Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in DISCRETE MATHEMATICS

Electronic Notes in Discrete Mathematics 58 (2017) 183–190 www.elsevier.com/locate/endm

Variable neighborhood descent for the incremental graph drawing

J. Sánchez-Oro [a,1] A. Martínez-Gavara [b,2] M. Laguna [c,3]
A. Duarte [a,4] R. Martí [b]

[a] *Department of Computer Science, Universidad Rey Juan Carlos, Móstoles, Spain*

[b] *Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Madrid, Spain*

[c] *Leeds School of Business, University of Colorado at Boulder, USA*

Abstract

Graphs are used to represent reality in several areas of knowledge. Drawings of graphs have many applications, from project scheduling to software diagrams. The main quality desired for drawings of graphs is readability, and crossing reduction is a fundamental aesthetic criterion for a good representation of a graph. In this paper we target the edge crossing reduction in the context of incremental graph drawing, in which we want to preserve the layout of a graph over successive drawings. We propose a hybrid method based on the GRASP (Greedy Randomized Adaptive Search Procedure) and VND (Variable Neighborhood Descent) methodologies and compare it with previous methods via simulation.

Keywords: Incremental graph drawing, variable neighborhood descent, metaheuristics

http://dx.doi.org/10.1016/j.endm.2017.03.024

1 Introduction

Most of the information systems nowadays are commonly represented by a drawing, which makes the system easier to interpret and understand. Graphs are the basic modeling unit in a wide variety of areas, like project and production scheduling, line balancing, business plans or software visualization. For this reason, graph drawing has become an important research area, with a large number of publications related. We refer the reader to [2] for a thoroughly survey on graph drawing. The selection of an objective measure of the quality of a graph is a controversial subject. However, the number of crossing edges is a widely admitted criterion for evaluating the quality of a draw. Specifically, the fewer of crossings, the better the drawing is [1]. The problem of minimizing the number of crossings is $\mathcal{NP}$-complete [5].

This paper focuses on finding the best drawing for hierarchical directed acyclic graphs, HDAG, which are usually known as hierarchical graphs, layered digraphs, or simply hierarchies. In order to represent a HDAG, we first need to draw the vertices in equally spaced vertical lines (layers), in such a way that every directed edge goes in the same direction. With this arrangement of vertices, arc crossing minimization consists of finding the appropriate ordering of the vertices in each layer. Fig. 1 shows a drawing of a HDAG with 8 vertices and 10 edges. The HDAG is split into three layers (highlighted in gray). The number of crossings between the first and second layer is 1, in the pair of edges $(3, 8) - (7, 2)$, while the number of crossings between the second and third layer is 3, in the pairs $(2, 1) - (8, 6)$, $(2, 5) - (8, 6)$, and $(2, 5) - (8, 1)$, respectively.

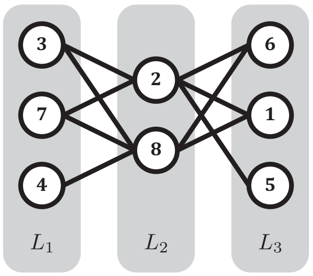

Fig. 1. Example of a drawing of a HDAG with 8 vertices and 10 edges.

Considering hierarchical digraphs is not a loss of generality since there are

[1] Email: jesus.sanchezoro@urjc.es
[2] Email: gavara@uv.es
[3] Email: laguna@colorado.edu
[4] Email: abraham.duarte@urjc.es

several well-known algorithms to convert any directed acyclic graph (DAG) into a HDAG. The simplest method consist of setting all the vertices in layers, considering that all the edges present the same direction (from a lower to a higher layer). The next step consists of removing those edges that connect vertices located in non consecutive layers. This removal is performed by adding artificial vertices in each layer traversed by the edge. For example, if an edge connects a vertex located in layer 1 with a vertex located in layer 4, then two new vertices are included, one in layer 2 and another one in layer 3, creating a path from vertex in layer 1 to vertex in layer 4.

Incremental graph drawing constructions are motivated by the need to support the interactive updates performed by the user. In this situation, it is helpful to preserve a "mental picture" of the layout of the graph over successive drawings. It can be distracting to make a slight modification on a drawing, perform the graph drawing algorithm, and have the resulting drawing appear very different from the previous one. In this paper, we consider the problem of minimizing the number of arc crossings when new vertices and edges are added to the hierarchical graph, while preserving the relative ordering in each layer among the original vertices.

2 Previous methods

Sugiyama [12] proposed the so-called barycentric method for arc crossing minimization in hierarchical digraphs. This is probably the most established method for crossing reduction. In their approach, a series of 2-layer subproblems are solved starting from an initial permutation for the first layer (L_1). The procedure moves from "left" to "right" until the subproblem involves the last two layers, and then from "right" to "left" until the first two layers are once again considered. Each subproblem i in the left-to-right sweep consists of finding the "best" permutation for layer L_i considering that the permutation of layer L_{i-1} is fixed. Similarly, each subproblem i in the right-to-left sweep consists of finding the "best" permutation for layer L_i considering that the permutation of layer L_{i+1} is fixed. Each subproblem is solved using the barycentric method originally designed for the 2-layer problem (also known as the relative degree algorithm or averaging). In this method, the position of each vertex v in layer i is given by the arithmetic mean of the positions of the vertices in layer L_{i+1} (in a left-to-right sweep) that are adjacent to vertex v. Similarly, the average position of the adjacent vertices in layer L_{i-1} determines the barycenter of vertex v in a right-to-left sweep. The original order is preserved when two vertices have the same barycenter, during the first phase of the procedure. A second phase is used to switch the order of vertices with equal barycenters. With the orderings determined during the second phase, the first phase is then re-applied.

After the seminal work by Sugiyama, many different algorithms have been proposed to minimize the number of crossings. Since this is an $\mathcal{NP}$-hard problem with practical applications, we can find both exact [7] and metaheuristic algorithms [8,9] to solve it. In spite of its practical significance, very little attention has been paid to the incremental variant of the graph drawing problem. We have only identified one previous contribution [10] which was limited to the case with only two layers.

Martí and Estruch [10] proposed both exact and heuristic approaches for the incremental two layer problem. The exact algorithm is a Branch & Bound method, while the heuristic one is based on the GRASP methodology, in which they proposed a constructive procedure based on locating the vertices close to their barycenter. As it is customary in GRASP, the local search post-processing is intended to improve the results of the constructive phase by moving each vertex to a position close to its barycenter. As far as we know, this method is considered the state of the art in terms of the incremental graph drawing problem. We will therefore compare the results of our proposal with those obtained with this method.

3 Variable Neighborhood Descent

Variable Neighborhood Search (VNS) is a metaheuristic for solving optimization problems based on systematic changes of neighborhood structures. In recent years, a large variety of VNS strategies have been proposed: Basic VNS, Reduced VNS, Variable Neighborhood Descent, Skewed VNS, and General VNS, among others. We refer the reader to [6] for a complete survey on VNS.

This work is focused on Variable Neighborhood Descent, which typically starts from a randomly generated solution, or using a heuristic constructive procedure, which may help the method by starting the search exploring a promising area of the solution space. Previous proposals have shown that using an initial solution of relatively high quality leads the VND algorithm to better results [3,11]. In line with that, we propose a constructive procedure to generate the initial solutions with good quality to launch the VND search. In particular we hybridize GRASP, which generates good and disperse initial solutions, with VND, which improves the constructed solutions searching on different neighborhoods.

Our constructive method considers that all the original vertices V are initially included in the incremental drawing ID following the same order as in the original drawing D. Then, the candidate vertices to be inserted in ID is $CL \leftarrow IV \setminus V$. The method evaluates the degree of each vertex with respect to those vertices already included in ID (initially V), constructing the restricted candidate list (RCL) with those vertices with a degree larger than or equal to a predefined threshold. The threshold is computed as $g_{max} - \alpha \cdot (g_{max} - g_{min})$, where g_{max} and g_{min} are the highest and lowest degree found in the candidate list, respectively. This constructive method is based on the GRASP methodology [4], where α is a parameter of the method which determines its level of greedyness/randomness. Specifically, if $\alpha = 0$, the method is a pure random method, while setting $\alpha = 1$ makes it totally greedy.

After generating a solution using the aforementioned constructive procedure, it is improved with the VND method, which is presented in Algorithm 1. The method starts from the initial solution ID, by exploring the first neighborhood (step 3), identifying its best solution $ID^\star$ (step 4), which eventually, can improve the current solution ID. This is tested in steps 5 to 10 where if $ID^\star$ does not improve upon ID, the method resorts to the next neighborhood by incrementing the value of k. Otherwise, the method updates ID and restarts the search from the first neighborhood (step 7). The algorithm finishes when none of the neighborhoods contain a solution better than the current one.

Algorithm 1 $VND(ID, k_{max})$

```
 1: $k \leftarrow 1$
 2: repeat
 3:     Explore the neighborhood $N_k(ID)$
 4:     Let $ID^\star$ be the best solution in $N_k(ID)$
 5:     if $Crossings(ID^\star) < Crossings(ID)$ then
 6:         $ID \leftarrow ID^\star$
 7:         $k \leftarrow 1$
 8:     else
 9:         $k \leftarrow k + 1$
10:     end if
11: until $k = k_{max}$
```

We have considered five neighborhood structures, all of them based on insert a given node in a different position in the same layer. Given a vertex v in position p of layer L_i, the insertion move $Insert(v, s)$ is defined as inserting vertex v in position $p + s$. Note that s can take positive and negative values. Specifically, the move $Insert(v, s)$ can only be applied if $1 \leq p + s \leq |L_i|$. Therefore, when the vertex v is the first one ($p = 1$), or the last one ($p = |L_i|$) in the layer, only one of these two moves can be considered. Neighborhood N_k contains the two solutions that can be obtained by applying either $Insert(v, k)$

or $Insert(v, -k)$, or both if possible. As shown in Algorithm 1, when N_k does not include a solution better than the current one, the method considers the solutions in N_{k+1}, until $N_{k_{max}}$ is reached.

Both the constructive and VND methods are embedded in a Greedy Randomized Adaptive Search Procedure (GRASP) algorithm [4]. GRASP is a multi-start methodology where each iteration consists of two stages. The first one is a greedy, randomized and adaptive construction of a solution. The second stage applies an improvement method to guide the constructed solution to a local optimum. These two steps are repeated until a termination criterion is met. The construction phase is performed using the constructive procedure described above while the improvement stage is performed with the VND algorithm. In this work we have selected the number of iterations as a termination criterion, totalizing 100 constructions and VND improvements.

4 Experimental results

This section presents the experimental analysis of the proposed algorithms. All the algorithms have been coded in Java 8, and the experiments have been performed in an Intel Core i7 920 (2.67 GHz) with 8 GB RAM. We have generated a set of instances based on the method proposed in [8], using 2, 6, 13, and 20 as the number of layers, and varying the graph density in the range $\{6.5\%, 17.5\%, 30\%\}$. The vertices of these original instances have been set applying the barycenter algorithm until no reduction on the number of crossings is achieved. After that, a set of new vertices and edges are added, obtaining the incremental graph, which conforms the complete instance to be tested with the proposed algorithm. Specifically, the number of vertices and edges added to the original graph is computed as a 20% of the vertices and edges in the original graph respectively. In a similar way we created instances incremented in a 60%.

In order to generate diverse instances, we randomly generated the number of vertices in each layer in the range $[5, 30]$. An instance has been generated for each possible parameter combination (number of layers, density, percentage of new vertices and edges), totalizing a set of 24 instances. Table 1 shows the results of the proposed algorithm compared with the best previous method found in the literature. As stated before, the best previous method (BestPrev in the table) is a GRASP algorithm which consist of a constructive method and a local search, both based in the barycenter algorithm. Following the authors instructions, the previous algorithm has been executed for 100 iterations. In the case of VND, we present the results when considering different values for the maximum neighborhood explored, k_{max} (which is specified between parenthesis in the name of the algorithm).

	BestPrev	**VND(1)**	**VND(2)**	**VND(3)**	**VND(4)**	**VND(5)**
Avg. Crossings	14101.13	12963.33	12668.50	12596.08	12546.71	12529.46
Time (s)	16.09	0.36	7.86	10.05	11.54	13.35
Dev (%)	58.47	8.65	1.43	0.69	0.20	0.00
#Best	0	1	4	7	9	24

Table 1
Comparison among the proposed method and the best previous state-of-the-art method.

Table 1 reports, for each method, the average number of crossings obtained, Avg. Crossings; the computing time in seconds, Time (s); the average percentage deviation with respect to the best known value, Dev (%); and the number of best solutions found, #Best. The results show the superiority of the proposal, even when considering the smallest neighborhood, VND(1). As expected, the computing time also increases when considering larger neighborhoods, since they explore a larger region of the solution space. However, even when considering the slowest method, VND(5), the associated computing time is reasonably small. The best results in terms of quality are achieved by VND(5).

We have additionally performed non-parametric statistical test in order to confirm the conclusions above. Specifically, we have performed the Friedman test for k samples, resulting in a p-value lower than 0.0001, which supports the comments above. Finally, we have performed the Wilcoxon sign test between the best presented method, VND(5), and the best previous method, BestPrev. The resulting p-value lower than 0.0001 confirms that there exist statistical differences between the results obtained with both methods.

5 Conclusions

In this work we have tackled the Incremental Graph Drawing Problem, which seeks to reduce the number of crossings due to new edges added to a previously drawn graph. We have proposed a Variable Neighborhood Descent algorithm, combined with the GRASP methodology to obtain high quality solutions in small computing times. Specifically, we have proposed a constructive method to generate initial solutions of a relatively high quality, as well as five neighborhood structures embedded in the VND method. Experimental results, supported with statistical tests, have shown that the proposed algorithm outperforms the best previous method.

Acknowledgments

This work has been partially founded by Ministerio de Economa y Competitividad with grants ref. TIN2015-65460-C2-1-P and TIN2015-65460-C2-2-P (MINECO / FEDER) and by Comunidad Autnoma de Madrid with grant ref. S2013ICE-2894.

References

[1] Carpano, M. J., *Automatic Display of Hierarchized Graphs for Computer-Aided Decision Analysis*, IEEE Transactions on Systems, Man, and Cybernetics **10** (1980), pp. 705–715.

[2] Di Battista, G., P. Eades, R. Tamassia and I. G. Tollis, "Graph Drawing: Algorithms for the Visualization of Graphs," Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998, 1st edition.

[3] Duarte, A., J. J. Pantrigo, E. G. Pardo and J. Sánchez Oro, *Parallel variable neighbourhood search strategies for the cutwidth minimization problem*, IMA Journal of Management Mathematics **27** (2016), pp. 55–73.

[4] Feo, T. A., M. G. C. Resende and S. H. Smith, *A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set*, Operations Research **42** (1994), pp. 860–878.

[5] Garey, M. R. and D. S. Johnson, *Crossing Number is NP-Complete*, SIAM Journal on Algebraic Discrete Methods **4** (1983), pp. 312–316.

[6] Hansen, P., N. Mladenović and J. A. Moreno Pérez, *Variable neighbourhood search: methods and applications*, 4OR **6** (2008), pp. 319–360.

[7] Jünger, M., E. K. Lee, P. Mutzel and T. Odenthal, *A polyhedral approach to the multi-layer crossing minimization problem*, in: G. Di Battista, editor, *Proc. of 5th International Symposium on Graph Drawing* (1997), pp. 13–24.

[8] Laguna, M., R. Martí and V. Valls, *Arc crossing minimization in hierarchical digraphs with tabu search*, Computers & Operations Research **24** (1997), pp. 1175–1186.

[9] Martí, R., *A tabu search algorithm for the bipartite drawing problem*, European Journal of Operational Research **106** (1998), pp. 558–569.

[10] Martí, R. and V. Estruch, *Incremental bipartite drawing problem*, Computers & Operations Research **28** (2001), pp. 1287–1298.

[11] Sánchez Oro, J., N. Mladenović and A. Duarte, *General Variable Neighborhood Search for computing graph separators*, Optimization Letters (2014), pp. 1–21.

[12] Sugiyama, K., S. Tagawa and M. Toda, *Methods for Visual Understanding of Hierarchical System Structures*, IEEE Transactions on Systems, Man, and Cybernetics **11** (1981), pp. 109–125.