Dynamic graph drawing with a hybridized genetic algorithm

Bruno Pinaud, Pascale Kuntz, Rémi Lehn

Dynamic Graph Drawing with a Hybridized Genetic Algorithm

Bruno Pinaud, Pascale Kuntz, Rémi Lehn

Ecole polytechnique de l'université de Nantes
Rue Christian Pauc - BP 50609
44306 Nantes Cedex 3
France
{bruno.pinaud, pascale.kuntz, remi.lehn}@polytech.univ-nantes.fr

Abstract

Automatic graph drawing algorithms, especially those for hierarchical digraphs, have an important place in computer-aided design software or more generally in software programs where an efficient visualization tool for complex structure is required. In these cases, aesthetics plays a major role for generating readable and understandable layouts. Besides, in an interactive approach, the program must preserve the mental map of the user between time $t-1$ and t. In this paper we introduce a dynamic drawing procedure for hierarchical digraph drawing. It tends to minimize arc-crossing thanks to a hybridized genetic algorithm. The hybridization consists of a local optimization step based on averaging heuristics and two problem-based crossover operators. A stability constraint based on a similarity measure is used to preserve the likeness between the layouts at time $t-1$ and t. Computational experiments have been done with an adapted random graph generator to simulate the construction process of 90 graphs. They confirm that, because of the actual algorithm, the arc crossing number of the selected layout is close to the best layout found. We show that computation of the similarity measure tends to preserve the likeness between the two layouts.

1 Introduction

Graph drawing algorithms are now generally included in Computer-Aided Design (CAD) software. A major advantage of graphs is that they can be used both as theoretical models for inherent relations between handled components, and as efficient visualization tools. They make access to complex structures easier without getting bogged down in abstract concepts. Among the numerous graph representations [1], hierarchical digraphs, also called layered digraphs, have a specially important place in CAD. In those graphs, vertices are arranged on vertical (Figure 1) or horizontal layers, and arcs

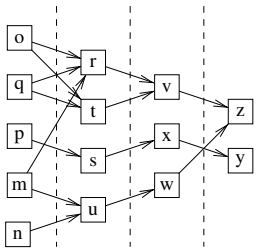


Figure 1: Example of a layered digraph with vertical layers

linking vertex pairs are represented by oriented line segments going in the same direction. These diagrams are particularly well-adapted to represent task decompositions and flow models in engineering planning.

When producing automatic layouts, the key words which guide the process are clarity and intelligibility. Although these notions call for subjective factors which closely depend on the audience, different criteria, modeled by combinatorial constraints, are commonly advocated by the information visualization community. Beside the physical constraints inherent to the medium (*e.g.* standard size sheets, computer screens), aesthetics plays a fundamental role. Their optimization aims at facilitating both readability and memorization of the information contained in the graphs. Frequently adopted aesthetic criteria include: minimization of the drawing area, minimization of the arc-crossing number, minimization of the sum of the arc lengths, *etc*. However, recent experiments have confirmed that the minimization of arc crossing is by far the most important criterion [2]. Hence, we here retain this one.

Minimizing arc crossing in a hierarchical layout could seem intuitively easier than the general problem of minimizing edge crossing on a plane since the choice of geometric coordinates for vertices is replaced by a choice of vertex ordering on each layer. Yet, it remains *NP*-complete [3]. Moreover, the explosion of the interactivity degree of CAD software has generated an additional constraint. Whatever the application field, numerous components and their associated links may be dynamically added or deleted during the first design stages. Consequently, the graph being visualized often changes over time. However, few algorithms take the interactivity with the user into account. If a modification is performed, the algorithm runs again and produces a new drawing which may be thoroughly different from the previous one. As noted by Papakostas *et al.* [4], "this is a waste of human resources to continually re-analyse the entire drawing and also of computation resources to re-compute the entire layout after each modification". Hence, the layout must not only remain readable over time, but the user's 'mental map' must also be preserved as much as possible [5].

In this paper, we tackle this multiobjective problem with a genetic algorithm (GA). GAs have already been applied to different static graph drawing problems with promising results (e.g. [6], [7], [8]). And, a recent statistical analysis of fitness landscapes associated with different local operators for the hierarchical layout problem has confirmed the presence of numerous local optima [9]. This advocates a highly stochastic

approach, like GAs, able to move away from neighbours of a local optimum 'late' in the search process. Moreover, due to their intrinsic parallelism, GAs are particularly well-adapted to easily take the additional stability constraint into account with a low cost. At each time t, a set of potentially good drawings according to the aesthetics is generated with a GA. Then a drawing is selected among them according to its resemblance to the previous one presented to the user at time $t-1$.

The rest of the paper is organized as follows. Section 2 briefly presents related works. Section 3 introduces a formal definition of the problem. Section 4 describes the optimization process based on a GA; we develop new specific problem operators and a local hybridization with the so-called averaging heuristics. Section 5 presents the experimental results and a comparison with Tabu Search.

2 Related works

Numerous deterministic heuristics for the static hierarchical layout problem follow the layer-by-layer sweep scheme: the vertices of each layer are reordered to reduce crossings while holding the vertex orderings on the other layers. Various strategies have been proposed for reordering (see [10] for a chronological review). The most commonly used are the sorting methods and the averaging heuristics which include the popular barycenter heuristics from Sugiyama [11], the median heuristics [3], and their variants. Sorting methods exchange vertices using crossing numbers in a way similar to classical sorts. Averaging heuristics are based on the idea that arc crossings tend to be minimized when connected vertices are placed facing each other. Consequently, vertices are arranged according to their neighbour average positions *e.g.* the arithmetic mean or the median.

These approaches are still certainly the most employed in software. However, it is well-known that their probability of getting down in local optima is far from being negligible. In an attempt to improve the quality of the results, metaheuristics have known a particular interest in graph drawing since the mid 90's. Tabu Search [10] and GRASP (Greedy Randomized Adaptive Search Procedure) [12] have been recently developed for the hierarchical drawing. Numerical comparisons on graphs with various densities (including those observed in real-life applications) show that both these search processes significantly outperform the deterministic layer-by-layer sweep approaches.

In comparison with static drawings, the literature on dynamic drawings is still in its infancy. In the hierarchical case, the first proposed heuristics are closely linked to the Sugiyama heuristics [13]. Also the multiobjective optimization has been tackled by different approaches: constraint propagation followed by constraint relaxation in order to cope with inconsistencies [14], or penalty assignment to vertex changes from $t-1$ to t and linear programming to compute the vertex coordinates on each layer [15].

3 Multiobjective problem description

In the following, we consider a hierarchical acyclic digraph $G = (V, A)$ with a vertex set V and an arc set A. Let $L = \{l_1, l_2, ..., l_K\}$ be a set of K layers and a given partition

$V_1, V_2, ..., V_K$ of V on L in classes with respectively n_1, n_2, ...,n_K vertices. The vertex assignment inside the layers is here supposed to be done. The vertex ordering on each layer l_k is defined by $\pi_k : l_k \rightarrow \{1, 2, ..., |l_k|\}$, where $\pi_k(u) = i$ means that the vertex $u \in l_k$ is on the i^{th} position on l_k. The drawing of G is completely defined by the set of the orderings $\Pi_G = \{\pi_1, \pi_2, ..., \pi_K\}$ on each layer l_i of L. The crossing number associated with Π_G is denoted by $c(\Pi_G)$. Note that, without loss of generality, we can restrict ourselves to proper graphs with arcs span equal to 1 (the span of an arc $(u, v) \in A$ with $u \in L_i$ and $v \in L_j$ is $j-i$). In practice, every arc with a span λ greater than 1 has to be replaced by a path of $\lambda-1$ dummy vertices on every consecutive layers.

When the drawing problem is included in a dynamical process, two constraints should be simultaneously satisfied: the drawing Π_{G_t} proposed at each step t must minimize $c(\Pi_{G_t})$ and must remain as close as possible to the previous $\Pi_{G_{t-1}}$ to preserve the user's mental map. To evaluate the likeness between different drawings, we resort to a measure $\delta\left(\Pi_{G_t}, \Pi_{G_{t-1}}\right)$ based on the number of inverted vertex pairs between $t-1$ and t:

$$\delta\left(\Pi_{G_t}, \Pi_{G_{t-1}}\right) = 1 - \frac{1}{K} \times \sum_{k=1}^{K} \frac{C_k(t-1,t)}{P_k(t-1,t)} \quad (1)$$

where $P_k(t-1,t)$ is the number of common vertex pairs for both drawings, and $C_k(t-1,t)$ is the number of vertex pairs which have been inverted between the drawing at $t-1$ and the one at t. If $\delta\left(\Pi_{G_t}, \Pi_{G_{t-1}}\right)$ is close to 1, then the number of inverted pairs is small and the drawing at $t-1$ is consequently preserved in the new drawing at t. More complex measures could be investigated in the future [16].

4 Graph optimization

The optimization process of the drawings is based on a hybridized genetic algorithm (HGA). And, the stability constraint (maximizing $\delta\left(\Pi_{G_t}, \Pi_{G_{t-1}}\right)$) is only taken into account in the selection of the solution in the last computed population. More precisely, at a given step t, HGA is applied to graph G_t, and among the best solutions computed by HGA, we select the one $\widehat{\Pi_{G_T}}$ which is the closest to the previous drawing $\widehat{\Pi_{G_{t-1}}}$. We first present the problem-based operators developed for the static drawing problem. Then we detail the integration of the dynamics.

4.1 The hybridized genetic algorithm

The HGA developed for minimizing $c(\Pi_G)$ follows the basic scheme with two specific characteristics. We have introduced problem-based crossovers and a local hybridization which significantly improves the computation time. Crossover is inspired from the following remark: the combination of two well-adapted drawings of sub-graphs can produce a better drawing (Figure 2). The combination may be applied between layers (inter-layer crossover) or inside each layer (intra-layer crossover).

The inter-layer crossover is a unique point crossover between layers. The intra-layer crossover aims at combining vertices of a same layer. However, combining blocks inside a layer meets a well-known difficulty for ordinal codings which is to define

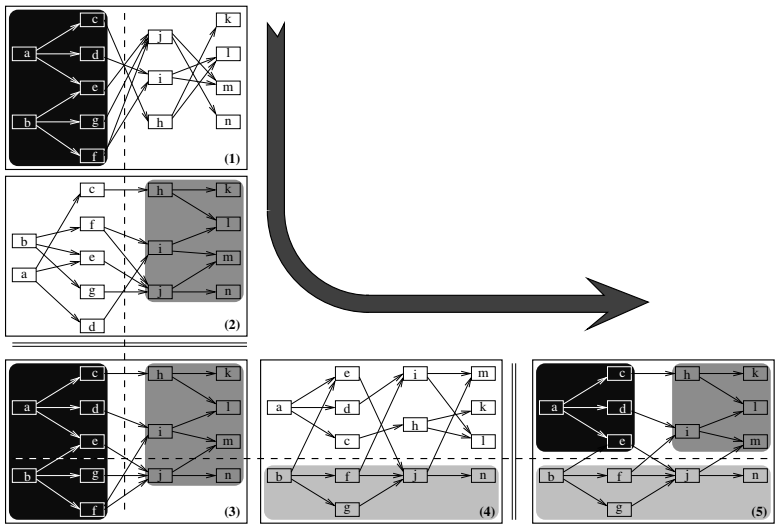


Figure 2: Improvement of an initial layout with successive sub-graph combinations. The inter-layer combination of (1) and (2) produces (3) and the intra-layer combination of (3) and (4) produces (5) which is a global optimum.

a crossover which guarantees a feasible solution [17]. The intra-layer crossover is a generalization of the Order Crossover 1 [17] for multi-permutations. A pivot whose random position is normalized by the layer cardinality is defined for each layer of the parents. For the first child, vertex positions of the first parent are retained above the pivot, and below it, positions of missing vertices are completed by those of the second parent according to the vertex ordering. And vice-versa for the second child.

The hybridization step is based on local search with the barycenter and median operator described in [9]. The selection is determined by a classical roulette wheel based on the arc crossing number as the fitness function. The mutation switches two randomly chosen vertices inside a layer. Each of these operators is applied with a given probability.

4.2 Integration of the stability constraint

While the graph is small ($|V| < 20$), we do not consider the dynamic constraint and the best solution computed by HGA at each step t is retained. Indeed, in this case, changes in the drawing aspect due to vertex and arc additions are very important and we cannot consider that a 'stable structure' exists to be preserved.

Algorithm 1 describes the integration of the stability constraint in the dynamic drawing process. At each step t, we select the drawing which maximizes the similarity among the 25% of the population composed of the best drawings produced by HGA. If the same similarity value is reached for different drawings, we keep the one which minimizes c.

Algorithm 1 Pseudo-code of the interactive drawing procedure

$t \leftarrow 0$;
while $|V_{G_t}| < 20$ {
  Apply HGA on Π_{G_t};
  Display the best drawing $\widehat{\Pi_{G_t}}$;
}
$t \leftarrow t + 1$;
while ¬termination criterion {
  Apply HGA on Π_{G_t};
  Put in *solution_array* the final population of HGA;
  For each drawing $\Pi^i_{G_t}$ of *solution_array* compute $\delta\left(\Pi^i_{G_t}, \widehat{\Pi_{G_{t-1}}}\right)$;
  Select the drawing $\widehat{\Pi_{G_t}}$ which minimizes $c\left(\Pi^i_{G_t}\right)$ and maximizes $\delta\left(\Pi^i_{G_t}, \widehat{\Pi_{G_{t-1}}}\right)$.
  Display $\widehat{\Pi_{G_t}}$;
  $t \leftarrow t + 1$;
}

5 Computational experiments

As far as we know, unlike the classical static drawing problem, there are no test case bases available for large scale numerical experimentations. Experiments are often done with human beings and focus on a panel of known responses. Although this approach is obviously necessary for the validation, it is faced with different intrinsic limitations: availability of the subjects, variations of their own behaviours and restrictions of the evaluated drawing number.

To obtain quantitative measures of validation on large graph sets, we have developed a graph generator adapted to the dynamic problem. We first briefly present it. Then, we describe the results obtained for the dynamic evolution of the drawings of 90 graphs. A comparison is given with Tabu Search (TS) which is to this day one of the best-known metaheuristics for the static problem [10].

5.1 Graph generator

The graph generator is composed of two major steps:

1. the initialization phase generates random connected graphs of small size. This phase reflects the beginning of numerous design processes where the graphs considered may be quite small and completed along the process.

2. at each step t, once a solution has been computed by the interactive drawing procedure for G_t, a new graph G_{t+1} is generated by adding randomly placed vertices, arcs and layers to G_t. For each generation, we ensure the preservation of the graph connectivity by adding at least one arc between the new vertices and the existing ones.

The generation process stops when the maximal vertex number (here set to 80) is reached or when $c(\Pi_{G_t})$ is greater than 500. The maximal number of layers is set to 15, the maximal number of vertices to add after each step has been set to 2. These sizes are typical of real-life applications where vertices are often represented by labelled boxes and can rarely exceed 70 or 80 on a standard size sheet for an exhaustive visualization.

This generator has been used to simulate the dynamic evolution of a set of 90 graphs which corresponds to a total of 1400 layouts. The mean density value defined by $d(G_t) = m/\sum_{k=2}^{h} |l_{k-1}| \times |l_k|$ is 0.65 (std. dev. 0.05).

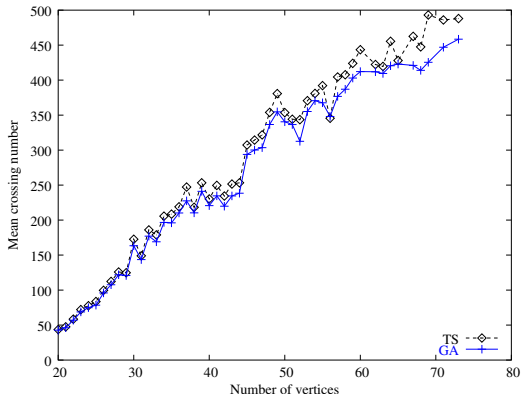
5.2 Results

In computational experiments, we have chosen the following genetic operator probabilities: 0.05 for mutation and 0.2 for each crossover. The population size has been fixed to 100. Preliminary experiments have shown that larger populations give sightly better results, but this choice is a compromise between the quality and the computational cost. The HGA stops at each step t when no improvement has occurred after 100 generations. HGA is coded in the C language.

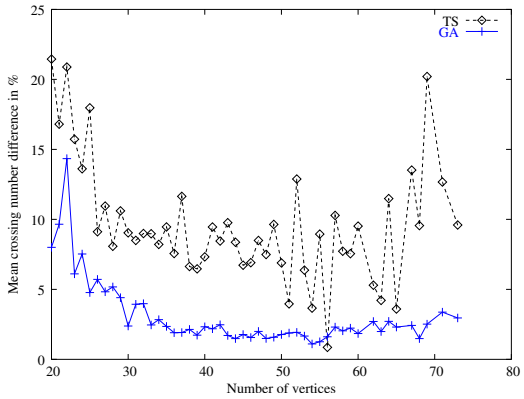
Figure 3-a shows the average crossing number evolution for the whole set of graphs built by the generator. We here retain the crossing number of the selected solution and compare it with a drawing optimized by TS. As expected, the crossing number linearly increases with the vertex number. Except for one case, HGA solutions are always better than TS solutions. And the improvement is all the greater as the vertex number increases.

The algorithm naturally implies that the selected drawing at each step t (which takes into account the stability constraint) is close to the best solution of HGA. This is confirmed by Figure 3-b which shows the difference between the crossing number of the selected solution and the crossing number of the best HGA solution. The mean difference is equal to 3.14% (std. dev. 2.42%) for HGA. However, it is significantly higher for TS: 9.64% (std. dev. 4.28%).

To better understand the integration of the stability constraint in the dynamic drawing procedure, we have studied the evolutions of the similarity index $\delta\left(\Pi_{G_t}, \Pi_{G_{t-1}}\right)$. Computing the mean value of the similarity index on the whole set of graphs is a nonsense. We prefer to show a figure of this evolution with a representative graph (Figure 4). This figure shows the evolution of δ for the best HGA solutions and the selected solutions. The similarity is significantly greater for the selected solutions. The important variations which appear from time to time are due to the dynamics: depending on the positions of the added vertices and edges, graphs G_{t-1} and G_t may be quite different and consequently require a new drawing. In this case, the subset of the best HGA generated solutions for the crossing constraint contains drawings of G_t different from those of G_{t-1}. However, overall, our strategy tends to preserve the likeness.



(a) Average crossing number for the selected solution. Comparison with TS



(b) Mean difference of the crossing number of the HGA selected solution and the best TS solution with the best static HGA solution

Figure 3: Analysis of the drawing quality for the dynamic evolution of a set of 90 pseudo-random graphs (total of 1400 layouts).

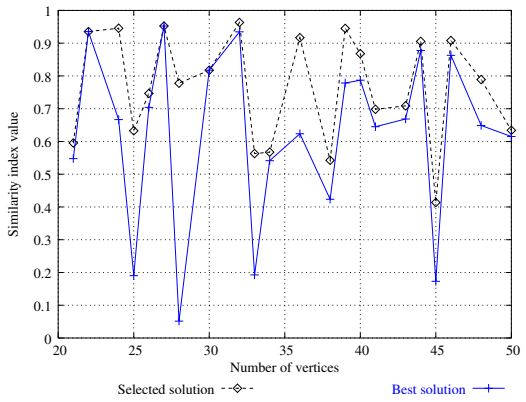


Figure 4: Evolution of the stability constraint for the selected solutions and the best HGA solutions.

6 Conclusion and perspectives

We developed a procedure for dynamic hierarchical digraph drawing based on a hybridized genetic algorithm. Computational experiments were done with a random graph generator adapted to this problem. They showed that quality-wise the solution displayed to the user (*i.e.* computed to be visually similar to the previous drawing) was not much debased from the best layout found. Moreover, our method also outperforms one of the best known metaheuristics for the static problem.

After these promising results, we still have to compare our algorithm with some widely used methods for multiobjective optimizations such as a real multiobjective genetic algorithm or more generally with Pareto-based approaches [18]. There is also another interesting point to study further, as noted in Section 3, the similarity measure. We need to use some more complex measures by, for instance, taking into account the degree of each vertex. A vertex with a high degree might produce numerous arc-crossings contrary to a vertex with a lower degree. In the construction phase of a new drawing, moving this vertex far from its original position will result in important changes in the structure of the graph. This consequently generates a new drawing quite dissimilar to the previous one which is not what we aim at.

For carrying out such experiments, we need a graph generator which can reproduce the behaviour of a human being. In the present version the addition of new vertices is done completely at random. When an individual uses a real application, he does not have a random behaviour. He may intensify a part of the graph, leaving another one not very closely connected ; he may also add vertices only in the first layer and not in the others, and so on. We need to enhance the generator with some human behaviours such as a set of scenarios of real graph constructions done by a person.

References

[1] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph drawing - Algorithms for the visualization of graphs*. Prentice Hall, 1999.

[2] H. Purchase. Effective information visualization: a study of graph drawing aesthetics and algorithms. *Interacting with computers*, 13(2), 2000.

[3] P. Eades and N. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.

[4] A. Papakostas, J.M. Six, and I.G. Tollis. Experimental and theoretical results in interactive orthogonal graph drawing. In *Proc. of Graph Drawing'96*, volume 1190 of *Lecture Notes in Computer Sciences*, pages 371–386. Springer Verlag, 1997.

[5] P. Eades, W. Lai, K. Misue, and K. Sugiyama. Preserving the mental map of a diagram. In *Proc. of Compugraphics*, pages 24–33, 1991.

[6] L.J. Groves, Z. Michalewicz, P.V. Elia, and C.Z. Janikow. Genetic algorithms for drawing directed graphs. In *Proc. of the 5^{th} Int. Symp. on Methodologies for Intelligent Systems*, pages 268–276. Elsevier, 1990.

[7] A. Ochoa-Rodrìguez and A. Rosete-Suàrez. Automatic graph drawing by genetic search. In *Proc. of the 11^{th} Int. Conf. on CAD, CAM, Robotics and Manufactories of the Future*, pages 982–987, 1995.

[8] J. Utech, J. Branke, H. Schmeck, and P. Eades. An evolutionary algorithm for drawing directed graphs. In *Proc. of the Int. Conf. on Imaging Science, Systems and Technology*, pages 154–160. CSREA Press, 1998.

[9] P. Kuntz, B. Pinaud, and R. Lehn. Elements for the description of fitness landscapes associated with local operators for layered drawings of directed graphs. In M.G.C. Resende and J.P. de Sousa, editors, *Metaheuristics: Computer Decision-Making*, volume 86 of *Applied optimization*, pages 405–420. Kluwer Academic Publishers, 2004. ISBN 1-4020-7653-2.

[10] M. Laguna, R. Marti, and V. Valls. Arc crossing minimization in hierarchical digraphs with tabu search. *Computers and Operation Research*, 24(12):1175–1186, 1997.

[11] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Trans. Syst., Man, Cybern.*, 11(2):109–125, 1981.

[12] R. Marti. Arc crossing minimization in graphs with GRASP. *IIE Trans.*, 33(10):913–919, 2001.

[13] J. Branke. Dynamic graph drawing. In D. Wagner M. Kaufmann, editor, *Drawing Graphs: methods and models*, pages 228–246. Springer, 2001.

[14] K.F. Böhringer and F.N. Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *Proc. of CHI'90*, pages 43–51. ACM, 1990.

[15] S.C. North. Incremental layout in DynaDAG. In *Proc. of Graph Drawing'95*, volume 1027 of *Lecture Notes in Computer Science*, pages 409–418. Springer-Verlag, 1996.

[16] S. Bridgeman and R. Tamassia. A user study in similarity measures for graph drawing. *J. of Graph Algorithms and Applications*, 6(3):225–254, 2002.

[17] D. Whitley and N. Yoo. Modeling simple genetic algorithms for permutation problems. In L. D. Whitley and M. D. Vose, editors, *Fundations of Genetic Algorithms III*, pages 163–184. Morgan Kaufmann, 1995.

[18] Carlos A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and information systems*, 1(3):269–308, 1999.