Using PQR-trees for reducing edge crossings in layered directed acyclic graphs

João Rubens Marchete Filho, Celmar Guimarães da Silva
Software Engineering and Information Systems Laboratory
School of Technology, University of Campinas
Limeira, Brazil

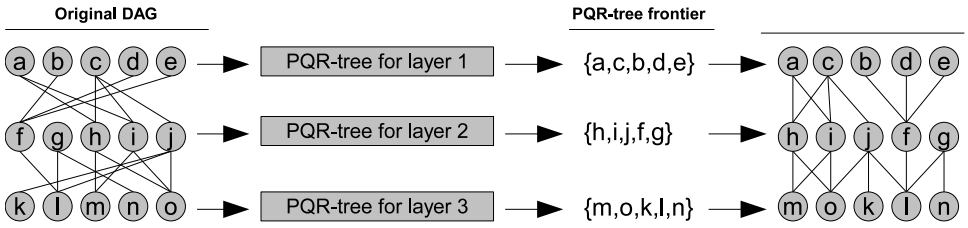


Fig. 1. Teasing result of our method: from the data input (left), we create a PQR-tree for each layer, extract their frontiers (middle) and reorder layers according to them (right), aiming to reduce edge crossings.

***Abstract*—Minimizing edge crossings of a layered directed acyclic graph is necessary for easing its visual analysis. Sugiyama framework introduced barycenter heuristic (BC) as a method for crossing reduction in this kind of graphs. This paper presents a new algorithm that enhances the performance of this method by the use of PQR-trees, a recent data structure that represents permutations of a set of elements that obey some defined grouping restrictions. We improved the quality of BC method by using PQR-trees in the permutation of the nodes in each layer of the graph. Our approach reaches 10% less crossings than the original BC in our experiments at the cost of multiplying its average execution time by approximately 1.45. Besides, this time remains lower than 100 ms for the tested graphs, which is a necessary condition for providing responsive interaction.**

***Keywords*-Data visualization; graph theory; user interfaces**

I. Introduction

Reducing the number of edges crossing is a very influential aesthetic criterion for helping people to understand the relations present in graphs [1]. This is also true for directed acyclic graphs (a.k.a. DAGs), data structures used to represent hierarchy-related problems, such as PERT diagrams, VLSI circuits, genealogical structures and class dependency on software [2]. Many approaches represent them as layered DAGs, i.e., they distribute graph nodes on horizontal (or vertical) sets called layers [3].

A drawing convention adopted for DAGs includes three items [4]: edges must be in a same direction (flow), nodes must be equally distributed over the drawing area, and edge crossings must be minimal. Minimizing these crossings, however, is a NP-complete problem [5], which demands alternative techniques for reaching a low number of crossings as fast as possible (if we intend to present it for users). Many approaches that produce good results for this problem are based on Sugiyama framework [3], which uses barycenter (BC) and median heuristic methods for crossing reduction.

However, there is no guarantee that these methods will achieve the best results and will be performed as fast as needed to reach them.

Contributions: This paper presents a different approach for crossing minimization problem in layered DAGs. By introducing PQR-trees to this context, we achieve significant improvements on BC method results, at an extra time cost. In particular, we test our approach with North DAGs [6]. In our experiments, we evaluate that our BC+PQR method returns 10% less crossings than the original BC, and spends in average 45% more time than BC, which validate our approach.

A. Related work

We can classify the approaches for crossing minimization in DAGs in two categories: approaches that follow Sugiyama framework, and approaches that do not (such as planarization [7]). We will focus on the first category.

The *Sugiyama framework* [3] defines the following sequence of steps for drawing a DAG: defining a layer for each node, creating dummy nodes for edges whose nodes belong to non-adjacent layers, minimizing edges crossing and defining coordinates for each node. We highlight the crossing minimization step, also called *multi-layer crossing minimization* (MLCM) step, which tries to define an ordering for the nodes of each layer. A technique called *layer-by-layer sweep* defines that solving a MLCM problem demands to iterate through pairs of adjacent layers of the DAG in order to solve a *two-layer crossing minimization* (TLCM) problem related to each pair of layers [2], i.e., minimizing crossings of bipartite graphs.

Many approaches have been used for solving TCLM and MCLM problems. They include methods that use tech-

niques such as linear, quadratic and semidefinite programming [8] [9] [10], tabu-search and genetic algorithms [11]. However, barycenter (BC) and median heuristic methods [3] are still the main reference for solving TLCM problems. In a TLCM problem, barycenter (or median) heuristic calculates the position of a node as the average (or median) position of its neighbors in the opposite layer. Layer-by-layer sweep and BC may be combined to solve MLCM problems; as a stop condition, the sweep process must stop after a predefined number of fails, i.e., situations in which the number of crossings was not reduced after a sweep [12].

It is worth to note the relationship between TLCM and binary matrix reordering. Given a bipartite graph and its adjacency matrix (which is binary), Mäkinen and Siirtola [13] perceived that minimizing edge crossings by BC heuristic tends to group cells with value "1" in top left and bottom right corners of this matrix.

B. Technique overview

Given our previous experience on applying PQR-trees for solving binary matrix reordering [14], and the relationship between this problem and crossing minimization, we present some approaches for using PQR-trees for solving MLCM. We formulate the problem of reducing edge crossings in a DAG as many problems of finding good permutations of layer elements. PQR-trees may represent permutations of these elements according to some rules which try to group nodes in layers. We use this permutation to reorder each layer and thereby reduce crossings. This process is used together with BC method as means to improve its results.

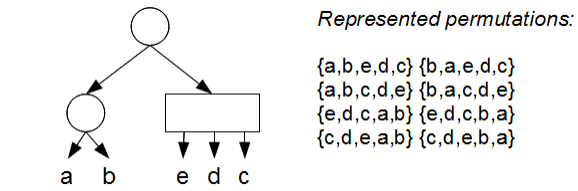
II. Technical Background

In this section, we briefly present what is a PQR-tree. The reader can find a more complete exposition of PQR-trees in the work of Telles and Meidanis [15].

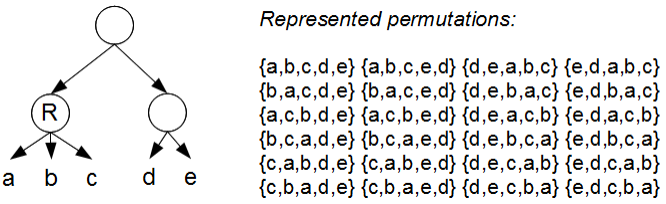
A. PQR-tree

A *PQR-tree* [15] is a data structure that represents a subset S of all possible permutations of a set of elements U. Each permutation of this subset tries to agree to a set R of restrictions. Each restriction is a subset of U whose elements should be consecutive in the permutations of S. A PQR-tree $T(U, R)$ is a tree whose leaves are elements of U, and whose non-leave nodes defines how these leaves may be permuted. Non-leave nodes may be from three types: P and R-node's children admit any kind of permutation, and Q-node's children may be only reversed. For example, Fig. 2a represents a PQR-tree related to $U = \{a, b, c, d, e\}$ and to the restriction set $R = \{\{a, b\}, \{d, e\}, \{c, d\}\}$. Observe that the frontier of this tree (i.e., its leaves, read from left to right; in this case, $\{a, b, e, d, c\}$) is one of the permutations in S.

R-nodes are used when it is impossible to obey simultaneously to all restrictions. In the previous example, if $R = \{\{a, b\}, \{a, c\}, \{b, c\}, \{d, e\}\}$, every permutation of the elements of U disagree with some of the three first restrictions. Therefore, a R-node becomes a parent node of a, b and c,



(a) A PQR-tree with two P-nodes (empty circles) and a Q-node (empty rectangle).



(b) A PQR-tree with two P-nodes (empty circles) and a R-node (circle with a "R" inside it).

Fig. 2. Examples of PQR-trees and the permutations that they represent.

as shown in Fig. 2b. Note that in this case the permutations represented by the PQR-tree still obey to the last restriction of R. Indeed, PQR-trees return an usable solution even when there is not a perfect one.

It is worth to consider that the construction of a PQR-tree has an almost linear time complexity [15].

B. Usual adaptation

This concept has been used for solving problems similar to ours [14] but related to binary matrix reordering. In this case, an algorithm constructs two PQR-trees, one for matrix columns and other for matrix rows. For the first one, the algorithm defines U as the set of columns, and R as a set of groups of columns that should be consecutive (i.e., similar columns). The frontier of this tree has an ordering of columns that tries to create these groups. The second tree is created in a similar way, but for rows. Therefore, the algorithm permutes columns and rows of the original matrix in the same way defined by the given orderings.

III. Technique Adaptation

Our technique aims at obtaining a low number of crossing edges in DAGs in a short execution time. PQR-trees suit to this problem since it is formulated as follow:

A. Formulation

Given a MLCM problem, we want to reduce the number of crossing edges of the original graph layout by permuting nodes of each layer.

B. Solution

We adapted the use of PQR-trees for DAGs, aiming to group nodes of a same layer that have a common neighbor in any adjacent layer. Therefore, we construct a PQR-tree for each layer for the purpose of reorganizing it and minimizing crossings. Given a layer L_i, we define U as the elements of

L_i. For the purpose of grouping nodes that have a common neighbor in any adjacent layer, we fix the order of L_{i-1} and L_{i+1} (when they exist), and for each node n of these layers, we define a restriction whose elements are the neighbors of n in L_i. After that, we create a PQR-tree based on U and R, and we define the order of L_i elements as the same order of the tree frontier. Fig. 1 summarizes this approach.

Some preliminary experiments revealed that this approach alone did not produce good results, probably because our problem is a MLCM and not a TLCM (which is closer to a single matrix reordering problem). Therefore, we combine it with BC method in two ways. In the first one, we execute our approach after a given number f of fails of BC, chosen empirically, in order to help BC method to find a better solution. We called this combination *BC+PQR*. In the second one, we execute our approach only one time, after the execution of BC, aiming to accelerate BC. We called this second combination *PQR+BC*.

C. Initialization and tuning

Our approaches consider that its input is a proper hierarchy [3] (a DAG in which every edge connects two consecutive layers). A non-proper hierarchy may be converted into a proper one by the insertion of dummy nodes in the edges.

For BC+PQR, it is necessary to define the value of f. PQR+BC does not have a tuning parameter.

IV. Implementation

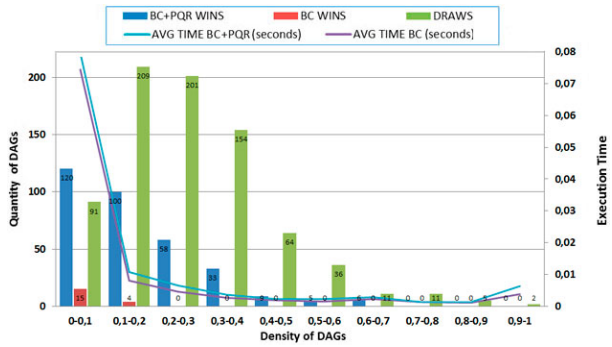
We built the experiment on OGDF (*Open Graph Drawing Framework*) [12], a C++ framework that implements the whole Sugiyama framework, including BC and other crossing minimization methods. OGDF is expansible, so we created functions that implement our approaches and attached them to the BC implementation. We also used a PQR-tree implementation in Java, which we connected to OGDF through JNI.

V. Experiment

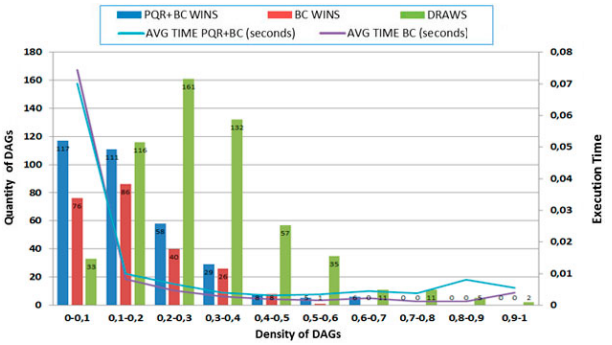
We validate our 2 methods through an experiment used to compare their crossing minimization performance versus BC method. The performance measures were: number of crossings, execution time of the crossing minimization step and the relationship between them (also used by Jünger & Mutzel [8]). We also want to analyze if the execution time of these methods is less than 100 ms, so they would be appropriate for providing responsive interaction [16] in user interfaces.

VI. Results and Discussion

We performed the above-mentioned experiment on a test set called *North DAGs* [6], which was previously layered by OGDF (longest path ranking algorithm). This set has a high concentration of low density graphs (approximately 82% of its graphs has density $\leq$ 0.4). We calculated DAG density according to Kuntz et al. [11]. For each graph, we tuned BC+PQR for using f=8 (as an empirically defined value). Besides, BC was tuned for 1 run and also 8 fails. Fig. 4 presents examples of these methods applied to a graph of this test set.



(a) Comparison of BC+PQR and BC methods



(b) Comparison of PQR+BC and BC methods

Fig. 3. Comparison of crossing minimization approaches. *AVG TIME* stands for the average execution time of a method.

TABLE I
Quality measures

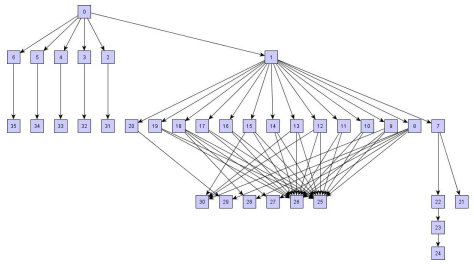
	BC+PQR	**PQR+BC**
Number of graphs		
- Method wins BC	29%	21%
- Draw	69%	50%
- BC wins method	2%	29%
Crossings		
- Average crossing reduction (relative to BC results)	10%	1%
- Average crossing reduction (relative to BC results in wins)	34%	39%
Time		
- Average time (relative to BC)	145%	184%

A. Quality and Performance

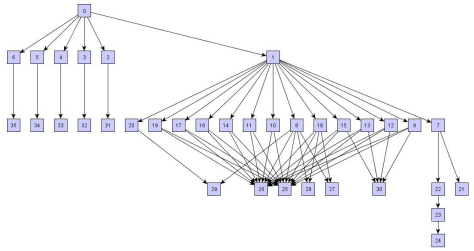
We report on Fig. 3a a comparison of BC and BC+PQR methods on a computer with Intel Core i5 processor at 2.5 GHz. We also present on Fig. 3b a comparison of BC and PQR+BC methods on the same computer. These graphics indicate that our methods lose only in few number of graphs for BC. They also show that all methods spend more time in the lower density graphs. However this time is lower than 100 ms in average, which is desired for providing responsive interaction.

The quality of our approach can be measured by analyzing number of wins of each method, average crossing reductionand average time (both relative to BC), as reported on Table I.

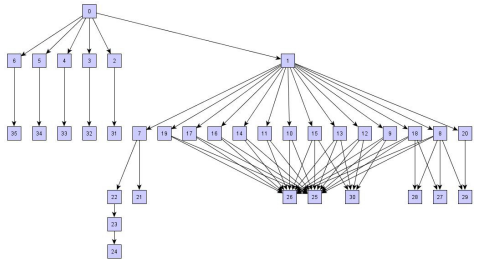
We highlight the performance of BC+PQR, which loses in



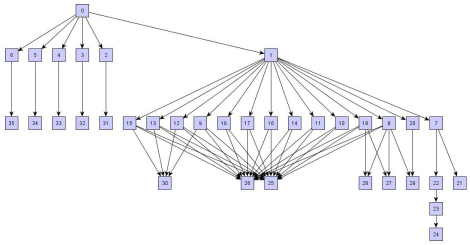
(a) Initial layout: 205 crossings



(b) Layout after BC method: 127 crossings



(c) Layout after PQR+BC method: 99 crossings



(d) Layout after BC+PQR method: 83 crossings

Fig. 4. Examples of graph layouts and respective number of crossings.

crossing reduction to BC in only 2% of the graphs, spending 45% more time (which is less than 2 ms in average) and reducing 34% more crossings in average than BC when BC+PQR wins. PQR+BC performance is not so good, because it almost doubles the BC execution time and loses in crossing reduction to BC in 29%, even though it reduces 39% more crossings when it wins. If we consider the entire graph package, BC+PQR reduces 10% more crossings in average than BC, and PQR+BC only reduces 1%.

B. Limitation

It is worth to note that our experiment used only North DAGs, and we must be aware of this condition before generalizing these results for any DAG set. We plan to analyze more graphs aiming to improve our tests and reinforce the effectiveness of our techniques.

VII. Conclusion

In this paper, we introduced the use of PQR-trees for edge crossing minimization on DAGs. By using them for reordering layers of DAGs, we obtained a good cost-benefit with our methods, specially with BC+PQR: 10% less edge crossings in average, with 45% extra time, when compared to BC method. Ongoing works include: testing our methods with other graph packages; creating and testing alternative approaches based on median heuristic instead of BC; and better understanding why and in which cases our approaches win the other ones.

Acknowledgment

We thank Prof. João Meidanis (Institute of Computing, University of Campinas) by the use of his PQR-tree package, and FAPESP for financial support.

References

[1] H. C. Purchase, "Which aesthetic has the greatest effect on human understanding?" in *Graph Drawing*, ser. Lecture Notes in Computer Science, G. D. Battista, Ed. Springer Berlin / Heidelberg, 1997, vol. 1353, pp. 248–261.

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

[3] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 11, no. 2, pp. 109–125, feb. 1981.

[4] P. Eades and L. Xuemin, "How to draw a directed graph," in *IEEE Workshop on Visual Languages*, oct 1989, pp. 13–17.

[5] P. Eades and N. Wormald, "Edge crossing in drawing of bipartite graphs," *Algorithmica*, vol. 11, pp. 379–403, 1994.

[6] G. D. Battista, A. Garg, G. Liotta, A. Parise, R. Tamassia, E. Tassinari, F. Vargiu, and L. Vis-mara, "Drawing directed acyclic graphs: An experimental study," *Int. J. Comput. Geom. Appl.*, vol. 10, no. 6, pp. 623–648, 2000.

[7] M. Chimani, C. Gutwenger, P. Mutzel, and H.-M. Wong, "Upward planarization layout," in *Graph Drawing*, ser. Lecture Notes in Computer Science, D. Eppstein and E. Gansner, Eds. Springer Berlin / Heidelberg, 2010, vol. 5849, pp. 94–106.

[8] M. Jünger and P. Mutzel, "2-layer straightline crossing minimization: performance of exact and heuristic algorithms," *Journal of Graph Algorithms and Applications*, vol. 1, pp. 1–25, 1997.

[9] L. Zheng and C. Buchheim, "A new exact algorithm for the two-sided crossing minimization problem," in *Proc. of the First International Conference on Combinatorial Optimization and Applications*, ser. COCOA'07. Springer-Verlag, 2007, pp. 301–310.

[10] M. Chimani, P. Hungerländer, M. Jünger, and P. Mutzel, "An SDP approach to multi-level crossing minimization," in *Proc. of the Thirteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2011.

[11] P. Kuntz, B. Pinaud, and R. Lehn, "Minimizing crossings in hierarchical digraphs with a hybridized genetic algorithm," *Journal of Heuristics*, vol. 12, pp. 23–36, 2006.

[12] OGDF, "Open graph drawing framework," http://www.ogdf.net/doku.php/start, 01/2012.

[13] E. Mäkinen and H. Siirtola, "Reordering the reorderable matrix as an algorithmic problem," in *Proc. of the First International Conference on Theory and Application of Diagrams*, ser. Diagrams '00. London, UK, UK: Springer-Verlag, 2000, pp. 453–467.

[14] M. F. de Melo, "Improvement of PQR-Sort algorithm for binary matrices reordering," Master's thesis, School of Technology, University of Campinas, Brazil, 2012.

[15] G. P. Telles and J. Meidanis, "Building PQR trees in almost-linear time," *Eletronic Notes in Discrete Mathematics*, vol. 19, pp. 33–39, 2005.

[16] R. Spence, *Information Visualization*. Addison-Wesley, 2001.