Applying Grid-based Algorithm and Quadratic Programming for Improvement Orthogonal Graphs

Nahla F. Omran
Department of mathematics
Faculty of science, South Valley University
Qena, Egypt

Sara F. Abd-el Ghany
Qena, Egypt
Email: s_comp62 [AT] yahoo.com

Abstract—**An orthogonal drawing of a graph is a drawing such that that all edges are drawn as sequences of horizontal and vertical segments. Traditionally, the quality of orthogonal planar drawings is quantified by either the total number of bends, or the maximum number of bends per edge. Although there are many algorithms for orthogonal graph drawings, but they sometimes produce undesirable drawings. For this reason we will develop a new methodology for drawing orthogonal graphs with nodes of a prescribed size. The algorithm is based on two steps, in the first step we apply grid-based algorithm on the given graph, in this step the final drawing is achieved through three steps: node placement, edge routing and normalization. And the second step we apply the quadratic programming at the second step of the grid-based algorithm that minimize the total edge length. This technique generates an aesthetically pleasing drawing of a given graph. This is achieved clearly in the Results given in the paper.**

Keywords-component; orthogonal graph drawing; grid-based algorithm; quadratic programming

I. INTRODUCTION

Graph drawing algorithms agree on some common criteria for what makes a drawing good, and what should be avoided, and these criteria strongly shape the final appearance of visualization. Each type of graph types suitable for a particular application. We will deal with orthogonal drawing where each edge is drawn as a chain of horizontal and vertical line segments. Orthogonal drawings have attracted much attention due to their numerous applications in circuit layouts, database diagrams, entity relationship diagrams, etc. The Specific requirements in this application are: Initially we will apply the grid-based algorithm that is divided into three main steps. The first step is the node placement step in this step each node moves from place to place, which reduces the length of the adjacent edges. The second step is the edge routing: The goal of this step is to minimize the edge length that minimizes crossing. The third step is the normalization in this step the goal is to normalize the graph. There are many orthogonal drawing algorithm is dedicated to producing drawings of some provable quality aspect. Di Battista et al. [1] Gave an algorithm that minimizes the total number of bends and hence solves 0-embeddability. On the other hand, Biedl and Kant [2] show that every 4-planar graph admits a drawing with at most two bends per edge with the only exception of the octahedron, which requires an edge with three bends. Similar results are obtained by Liu et al. [3]. Liu et al. [4] Claim to have found a characterization of the planar graphs with minimum degree 3 and maximum degree 4 that admit an orthogonal embedding with at most one bend per edge. They also claim that this characterization can be tested in polynomial time. Unfortunately, their paper does not include any proofs and to the best of our knowledge a proof of these results did not appear. Morgana et al. [5] characterize the class of planar graphs (i.e., planar graphs with a given embedding)
That admits a 1-bend embedding on the grid by forbidding configurations. They also present a quadratic algorithm that either detects a forbidding configuration or computes a 1-bend embedding. If the combinatorial embedding of a 4-planar graph is given, Tamassia flow networks can be used to minimize the total number of bends [6]. Note that this approach may yield drawings with a linear number of bends for some of the edges. Given a combinatorial embedding that admits a 1-bend drawing, however, the flow network can be modified in a straightforward manner to minimize the total number of bends using at most one bend per edge. In 2000, A. Papakostas, and I.G. Tollis [7], present an algorithm based on creating groups of vertices of the graph. Two important properties of the algorithm are that the drawings exhibit a small total number of bends, and that there bends at most one per edge. In 2012, Thomas Bläsius · Marcus Krug · Ignaz Rutter · Dorothea Wagner [8] investigate an approach that allows to specify the maximum number of bends for each edge individually, depending on its importance. In 2005, Xiao Zhou and Takao Nishizeki [9] present an algorithm to find an optimal orthogonal drawing of any given series-parallel graph of the maximum degree at most three. The algorithm takes linear time, while the previously known best algorithm takes cubic time. Furthermore, the algorithm is much simpler than the previous one. Also obtain a best possible upper bound on the number of bends in an optimal drawing.

Finally, applied the quadratic programming at the edge routing step of the GBA to compact the graph by minimizing the edge length of the graph. In 2007, Michel Neuhaus and Horst Bunke [10] used quadratic programming to compute the edit distance of the graph. In 2010, Tim Dwyer, Kim Marriott, and Peter Sbarski [11] used quadratic programming to replace the nodes in the graph for minimizing edge crossing.

In this chapter, we will use GBA to find the place of each node in the graph to minimize the bends and then apply the QP in the second step of GBA to minimize the edge length in the graph to produce a graph with good esthetic criteria.

II. THE GRID BASED ALGORITHM

We use the graph that each node represented by rectangles of a given minimum size, edges is represented with orthogonal polylines connecting the associated nodes. The crossing between nodes or between edges and nodes must be minimized. The distance & between them must be minimized. The GBA algorithm is divided into three steps: node placement, edge routing and normalization. The grid based algorithms have two advantages: first, their performance on planar graphs has been theoretically analyzed. And second, their running times are usually low. The disadvantage is that a nonplanar DAG needs to be converted into a planar graph.

A. Node placement

Node placement is the main step that determines the quality of the graph and other steps depends on it. In the node placement step nodes are placed in a two-dimensional rectangular grid. Node placement step is divided into two stages. In the first stage all nodes are treated to be the same size each of them put in one grid cell. The second stage could be used alone, but obtaining an initial approximation with the unit size nodes often results in better layouts. The idea of using a grid for node placement (although for straight-line drawings) together with simulated annealing is used in drawing of biochemical networks [12, 13, 14]. But in our algorithm we extend it with repeated global compaction steps that allow escaping from the many local minima of the optimization problem. We will apply the node placement step on the graph in Figure1; we will get a different graph by varying the place of some node or by swapping some of them.

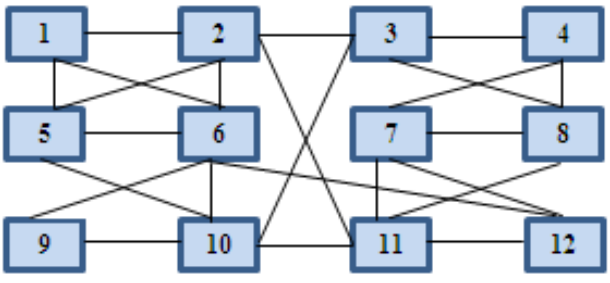

Fig.1. Example of rectangular graph

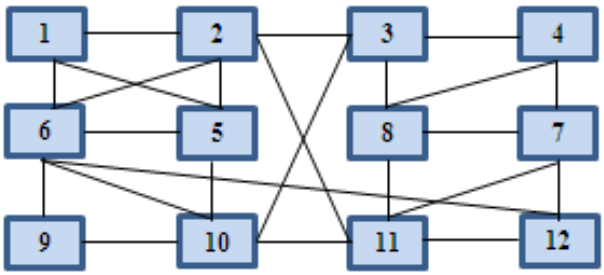

Fig.2. Node placement

B. Edge routing

In this step the edges will represented by orthogonal polylines connecting the associated nodes. We will find routes for edges that are short and with few bends by using quadratic programming [15]. We consider graph G = (V, E) with a node set V and edge set E and the minimum width w_i and height h_i of each node. In output the algorithm gives the top-left corner (x_i, y_i) of each node. To deal with nodes of different sizes we need to calculate the size of a grid cell. We assume the grid cell to be a square of side length c which is calculated as

$$C=\begin{cases} L_{max} & \text{if } L_{max} < 3L_{min} \\ \dfrac{3L_{min}}{2} & \text{if } 3L_{min} \le L_{max} < 15L_{min} \\ L_{max} & \text{if } 15L_{min} \le L_{max} \end{cases} \quad (1)$$

where L_{min} = min (min $(w_i + \delta)$, min $(h_i + \delta)$) and L_{max} = max (max $(w_i + \delta)$, max $(h_i + \delta)$). The main case of c is the middle one. The first case is chosen when all nodes are of a similar size and we define c such that all boxes take only one grid cell for more pleasant results. The third case prevents excessive memory usage in case of widely different node sizes. When nodes are placed in the grid, they are given integer coordinates and sizes. The top-left corner of a node v_i in the grid will be denoted by (x'_i, y_i'). Its width in the grid w'_i is calculated as $\left\lceil \frac{wi + \delta}{c} \right\rceil$. Its height in the grid h'_i is calculated as $\left\lceil \frac{hi + \delta}{c} \right\rceil$. We can use different functions for the edge length to be minimized. Common examples include Euclidean or Manhattan distance. To deal with nodes of different sizes better, we use a distance function $d(v_i, v_j)$ between two nodes v_i and v_j defined as follows:

$d(v_i, v_j) = de(v_i, v_j) + \frac{1}{20}$ min

$$\left(\left| \frac{x_i^c - x_j^c}{w_i' + w_j'} \right| \left| \frac{y_i^c - y_j^c}{h_i' + h_j'} \right| \right) \quad , \quad (2)$$

where $de(v_i, v_j)$ is the Euclidean distance between the node rectangle borders and

$x_i^c = x_i' + 1/2\ w_i'$ and $y_i^c = y_i' + 1/2\ h_i'$ are center coordinates of the nodes.

The second addend helps to align node centers when the distance between their borders is approximately equal. The constant 1⁄20 was chosen experimentally to balance the need for short edges with alignment of node centers. the result is shown in Figure3.

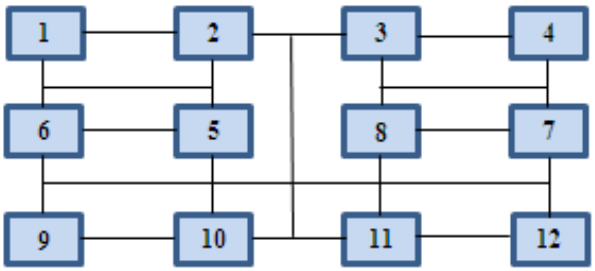

Fig.3. edge routing

C. *Normalization*

In this step we remove overlaps between nodes while minimizing the node movement. The normalization phase consists of two steps:

1- Normalization: for planar representation vertices with degree greater than 4 are changed into new multiple vertex representation, thus changing into normalized planar representation with degree less than 4.

2- Creating an orthogonal graph.
Applying the normalization on figure1 we get the figure 4.

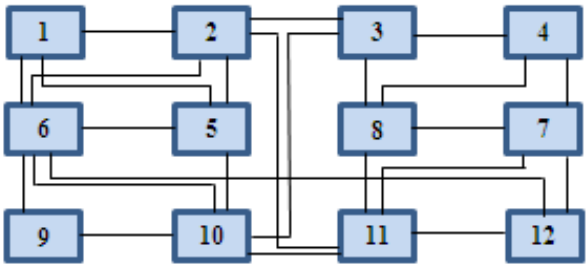

Fig.4. normalization

III. THE PSEUDOCODE OF THE ALGORITHM

1 Initialize the grid of size 5√|V | × 5√|V | Put nodes randomly into grid (treat them all 1 × 1 sized);
2 compactionDir=true;
3 iteration Count=90√|V |;
4 T=2√|V |;
5 k= $(0.2/T)^{1/\text{iteration Count}}$;

```
6   for (i=0; i<iteration Count/2; i++) do
7       for (j=1; j ≤ |V |; j++) do
8           x=neighboursMedianX (v_j) + random (−T, T);
9           y=neighboursMedianY (v_j) + random (−T, T);
10          Put v_j near (x, y);
11          if v_j has not changed its place from the previous
                        iteration then
12              Try to swap v_j with nodes nearby;
13          end
14      end
15      if iteration Count mod 9 == 0 then
16          compact (compactionDir, 3, false);
17          compactionDir=! compactionDir;
18      end
19      T=T·k;
20  end
21  compact (true, 3, true);
22  compact (false, 3, true);
23  for (i=iteration Count/2+1; i<iteration Count; i++) do
24      for (j=1; j ≤ |V |; j++) do
25          x=neighboursMedianX (v_j) +
                    random (−T·w'_j,T·w'_j);
26          y=neighboursMedianY (v_j) +
                    random (−T·h'_j, T·h'_j);
27          Put v_j near (x, y);
28          if v_j has not changed its place from the previous
                        iteration then
29              Try to swap v_j with nodes nearby;
30          end
31      end
32      if iteration Count mod 9 == 0 then
33          compact_ compactionDir,
            max (1, 1 + 2(iterationCount−i−30), false);
                        0.5iterationCount
34          compactionDir=! compactionDir;
35      end
36      T=T·k;
37  end
```

As the first steps, the grid of size 5√|*V* | × 5√|*V* | is initialized and nodes are randomly placed in the grid. The first stage of the algorithm (lines 6–20) treats each node of size 1. The grid is dynamically expanded during layout, if required. The algorithm works in iterations and the number of iterations *iteration Count* is taken proportional to √|*V* |. At each iteration, local optimization is performed that decreases the total edge length. The optimization process is based on the simulated annealing idea. It requires the notion of temperature *T* which influences how much node positions are perturbed by randomness. The starting temperature *T* is set equal to 2√|*V* | to allow nodes to be placed almost everywhere initially. The temperature is smoothly reduced by a cooling coefficient *k* until it reaches the lowest temperature *Tmin*; we take *Tmin* equal to 0.2. The cooling coefficient *k* is calculated in line 5 such that *T* reaches *Tmin* in *iteration Count* iterations. To perform local optimization, every node is moved to a location

that minimizes the total length of its adjacent edges. We use a heuristic to calculate this location approximately. Calculating the optimal location is expensive and actually is not needed since the added random displacement disturbs it anyway. We calculate an initial estimate to node's position (x,y) that minimizes the Manhattan distance to the adjacent nodes. Such point is found as a median of the neighbors' centers. A random displacement proportional to the temperature T is added to that point. Then we search the closest place to (x,y), where vj can be put (line 10). We calculate the Manhattan distance d of the closest free place to (x,y). Then we check all cells within Manhattan distance $d + 1$ from (x,y) and choose the position with the least total edge length according to (2) to place v_j. If this place is different from the location of vj from the previous iteration, we leave the node there. Otherwise, we try to swap it with the nodes nearby. We do this by checking the nodes residing in adjacent grid cells to vj . For each of these we calculate the gain of the total edge length if we swap the adjacent node with v_j. If the gain is positive we swap the nodes. Compaction is performed every 9-th iteration each time changing direction (lines 15–18). The variable *compactionDir* defines direction in which compaction is be performed, *true* for horizontal direction *false* for vertical. Compaction is performed by function compact. Compaction is done separately in horizontal and vertical directions. Let us consider the horizontal direction; the vertical one is similar. The relative ordering is expressed as a visibility graph. A visibility graph is a directed graph with the same set of nodes V but with a different set of edges S. There is a directed edge $(i, j) \in S$ in the visibility graph if and only if $x'_j > x'_i$ and it is possible to connect nodes v_i and v_j with a horizontal line segment without overlapping any other node. The visibility graph can be constructed with a sweep-line algorithm in time |V| log |V| but in our case we can extract it directly from the grid in time proportional to the number of grid cells.

We construct the following optimization problem

$$\text{minimize} \quad \sum_{i,j \in E} (Z_i + \tfrac{1}{2}W_i' - (Z_j + \tfrac{1}{2}W_j'))^2 \qquad (3)$$

$$\text{subject to } z_j - z_i \geq d_{ij}, (i, j) \in S$$

where $d_{ij} = \gamma \cdot w'_i$ and $\gamma \geq 1$ is a coefficient that defines how much empty space will be left between nodes. To obtain the maximum compaction we should set $\gamma = 1$. Such setting is desirable at the final few iterations of the algorithm but otherwise using $\gamma > 1$ leaves some empty places between nodes giving additional freedom for node movement to find a better solution.

The parameter *expand* is true if boxes need to be expanded to its real sizes, otherwise it is false. In line 16 compaction is done with $\gamma = 3$ and direction is changed after every compaction (line 17). The temperature T is reduced at the end of the iteration (line 19). In lines 21 and 22 switching from the first stage to the second is done by performing compaction with the new node sizes. The second stage (lines 23–37) is similar to the first one, only all boxes are treated with their prescribed sizes. Searching for a place for a node has to check if all grid cells under a larger node are free. This modification influences node swapping – there may be cases when adjacent nodes of different sizes cannot be swapped. This is the main motivation why the first stage with unit node sizes is beneficial. In line 33 compaction is done with gradually decreasing γ which becomes 1 in the last 3 compactions. In this way the available free space for node movement is gradually reduced giving more emphasis to node swapping.

IV. RESULTS

In this paper, we tested many types of graphs, to generate an optimal graph by applying the GBA. The algorithm produces pleasant drawings with small area and low number of crossings and edge bends. Where the improvement rate reached 97% in it. But, when we test the quality and performance of our algorithm we run it on three automatically generated graph classes – partial grids, random trees and random graphs. The results for the partial grid graphs are shown in Fig. 5. A partial grid graph is a square grid with the specified number of nodes where 10% of nodes are randomly removed. The results show that the algorithm with all the initial placement methods produce a planar layout of small instances (up to about 1000 nodes) but further only force-directed initialization is able to recover the graph structure correctly, breadth first search (BFS) initialization being the worst. It has to be mentioned that, if we increase the number of iterations of the BFS case to match the random case, we obtain drawings of similar quality. But our intention for the BFS method was to check whether we can improve running time with a better initialization. Tests showed that BFS initialization does not give any advantage over the random one. The quality on tree graphs is similar for all three modifications (Fig. 6). None is able to produce completely planar drawings of larger instances, although the crossing count is small. The BFS method has slightly more crossings than the other two. Random graphs are generated by including randomly chosen node pairs as edges in the graph with density $|E| = 1.2|V|$. The quality on random graphs is similar for all three methods (Fig. 7). That is expected since random graphs cannot be drawn with significantly less crossings than any of these methods produce.

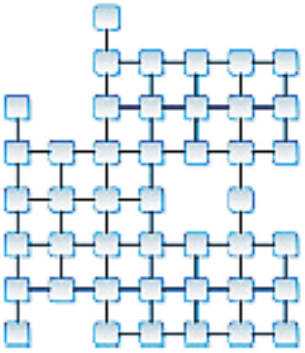

Fig.5. partial grid graph

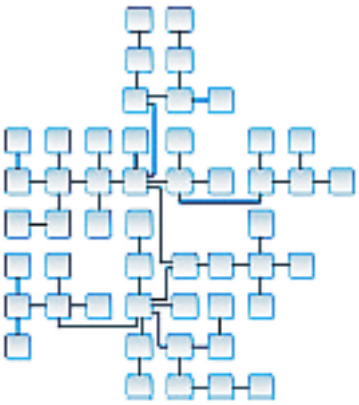

Fig.6. tree graph

Fig.7. random graph.

V. CONCLUSIONS

In this paper, we present a new method for orthogonal graph drawing by using grid-based algorithm that minimize edge crossings or edge bends to produce the best drawing quality in the least running time.

ACKNOWLEDGMENT

I want to thank my family that helped me along the journey of my studies and in the preparation of this research and I hope from God to care for them and protect them.

REFERENCES

[1] Di Battista, G., Liotta, G., Vargiu, F.: Spirality and optimal orthogonal drawings. SIAM J. Comput. **27**(6), 1764–1811 (1998)
[2] Biedl, T., Kant, G.: A better heuristic for orthogonal graph drawings. Comput. Geom. **9**(3), 159-180 (1998)
[3]. Liu, Y., Morgana, A., Simeone, and B.: A linear algorithm for 2-bend embedding's of planar graphs in the two-dimensional grid. Discrete Appl. Math. **81**(1–3), 69–91 (1998)
[4] Liu, Y., Marchioro, P., Petreschi, R., Simeone, and B.: Theoretical results on at most 1-bend embed ability of graphs. Acta Math. Appl. Sin. **8**, 188–192 (1992)
[5] Morgana, A., de Mello, C.P., Sontacchi, and G.: An algorithm for 1-bend embedding's of plane graphs in the two-dimensional grid. Discrete Appl. Math. **141**(1–3), 225–241 (2004)
[6] Tamassia, R.: On embedding a graph in the grid with the minimum number of bends. SIAM J. Comput. **16**(3), 421–444 (1987)
[7] A. Papakostas and I. G. Tollis, Efficient Orthogonal Drawings of High Degree Graphs, Springer Verlag New York Inc, 2000
[8] Thomas Bläsius · Marcus Krug · Ignaz Rutter · DorotheaWagner, Orthogonal Graph Drawing with Flexibility Constraints, Springer Science Business Media New York, 2012
[9] Xiao Zhou and Takao Nishizeki, Orthogonal Drawings of Series-Parallel Graphs With Minimum Bends, Springer-Verlag Berlin Heidelberg, 2005
[10] Michel Neuhaus and Horst Bunke, A Quadratic Programming Approach to the Graph Edit Distance Problem, Springer-Verlag Berlin Heidelberg, 2007.
[11] Tim Dwyer, Kim Marriott, and Peter Sbarski, Hi-tree Layout Using Quadratic Programming, Springer-Verlag Berlin Heidelberg, 2010.
[12] Kojima, K., Nagasaki, M., Miyano, S.: Fast grid layout algorithm for biological networks with sweep calculation. Bioinformatics **24**(12), 1433–1441 (2008)
[13] Kojima, K., Nagasaki, M., Miyano, S.: An efficient biological pathway layout algorithm combining grid-layout and spring embedder for complicated cellular location information. BMC Bioinformatics **11**, 335 (2010)
[14] Li, W., Kurata, H.: A grid layout algorithm for automatic drawing of biochemical networks. Bioinformatics **21**(9), 2036–2042 (2005)
[15]Lengauer, T.: Combinatorial algorithms for integrated circuit layout. John Wiley and Sons Inc., New York (1990)