable layout. The search process relies on the capability of the system to evaluate the similarity between two drawings. The second contribution of this paper is a similarity metric used as a fitness function for the genetic algorithm. Its main features are its computational cost and its insensitivity to planar homotheties.

***Keywords*-graph drawing; multiple drawings; similarity**

I. Introduction

The graph drawing community has introduced a series of useful aesthetics for drawing graphs [1], [2], most of which can be translated into optimization goals algorithms try to reach. Many graphs – those with no special properties such as begin tree-like, acyclic or planar – however do not call for any specific aesthetics; except those a user may have in mind because he/she has specific knowledge obtained from the data, or simply because of individual preferences. There is however no one algorithm that may serve each user's specific needs and/or graphical preferences.

As a consequence, it may be necessary to offer different drawings of a same graph to satisfy all users. Different approaches may be followed to solve this problem. A possible strategy is to run different algorithms on a same graph to produce different layouts. However, this option presents two main drawbacks. A strong expertise on graph drawing is required to master a wide variety of algorithms, and there is currently no software offering an exhaustive panel of layout algorithms. The second drawback is the impossibility to systematically produce a hybrid layout of a graph combining different algorithms. Producing variations of a layout, or applying variations on subparts of a same graph using parameter tuning may also be tedious. According to our own experience, we may even have no guarantee that the resulting layout will indeed differ from the use of default parameters.

This paper presents an approach to generate different drawings of a same graph using a unique algorithm. We designed a modified version of a force-directed layout, turned into a highly parametrizable algorithm. This modified algorithm is combined with a genetic algorithm (GA) to automate the generation of a parameter sets controlling the behavior of the force-directed layout. The whole framework can be steered by the user to influence the behavior and results offered by the genetic algorithm. The first part of the paper reviews past approaches who have partly addressed this problem (section I-A). Section II presents our algorithm and its different features. A similarity metric, used to compare different drawings of a same graph is described in section IV. Results are presented and discussed in section V.

A. Related work

Since the seminal work of Tutte [3], the Graph Drawing community has developed a rich panel of algorithms to layout different classes of graphs [1], [2]. Force-directed layouts (sometimes also called spring embedders) form an important part of the Graph Drawing body of algorithms, mainly because they can be used to draw graphs with no specific properties. The work by Fruchterman-Reingold [4] and by Kamada and Kawai [5] are part of the most well known and largely used force-directed algorithms. The GEM algorithm published later by Fricke [6] proposed improvements by introducing additional parameters, and incidentally had a greater influence on our work. Several other improvements were proposed, mainly tackling scalability issues [7], [8], [9] or relying on multi-level approaches [10], [11], [12] sometimes exploiting hardware [13], [14].

In his 2011 keynote talk, Brandes [15] underlined the fact that most force-directed layouts do not succeed at delivering insight on the underlying data. Brandes' warning should be seriously considered, especially when visualization is targeted at a novice audience. In our opinion, part of the problem pointed at by Brandes comes from the inability of any given force-directed model to adapt to the topology of a graph. Some authors have produced higher level algorithms pre-processing a graph trying to detect topological features, then applying different layout algorithms to different parts of the graph [12]. When no feature can be found, these approaches unfortunately are as unproductive as any force-directed layout. Other approaches try to improve the overall quality of the layout by focusing on the initial placement of

1550-6037/13 $26.00 © 2013 IEEE
DOI 10.1109/IV.2013.55

CPS Conference Publishing Services

nodes [16], with no guarantee whatsoever of being able to vary the final output.

The work by Biedl *et al.* [17] is closer in spirit to our approach. Starting from the assumption that novice users cannot specify their needs in terms of graph layout, their method consists in generating different drawings for a same graph, from which the user can choose. The algorithm can not however deal but with small graphs, and is also limited in terms of diversity of layouts because restrictions are inherited from the GLIDE system they use [18].

Previous attempts at using genetic algorithms to draw graphs have been made. Eloranta *et al.* [19] used a GA to draw graphs on a grid, while Barreto *et al.* [20] considered layouts with coordinates in $\mathbb{R}^d$. In both approaches, the genomes stored the node coordinates, so the aim of the GA was to produce a layout of a graph by selecting candidate genes (node coordinates) and mixing them. These approaches are not totally satisfying, because initial genomes (positions) are randomly assigned, and because the GA may only produce a layout by combining existing node coordinates without reference to any graph property. The path we follow is totally different. In our GA, genomes do not store node positions; they store parameters that guide the behavior of a layout algorithm.

II. A HIGHLY PARAMETRIZABLE GEM ALGORITHM

This section presents the algorithm we designed in order to obtain a wide variety of layouts for a same graph. The basic ingredient of our approach is to highly parametrize an existing force-directed layout algorithm. The management of this large set of parameters is taken in charge by using a genetic algorithm (GA). Using the GA ensures a thorough search of the parameter space.

The force-directed layout algorithm GEM [6] adds several features to the classical force directed paradigm. As in most force-directed algorithm, a temperature is used to speed-up convergence of the computations. Each node is assigned an initial individual temperature, so nodes stabilize as they cool down. The algorithm also tries to detect whether a change in the layout is due to a rotation and/or oscillation to allow a speed-up of stabilization. Other parameters are used to tune the algorithm. Each vertex has a *mass* so heavier nodes help the layout to stabilize. As with most force-directed models, GEM uses *coefficients* to tune the effect of attractive and repulsive forces. GEM also uses the *barycenter* of the layout to avoid components to be pushed away from each other, again using a coefficient to dose the attractive effect of the barycenter. A *random force* (and a coefficient) is also applied locally so the algorithm may get out of local minima situations. Finally, adjacent nodes try to reach an *ideal edge length*.

Apart from the vertex temperature and mass, all parameters are global and defined at the graph level. Fig. 2 (1a, 2a) shows different drawings obtained with the basic version of GEM (using default parameters).

A. Vertex parametrization

In order to reach a maximum of possible algorithm behaviors, leading to different outputs, we individualize all parameters and define their values at the node level. That is, nodes get assigned coefficients for all forces (attractive, repulsive, barycenter, etc.) independently. When dealing with larger graphs, the number of parameters makes it impossible for a user to set them manually. Indeed, using 6 parameters per vertex requires to deal, set and manage 300 parameters for a graph with 50 vertices only.

Moreover, our experience shows that parameter tuning is tricky. A slight modification of a parameter may produce no change in the resulting drawing; conversely, more radical changes may simply impair the algorithm from finding an acceptable layout. Hence, we use a genetic algorithm to (i) automate the management of parameter values and (ii) combine them and efficiently search the space of all possible combinations. The search strategy will be discussed later.

B. Exploring the parameter space

Parameter values are managed using a GA and stored as genomes. The algorithm thus deals with a population of genomes, and supervises the evolution of future genomes. A gene corresponds to a (key/value) pair together with a third information indicating the domain within which the parameter may vary (a bounded interval $\subset \mathbb{R}$); a genome thus corresponds to a list of (key/value) pairs. At each generation, the population of genomes is evaluated – each genome is assigned a *score*. Best genomes are selected and form the next generation of the population. Two main genetic operators are used: the crossover and the mutation. A crossover takes two parent genomes as input and produces a child or two children genomes. The first child inherits one half of all genes from each parent. In the case where two children are produced, the second child inherits of genes unused in the production of the first child. A mutation consists in randomly modifying the values of one or several genes of a genome.

We shall limit the discussion of the GA we actually designed to its fundamental features. A complete presentation of the genetic algorithm and discussion of its features is out of the scope of this paper [21]. The genetic algorithm we have implemented is highly configurable and allows a fine-grain control over population evolution.

III. GRAPH EVALUATION

This section describes the main features of the GA we designed and focuses on the evaluation mechanism guiding the exploration of the genome space.

In our case, the evaluation is a two-step process. We first select genomes in the population and map their (key/value)

pairs to nodes in the graph. The layout algorithm GEM is then run using this set of parameters. The second part of the evaluation consists in computing a fitness function for each vertex of the graph taking the layout coordinates into account.

The score assigned to a vertex can be accomplished in two different manners discussed in the next sections. A first (and most common) way is to compute a score based on a set of *graphical metrics* (section III-A). Alternatively, we may use a *similarity metric* used to compare early layouts to target layout, in the learning phases of the GA.

This latter type of evaluation is called "*auto-evaluation*". Nodes are evaluated by comparing them to previously drawn and *similar* vertices. Genomes of these previously drawn vertices are stored in a database to enhance the learning capabilities of the GA.

A. Graphical metrics

The score of a node is computed based on how well a node has been positioned with respect to different node properties. We list here the metrics we used in order to evaluate whether the GA was able to find a good position for a node. Some of these metrics use individual node metrics, other use metric that are global to the whole graph.

Denote the graph as $G = (V, E)$. *Distance conservation* evaluates, for a given node $u \in V$, how close Euclidean distance is to graph distance. That is, the metric is obtained by calculating the standard deviation of all ratios $d(u, v)/d_G(u, v)$ for all pairs of nodes (u, v), with $v \in V$.

Average edge length evaluates, for a given node $u \in V$, how close the Euclidean length of edges is to their individual ideal length. That is, the metric is obtained by calculating the standard deviation of all ratios $d(u, v)/d_G(u, v)$ for all pairs of *neighbor* nodes $(u, v), v \in N_G(u)$.

Number of edge crossings counts the number of edges that cross in the Euclidean drawing of the graph. The *graphical density* of a node u counts the node in a circle around u with a radius proportional to $\frac{layout\_size}{\sqrt{n}}$.

The fitness function is a non-linear combination of all these metrics. This combination can be customized in order to change the behavior of the algorithm, searching different parts of the parameter space, producing different types of layouts.

1) Building a knowledge base for the GA: Defining a good fitness function is however difficult, particularly for novice users. This also relates to the impossibility of defining in an absolute manner what a "good" drawing of a graph is with no reference to the graph being drawn – ultimately requiring to adapt the fitness function to each graph.

A solution to deal with this problem is to build a knowledge database collecting "good" and "bad" drawings of a same graph. To this end, genomes are stored together with information describing the *context* in which they are being evaluated. The context of a node is formed of different measures related to the topology of its neighborhood: its degree, clustering coefficient, betweenness centrality, and eccentricity (see [22] for definitions of these metrics). The *behavior* of a node in a given context is provided by the values of all graphical metrics enumerated in the precious section. In other words, the database stores genomes associated with a context together with the graphical metrics, in a sense describing how they behave in this particular context.

During the learning phase, because the knowledge base needs to be bootstrapped, a target layout is given along with the graph. The fitness function then computes the similarity between the laid out graph (using the genomes being evaluated) and the target layout. In other words, the similarity computed from graph metrics help decide whether a node has been assigned a good position with respect to the target layout.

With node context and behavior, it then becomes possible to select all genomes that behaved well in a given context bringing them into the population considered by the GA. This allows a speed-up of the algorithm by starting with a population taken from the database rather than initializing the GA with a random population.

IV. Graph similarity

As we say in section III-A1, we use a similarity metric as fitness function during database bootstrapping. We give good layout for a graph, and the GA have to copy it. In order to evaluate each genome, we need to evaluate how each vertex is close to its target position. As this metric is computed thousand times, it needs to have a low computational cost. The second constraint is that the similarity has to be insensible to any planar homothety, as translation, rotation, scale or symmetry which could be induced by GEM.

This task is very close to conducting a Procrustes analysis [23], trying to find a best superimposition minimizing the overall distance between two point clouds applying a combination of rotation, translation and rescale.

Biedl *et al.* [17] face a similar problem as they need to determine how much two drawings of a graph are different (they also group layouts into classes of more similar drawings). They compute a one-to-one matching of nodes based on distances between neighbor nodes. Once a matching is found, they compute the distance between the two drawings as the sum of distances between all pairs of vertices.

A. Triangle-based similarity

We now describe a method we designed to achieve both efficiency and accuracy in computing graph similarity. Nodes are considered as part of triangles; a triangle is a subset of three pairwise nodes (adjacent or not). Two triangles are considered similar when their angles match.

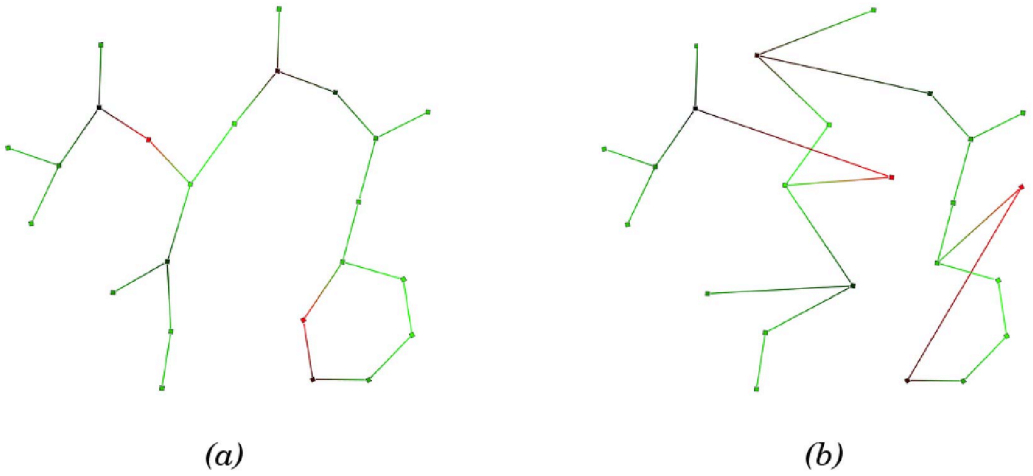

Figure 1. Two layouts of a same graph and nodewise similarity between them (red/green for low/high node similarity).

This criteria is handy because it can be measured by testing whether sidewise ratios are similar:

$$\frac{a}{a'} \sim \frac{b}{b'} \sim \frac{c}{c'}$$

where a, b, c and a', b', c' respectively denote lengths of the sides of the two considered triangles. Observe that triangle similarity is invariant under planar homothety (i.e. translation, rotation, scaling or symmetry) because these transformations preserve ratios of side lengths. The rationale underlying this similarity criteria is that layout similarity follows when we have triangle similarity for all possible triangles.

A naive approach to accomplish triangle similarity is to compute ratios for all triangles in the layout. The computational cost of this naive approach would however compare to a Procrustes analysis since there potentially are n^3 triangles to inspect. A more efficient and less costly solution is possible. Assuming the two layouts to be similar, all pairwise distances $d_L(u,v)$, $d_{L'}(u,v)$ between nodes should match in both layouts L, L'. As a consequence, the ratios $r(u,v) = \frac{d_L(u,v)}{d_{L'}(u,v)}$ should have null standard deviation. Let $\sigma_G^2 = \frac{1}{|V|-1} \sum_{\{v\} \in V, v \neq u} (r(u,v) - \overline{r})^2$ denote the variance of the ratios $r(u,v)$. We define a similarity metric as

$$s_G(u) = \sqrt{\sigma_G^2}/\overline{r}$$

where $\overline{r}$ the average value of $r(u,v)$ for all $v \in V$. A main feature of this metric is its invariance with respect to any planar homothety, as all length ratios are preserved. As we need to compute one similarity per node (and highlight most displaced vertices), we compute a partial standard deviation of the ratio $r(u,v)$ with u fixed and $v \in V, v \neq u$ as the similarity for the node u.

Fig. 1 shows two layouts of a same graph. Green bright nodes correspond to highest similarity, while bright red nodes are most dissimilar – observe the two red nodes with incident edges forming spikes on the right image.

1) Local similarity: Similarity can be controlled and somehow kept local by considering weights $d_G(u,v)$ to compute a weighted average $r_u = \frac{1}{|V|-1} \sum_{\{v\} \in V, v \neq u} d_G(u,v)^k r(u,v)$ (and compute σ_G^2 accordingly).

Positive and larger (integer) values of k forces the distances to farther nodes (from u) to be taken into account, while negative values of k somehow restricts the metric to nodes closer to u. Observe that this localized version of the metric assumes the graph to be connected.

However, this modified version requires to compute all pairwise node distances in the graph, prior to metric computation. Its complexity being in $O(n^2)$, the global time complexity of the metric is maintained.

V. Results

This section presents results obtained on two different graphs (Fig. 2). (Top row) The first target graph we used models the word "HELLO" using nodes and edges – the target nodes consists of horizontally aligned square-shaped letters. Obviously, GEM totally ignores the graph models a word and produces an abstract layout with no link whatsoever with the word "HELLO" (1a). This toy example shows just how capable the GA, guiding a highly parametrized force-directed algorithm to reproduce such a constrained drawing (1b).

The second graph models a metabolic pathway (biomolecular reactions), the plastids of Arabidpsis. GEM, as any

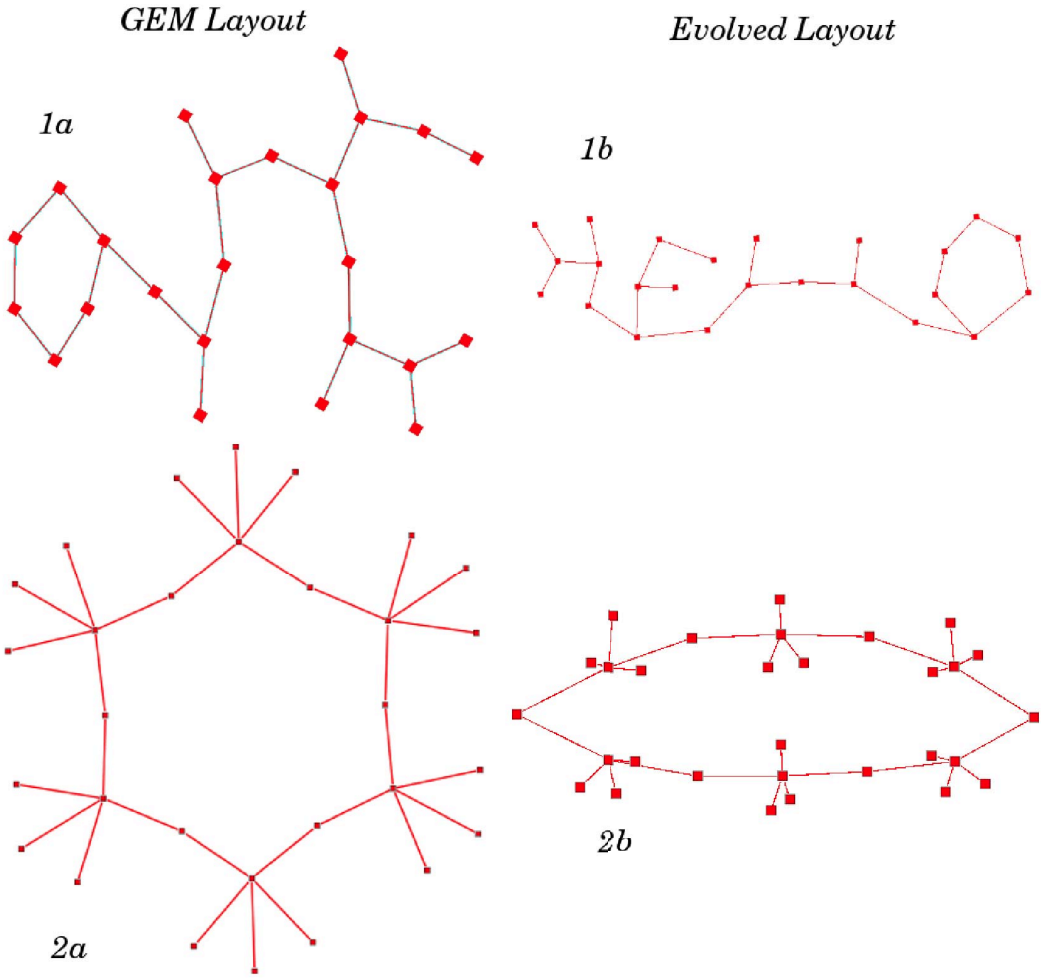

Figure 2. Comparisons between layouts obtained with the original GEM algorithm (left) and our GA-enhanced version (right) for two different graphs.

force-directed algorithm, tends to layout cycles as circles and spreads out nodes away from this central component (2a). This type of layout is informative and sometimes used in the literature. It can indeed be obtained form the GA with a parameter set mimicking what GEM would do using its default parameters. However, we may use a target layout for this graph positioning the backbone cycle on the periphery of a rectangle. Because the GA is capable of searching all combinations of node parameters, it also can produce a layout constraining the backbone cycle to an elliptical shape (2b). This example shows the capability of the GA of producing a wide variety of layouts for a same graph.

It is also worth noting that the smaller star-shaped components have been positioned in similar manners. This underlines the effect of the similarity metric in the choice of genomes and ultimately in the final layout.

VI. Conclusion & Future Work

This paper presented two main contributions. We have first designed an enhanced version of a force-directed algorithm tuned into a highly parametrizable layout. Using a genetic algorithm to guide the search of parameter sets, we are able to produce a wide variety of layouts of a same graph. This feature is certainly useful when users have no predefined visual encoding for their data. It may also be useful to provide users with a panel of possible layouts from which to choose.

Our second contribution is a similarity metric used as a fitness function for the genetic algorithm. This metric is used to compare two layouts of a same graph, based on triangle congruence. The main advantages of this method are its insensitivity to planar homotheties and its computational cost, lower than other methods inspired from Procrustes

analysis. This similarity metric can furthermore be tuned in order to act locally.

These two contributions are a step towards a more ambitious goals. We are currently working on different methods to automatically evaluate a graph drawing, based on data accumulated from past executions. We also plan to expand the GA framework and include curiosity guided exploration. Once such a system is built, we may expect novice users to simply load their data and select the most appropriate graph from a proposed selection. We could even and act on the learning process and make the system aware of the users preference to guide future layouts.

REFERENCES

[1] G. di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualisation of Graphs*. Prentice Hall, 1998.

[2] M. Kaufmann and D. Wagner, *Drawing Graphs, Methods and Models*, ser. LNCS. Springer, 2001, vol. 2025.

[3] W. Tutte, "How to draw a graph," *Proc. of the London Math. Soc.*, vol. 3, no. 13, pp. 743–768, 1963.

[4] T. Fruchterman and E. Reingold, "Graph drawing by force-directed placement," *Software Practice & Experience*, vol. 21, pp. 1129–1164, 1991.

[5] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Inform. Process. Letters*, vol. 31, pp. 7–15, 1989.

[6] A. Fricke, A. Ludwig, and H. Mehldau, "A fast adaptive layout algorithm for undirected graphs," in *Symposium on Graph Drawing GD '94*, ser. LNCS, vol. 894. Springer Verlag, 1995, pp. 389–403.

[7] S. Hachul and M. Jnger, *Drawing Large Graphs with a Potential-Field-Based Multilevel Algorithm*, ser. LNCS. Springer, 2005, vol. 3383, pp. 285–295.

[8] ——, "Large-graph layout algorithms at work: An experimental study," *J. Graph Algo. & App.*, vol. 11, no. 2, pp. 345–369, 2007.

[9] E. Yunis, R. Yokota, and A. Ahmadia, "Scalable force directed graph layout algorithms using fast multipole methods," in *ISPDC 2012*, 2012, pp. 180–187.

[10] Y. Koren and D. Harel, "Ace a fast multiscale graph algorithm," in *IEEE InfoVis*, 2001, pp. 137–144.

[11] C. Walshaw, "A multilevel algorithm for force-directed graph-drawing," *J. Graph Algo. & App.*, vol. 7, no. 3, pp. 253–285, 2003.

[12] D. Archambault, T. Munzner, and D. Auber, "Topolayout: Multi-level graph layout by topological features," *IEEE Trans. Vis. & Comp. Graph.*, vol. 13, no. 2, pp. 305–317, 2007.

[13] D. Auber and Y. Chiricota, "Improved efficiency of spring embedders: Taking advantage of gpu programming," in *Sixth IASTED – VIIP 2007*. IASTED, Acta Press, 2007, pp. 169–175.

[14] Y. Frishman and A. Tal, "Multi-level graph layout on the gpu," *IEEE Trans. Vis. & Comp. Graph.*, vol. 13, no. 6, pp. 1310–1317, 2007.

[15] U. Brandes, "Keynote address: Why everyone seems to be using spring embedders for network visualization, and should not," in *IEEE PacificVis*, 2011, pp. xii–xii.

[16] P. Gajer and S. G. Kobourov, "Grip: Graph drawing with intelligent placement," *J. Graph Algo. & App.*, vol. 6, no. 3, pp. 203–224, 2002.

[17] T. Biedl, J. Marks, K. Ryall, and S. Whitesides, *Graph Multidrawing: Finding Nice Drawings Without Defining Nice*, ser. LNCS. Springer Berlin Heidelberg, 1998, vol. 1547, pp. 347–355.

[18] K. Ryall, J. Marks, and S. Shieber, "An interactive constraint-based system for drawing graphs," in *10th UIST Conf.*, 1997, pp. 97–104.

[19] T. Eloranta and E. Mäkinen, "TimGA: A Genetic Algorithm for Drawing Undirected Graphs," *Computer*, vol. 9, no. 2, pp. 155–171, 2001.

[20] A. M. S. Barreto and H. Barbosa, "Graph layout using a genetic algorithm," *Proc Vol.1. Sixth Brazilian Symposium on Neural Networks*, pp. 179–184, 2000.

[21] M. Nadal and M. Guy, "Reusable genomes : Welcome to the green genetic algorithm world," CNRS UMR 5800 LaBRI, Tech. Rep. RR-1469-13, Mar. 2013, 8 pages. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00802500

[22] U. Brandes and T. Eriebach, *Network analysis Methodological foundations*. Springer, 2005, vol. 3418.

[23] U. Brandes and C. Pich, *An Experimental Study on Distance-Based Graph Drawing*, ser. LNCS. Springer Berlin Heidelberg, 2009, vol. 5417, pp. 218–229.