The Three-Phase Method: A Unified Approach to Orthogonal Graph Drawing *

Therese C. Biedl[1] and Brendan P. Madden[2] and Ioannis G. Tollis[3]

[1] McGill University, 3480 University St. #318, Montreal, Quebec H3A2A7, therese@cgm.cs.mcgill.ca
[2] Tom Sawyer Software, 804 Hearst Avenue, Berkeley, CA 94710, bmadden@tomsawyer.com
[3] Dept. Computer Science, Univ. of Texas at Dallas, Richardson, TX 75083, tollis@utdallas.edu

Abstract. In this paper, we study orthogonal graph drawings from a practical point of view. Most previously existing algorithms restricted the attention to graphs of maximum degree four. Here we study orthogonal drawing algorithms that work for any input graph, and discuss different models for such drawings. Then we introduce the three-phase method, a generic technique to create high-degree orthogonal drawings. This approach simplifies the description and implementation of orthogonal graph drawing, and can be applied to global as well as interactive and incremental settings.

1 Background

In recent years, graph drawing has created intense interest due to its numerous applications. In networking and database applications, graph drawings serve as a tool to help display large diagrams efficiently. Different drawing styles have been investigated (see [4, 5, 12, 21] for overviews). One drawing technique, called orthogonal drawing, routes edges along the rows and columns of an underlying rectangular grid. Specific uses of orthogonal graph drawings include Entity Relationship (ER) Diagrams and large industrial schematics. The goal is to obtain aesthetically pleasing drawings. Common objectives include small area, few bends, and few crossings.

A drawing cannot be understood clearly if two edges overlap. Therefore, a feasible orthogonal drawing, with nodes drawn as points, is possible only if the maximum degree of the graph is at most four. Many orthogonal drawing heuristics have been developed for such graphs, see for example [2, 16, 18].

* This work was performed while the first author was working at, and the third author was consulting with Tom Sawyer Software. It was, in part, funded by the NIST under grant number 70NANB5H1162. This paper is part of a series about the orthogonal library of the Graph Layout Toolkit produced by Tom Sawyer Software. Patent on these and related results is pending.
These results are part of a Ph.D thesis of the first author at Rutgers University under the supervision of Prof. Endre Boros.

If the degree of a node v is larger than four, then more than one grid-point must be assigned to v. To maintain the semblance to a point, one uses a box which should not be too large. We discuss different models of high-degree drawings in Section 2.

There are different approaches to orthogonal graph drawing with high degree. One approach is to modify the input graph until it has maximum degree 4, by splitting nodes into chains or cycles of nodes. Then an algorithm for 4-graphs is applied, possibly with some modification to ensure that split nodes are drawn as boxes. This approach was taken in the GIOTTO-system [19] based on the algorithm by Tamassia [18], and in the algorithm by Biedl and Kant [2]. Its main disadvantage is that we have to decide on the ordering of the edges around the node to split it, which imposes unwanted structure. Additionally, the boxes are often rather large relative to the degree of a node.

Another approach is to directly assign boxes to nodes. Again there arises the difficulty to determine which edge should attach where along a side of the box. Also, to increase space for adding a box, previously existing boxes may be stretched, so control over the dimensions of the boxes is hard to achieve. Most algorithms for visibility representations, i.e., orthogonal drawings in which all edges are drawn as straight lines, work in this fashion, see for example [17, 20]. Not every graph has a visibility representation.

Only recently have there appeared algorithms for orthogonal drawings of any input-graph [7, 15]. These papers present algorithms where each edge has at most one bend. The first paper achieves a minimum number of bends (under the assumption that each edge may bend at most once) for planar graphs in a specialized model. The second paper presents an algorithm for general graphs with area guarantees of at most $m \times m/2$.

We present a third approach to create high-degree drawings, called the *three-phase method*. The main difference from the other approaches is that it first creates an infeasible drawing, called a *sketch*: we draw nodes as points and route edges with overlaps. Once all nodes and edges have been placed, we increase the nodes to boxes and determine the assignment of edges to particular port locations. This choice can now be made while avoiding crossings, since the routing of edges is known. We study this method in Section 3.

The three-phase method enables us to easily create an interactive framework, i.e., we allow the user to change the resulting drawing by adding or deleting nodes or edges, such that these changes result in only a small distortion of the picture. We study interactive drawings in Section 4, and end in Section 5 with conclusions and open problems.

We have implemented the three-phase method as part of the Graph Layout Toolkit, a family of general-purpose graph visualization libraries developed by Tom Sawyer Software. This toolkit is available with a license agreement for use in commercial applications, and may be obtained for a very small fee by academic institutions.[4]

[4] For information, see *http://www.tomsawyer.com* or contact *info@tomsawyer.com*.

2 Different models for high-degree drawings

In an orthogonal drawing, nodes should resemble points, hence it is undesirable that node dimensions are arbitrarily large. We present here three different models of orthogonal drawings, and discuss their advantages and disadvantages.

2.1 The Unlimited-growth model

In the first model, which we call the *unlimited-growth model*, no restrictions are placed on the dimensions of the boxes of the nodes. The GIOTTO system works in the unlimited-growth model [19], and so do all algorithms for visibility representations [17, 20]. The main advantage of the unlimited-growth model is that frequently we can stretch a node to cover the bend of an incident edge (see e.g. in [6]). In particular, planar graphs can be drawn without bends [17, 20]. The main disadvantage of this model is that nodes are not recognizable as points, and that we have no means of influencing the width and the height of the nodes.

2.2 The Kandinsky model

Fößmeier and Kaufmann introduced a model called the *Kandinsky model* [7]. In such a drawing, there are two different types of grid-lines. The grid-lines of a coarse grid are used to place the nodes. The grid-lines of a finer grid serve to allow edges to attach on each side of a node.

The Kandinsky model has many appealing features. It is possible to draw the nodes such that the boxes are aligned and have the same size, thus the concept of points has been generalized. The Kandinsky model is probably the best choice for drawings with uniform node dimensions. A disadvantage of the Kandinsky model is that it is somewhat wasteful in terms of bends and grid-size. There exists a graph that must be drawn with $m-4$ bends in the Kandinsky model, but that can be drawn without bends in another model.

2.3 The Proportional-growth model

The unlimited-growth model permits drawings with few bends, but may result in large node dimensions. The Kandinsky model permits control over the node dimensions, but may do so at the cost of introducing bends. Therefore we developed a third model for high degree orthogonal drawings, which we refer to as the *proportional-growth model*. In this model, we allow nodes to grow in size, but the growth must be reasonable.

Precisely, assume that we are given a drawing Γ. For each node, let $r(v)$ be the number of edges that attach on the right side of the box of v in Γ. Similarly, we define $l(v), t(v), b(v)$ for the other three sides. Our condition can then be formulated as follows:

Definition 1. A two-dimensional drawing is said to be in the *proportional-growth model*, if for each node v, the width of the box of v is $\max\{1, t(v), b(v)\}$, and the height of the box of v is $\max\{1, r(v), l(v)\}$.

We have found the proportional-growth model to be a good model for practical purposes. The growth of a node is related to its degree, and therefore nodes are not too distorted. In fact, some users have remarked on the degree-related node-growth as a positive feature of a graph visualization library, since it immediately displays influential nodes in diagrams.

3 The three-phase method

In this section, we introduce the three-phase method. Besides the three main phases, which are illustrated in Fig. 1, there are pre-processing and post-processing steps. In Fig. 5 on Page 10, we show a flow chart of the process of creating an orthogonal drawing.

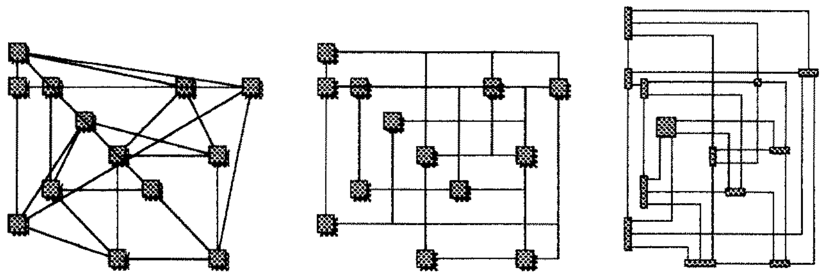

Fig. 1. A drawing after the three main phases.

The three-phase method is not an algorithm in itself, but it is a framework that unifies different approaches. It can be used for either of the three models. It is easily changeable, since an algorithm for a phase can be replaced by a better algorithm, if one becomes available in the future.

3.1 Pre-processing steps

Before starting the layout process, we apply the transformations needed to convert the graph into a *normalized graph*, i.e. a connected graph without reflexive edges and without nodes of degree 1. If the input graph is not connected, then we draw each connected component separately. Each reflexive edge is removed and added into the finished drawing close to the node. Each node of degree 1 is removed, and added into the finished drawing close to its neighbor.

3.2 Node placement

The first main phase of the algorithm is called *node placement*. In this and the following phase, we draw the nodes as points, not as boxes. During the node placement phase, we assign each node to a point in an $n \times n$-grid such that no two nodes are placed at the same point.

Any node placement is feasible, even though some can be achieved only by allowing more than one bend per edge. A node placement is said to be in *general position* if no two nodes are placed on the same grid-line. If G is a normalized graph, then for any node placement in general position, we can achieve an $m \times m$-grid and one bend per edge (see Section 3.6).

A node placement is an assignment of nodes to an $n \times n$-grid. Thus, it can be formulated as a 0-1 integer program with n^3 variables, and adapted to a number of objectives, such as for example the edge length. Solving this problem to optimality is prohibitively slow, therefore one has to develop heuristics.

Median placement We present here one heuristic for node placement, called the *median placement* strategy. To add a node v into a given drawing, let $w_1, \ldots, w_r$ be the embedded neighbors of v. A natural place for v is "in the center of its neighborhood", that is, roughly equidistant from $w_1, \ldots, w_r$.

Let w_j be placed at (x_j, y_j), and sort the neighbors of v such that $x_{i_1} \leq \ldots \leq x_{i_s}$ and $y_{j_1} \leq \ldots \leq y_{j_s}$. Let k be the median of $\{1, \ldots, s\}$ and define the *median center* as (x_{i_k}, y_{j_k}). Place v_{i+1} in a newly added row and column next to the median center. To compute the median center we need the relative order of x_j and y_j, but not the absolute values, and it can be found in $O(deg(v))$ time.

The median-placement should be used in an interactive setting if we want to add a new node into an existing drawing. A similar technique is one of the options used in the interactive technique described in [13].

3.3 Edge routing

The second phase of the algorithm is called *edge routing*. Assume that the node placement has been chosen. Now we want to decide on a route for each edge. During this phase, edge routes may intersect nodes or overlap each other, such conflicts will be removed during port assignment. For each edge, there are at most two possible routes with one bend.

Computing an edge routing with at most one bend per edge corresponds to a 0-1 integer program with at most $m + 2n$ variables, which can be adapted to objectives such as the desired dimensions of the nodes and the half-perimeter of the drawing. Solving this problem to optimality is prohibitively slow. A simple heuristic, using a Eulerian circuit, yields the following result. Details are omitted.

Lemma 2. *For any node placement in general position, there is an edge routing with one bend per edge such that at most* $\lceil \frac{\deg(v)}{2} \rceil$ *edges attach on each side of a node* v. *It can be found in* $O(m)$ *time.*

Another heuristic, based on randomized rounding of the optimal fractional solution of the 0-1 program yields the following result. Details are omitted.

Lemma 3. *For any normalized graph and any node placement in general position, if* HP_{IP} *is the optimal half-perimeter (among the edge routings with one bend per edge), and* HP_{RR} *is the randomized rounded half-perimeter, then*
$P\left(HP_{RR} > HP_{IP} \cdot \left(1 + \sqrt{4 \ln(4n)/n}\right)\right) < \frac{1}{n}$.

Routes with more than one bend Sometimes we want to route edges with more than one bend. In this case, we *triple the grid* before edge routing, i.e., in both orientations, we add one new grid-line before and after each used grid-line. Define the *off-grid-lines* of a node v as the grid-lines before and after the grid-line of v. To route an edge $e = (v, w)$ with more than one bend, we use the off-grid-lines at v and w, see Fig. 2. An off-grid-line may be used by more than one edge route, such overlap will be removed during port assignment.

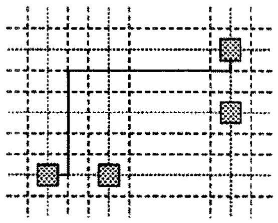

Fig. 2. An edge route with three bends. Grid-lines are dotted, off-grid-lines are dashed.

With three bends per edge, we can achieve the minimal possible node dimensions, again using a Eulerian circuit. Details are omitted.

Lemma 4. *For any node placement there exists an edge routing with three bends per edge such that at most $\lceil \frac{\deg(v)}{4} \rceil$ edges attach on each side of a node v. It can be found in $O(m)$ time.*

3.4 Port assignment

In the third phase, called *port assignment*, we increase node dimensions, adding new rows and columns if needed. Afterwards, each node v has a number of intersections with grid-lines, these places are called the *ports* of v. We assign a port to each endpoint of an edge such that no two edges overlap.

Such an assignment is not always possible without adding bends to edges. We will study in the following sufficient conditions for the existence of a port assignment without additional bends. Then we study how to achieve these conditions by adding bends and changing the node placement. Port assignment is done for one grid-line at a time, after the feasibility of port assignment has been assured for all grid-lines. Thus, during port assignment for a row, the port assignments in the columns stay unchanged, and vice versa.

One node per grid-line Assume row r contains only one node v. We add $\max\{1, r(v), l(v)\} - 1$ new rows after r, and extend v to cover these rows. We assign the edges to ports of v in such a way that no two incident edges on one side intersect. The resulting drawing is in the Kandinsky model.

Hamiltonian paths If the graph induced by the nodes in one row contains a hamiltonian path, then port assignment is possible such that the resulting drawing is in the proportional-growth model. Details are omitted, see Fig. 3.

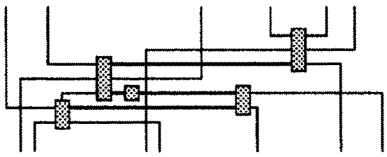

Fig. 3. Port assignment if the row graph has a hamiltonian path (shown in thick lines).

Non-overlapping trees If the graph of the nodes and bends in one row contains a rooted spanning tree such that for any two siblings v and w, the column-intervals spanned by the descendants of v respectively w are disjoint, then port assignment is possible such that the resulting drawing is in the proportional-growth model. Details are omitted, see Fig. 4.

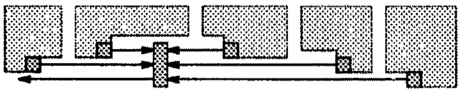

Fig. 4. Port assignment if the row graph contains a non-overlapping tree.

Resolving conflicts Port assignment is not always possible, and we also may not want to spend the time to find out, since testing some of the conditions is $\mathcal{NP}$-complete. Hence, we proceed as follows: Pick a condition that we would like to satisfy. Apply a simple heuristic to determine whether the condition is satisfied (it may erroneously report that it is not). If the condition is not satisfied, determine edges and nodes that prevent the condition, these are the *conflicts.*

Conflicts can be resolved in two possible ways. One simple solution is to reroute a conflicting edge with two or three bends. Alternatively, we can split the nodes in one grid-line into two groups, and move one of the groups to a newly added grid-line. This maintains the property that every edge is routed with at most one bend. Details are omitted.

3.5 Post-processing steps

There are two types of post-processing steps. First, we have to undo the changes of the pre-processing step, i.e., add previously removed trees and reflexive edges and combine the drawings of connected components. Secondly, we improve the drawing, by applying compaction techniques from VLSI-design, see the book by Lengauer [9].

3.6 Bounds

Let $h(v)$ and $w(v)$ be the height and width of v, so in the proportional-growth model, $h(v) = \max\{1, r(v), l(v)\}$ and $w(v) = \max\{1, t(v), b(v)\}$. We use the values as updated after the port assignment (they might change if we re-route an edge or re-locate nodes). In a drawing with at most one bend per edge, each used grid-line contains a node, therefore the total height cannot be more than $\sum_{v \in V} h(v)$. If (v, w) is an edge drawn without bend, then its row is used for both v and w. So if t_h is the number of horizontal straight edges, then the height is $\sum_{v \in V} h(v) - t_h$. Similarly one shows a bound for the width.

Lemma 5. *If Γ is a drawing with at most one bend per edge, and with t_h and t_v straight horizontal and vertical edges, respectively, then the height is at most $\sum_{v \in V} h(v) - t_h$, and the width is at most $\sum_{v \in V} w(v) - t_v$.*

Two important results follow from this lemma and Lemmas 2 and 4.

Theorem 6. *For any normalized graph G and any node placement in general position, we can find an edge routing and port assignment such that the resulting grid-size is $m \times m$ and each edge has one bend.*

Theorem 7. *For any normalized graph G and any node placement, we can find an edge routing and port assignment such that the resulting side-length of the grid is at most $\frac{3}{2}m + \frac{3}{4}n$, the half-perimeter is at most $3m + n$, and each edge has at most three bends.*

4 Interactive drawings

In this section we show that the three-phase scheme is helpful for developing efficient interactive high-degree orthogonal drawing algorithms. In an interactive setting we are given a legal orthogonal drawing, and an operation such as insert a node, insert an edge, delete a node, delete an edge, or move a node. The objective is to perform the desired change without major disturbance to the existing drawing, in order to preserve the "mental map" [11]. Algorithms for interactive orthogonal drawing were presented in [10, 13, 14].

The main problem in interactive drawing is to detect sufficient existing space or increase the space for adding edges and nodes, and to delete superfluous space. We attack this problem by uncompressing and then again compressing the drawing with every interactive change. Also, to find a suitable place to add grid-lines, if needed, we revert the valid drawing back into the sketch-status with every change. See Fig. 5 for a flow-chart of the interactive process.

Thus, we need only describe the changes to the sketch in the five possible interactive operations. For each, there are only a few affected grid-lines of the sketch, and only for these grid-lines do we undo the port assignment.

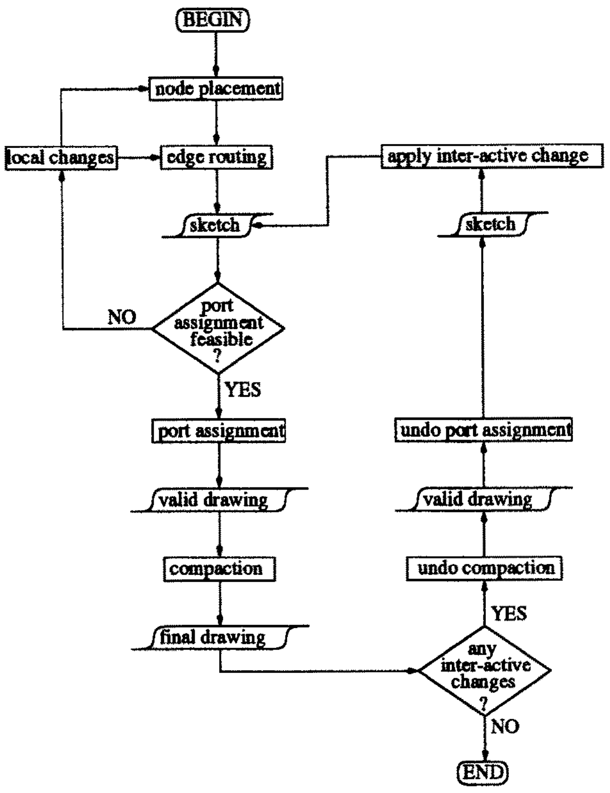

Fig. 5. The flow-chart of the three-phase method.

Delete an edge If we delete an edge $e = (v, w)$, the affected grid-lines are the ones that contain a segment of e. We delete all segments of e, and test for each affected grid-line whether it still contains elements; if not, we delete it. Then we try to join each affected grid-line with another grid-line, if this is feasible for port assignment.

Delete a node If we delete a node v with incident edges $e_1, \ldots, e_d$, the affected grid-lines are those that contain some segment of some edge $e_1, \ldots, e_d$, and both grid-lines of v. We delete the edges $e_1, \ldots, e_d$ as described above. Then we delete v, and check whether the grid-lines containing v are now empty; if so, we delete them. Then we merge each affected grid-line with another grid-line, if this is feasible for port assignment.

Insert an edge If we insert an edge $e = (v, w)$, the affected grid-lines are both grid-lines of both v and w. We test whether we need a bend for e, and if so, where to place it.

We first try to route e without bend. Thus, if v and w share a grid-line, then we draw e as a straight line, if this is feasible for port assignment. Otherwise, we try whether v and w are in neighboring grid-lines. If so, we test whether it is feasible for port assignment if we join these two grid-lines and add edge e to it. If not, then we try whether v and w are in neighboring grid-lines in the other direction and proceed similarly. If neither of these options is successful, then we conclude that e must be drawn with a bend.

If e must be drawn with a bend, then we have two possibilities to route e. Possibly a preference for the routing has been indicated (see later in "Moving a node"). Otherwise, we determine a route with the chosen heuristic for edge routing.

Insert a node If we insert a node v with incident edges $e_1, \ldots, e_d$, where $e_i = (v, w_i)$, the affected grid-lines are the horizontal and vertical grid-lines of w_i, $i = 1, \ldots, d$.

We first need to find a good placement for node v. This could have been indicated already (see "Moving a node") by listing the closest neighbors w_S, w_N, w_E and w_W in either direction. We add a new row between the rows of w_S and w_N, and a new column between the columns of w_E and w_W, and place v in it. If no placement was indicated, then we choose one, for example using the median placement discussed in Section 3.2, and add a new row and column there. We route $e_1, \ldots, e_d$ as described in the previous subsection.

Moving a node Moving a node is probably the most important interactive change, and it builds on top on the previous operations. In order to move a node v, we determine the nodes w_N, w_S, w_E and w_W that are closest to the desired new place of v in the four directions. We also store for each incident edge of v whether it attached to v horizontally or vertically.

We delete v and all its incident edges, and then re-insert v, using the information about the four closest neighbors. Then we insert the incident edges of v with the same bend placement as before, if possible.

Worst-case bounds At any time, every edge has at most one bend, so we use at most one row and one column for each edge. For some nodes, all incident edges may attach in one orientation, say horizontally, thus one column is not accounted for by any edge. Consequently, the total grid-size may be at most $m + n$ in either orientation.

Theorem 8. *Let G be a graph that is changed interactively, through insertions, deletions, and move operations. Under any sequence of operations, we can maintain a drawing of G with only local changes. At any time, the grid-size is at most $(m + n) \times (m + n)$, and there is at most one bend per edge, where n and m are the current number of nodes and edges.*

5 Conclusion

In this paper, we introduced the three-phase method for creating orthogonal drawings of graphs with high degrees. This method breaks the layout problem into three separate phases, and thus permits detailed study of each. With this method, we can create drawings in a newly defined model, called proportional-growth model, which allows us to save bends and at the same time maintains box dimensions of nodes within reasonable bounds.

The three-phase method is general and flexible, and has numerous applications, including the following results for normalized graphs:

- Any simple graph can be embedded in an $\frac{m+n}{2} \times \frac{m+n}{2}$-grid with at most 1 bend per edge [3].
- Any simple graph can be embedded with half-perimeter $2m-n$ with $m-n+1$ bends and at most one bend per edge [1].
- Any simple triconnected planar graph can be embedded without crossings in an $(m-n+1) \times \min\{m-n, \frac{m}{2}\}$-grid with $m-n$ bends and at most 1 bend per edge [3].
- For any given node order $\{v_1, \ldots, v_n\}$, we can build a drawing incrementally by adding nodes in this order, such that with any addition there are only local changes. At any point in time, the grid-size is at most $(\frac{m}{2}+n) \times (\frac{2}{3}m+n)$ and there is one bend per edge [3].
- *Relative node placement constraints*: For any pair of nodes v, w, we can specify whether v should be below w, or whether v should be to the left of w, or both, as long as these constraints are not contradictory in themselves. The drawing has grid-size $m \times m$ and at most 1 bend per edge [1].
- *Port specifications*: We can specify constraints of the form "edge (v, w) should attach at the top side of node v, and it should be the kth of the edges attaching there" [1].

One other advantage of the three-phase method is its adaptability: the algorithm for a phase can be replaced by a better algorithm, when one becomes available in the future. We expect further improvements on the area bounds and number of bends, based on this method.

As for open problems, we plan to develop improved node placement schemes. In theory, a node placement in general position gives the best known worst-case bounds (compare Theorem 6 with Theorem 7). But if we place more than one node into a grid-line, and if this does not create a conflict for port assignment, then we save one bend and therefore also one grid-line (Lemma 5). So a node placement scheme is best in terms of area if it permits many nodes in one grid-line, while satisfying a necessary condition for port assignment. How can such a node placement be found? How can node placements be found that ensure a small number of crossings?

References

1. T. Biedl. *Orthogonal Graph Visualization: The Three-Phase Method With Applications*. PhD thesis, RUTCOR, Rutgers University, May 1997.
2. T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. In *2nd European Symposium on Algorithms*, volume 855 of *Lecture Notes in Computer Science*, pages 124–135. Springer Verlag, 1994.
3. T. Biedl and M. Kaufmann. Static and incremental high-degree orthogonal drawings. In *5th European Symposium on Algorithms*, Lecture Notes in Computer Science. Springer Verlag, 1997. To appear.
4. F. Brandenburg, editor. *Symposium on Graph Drawing 95*, volume 1027 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.
5. G. Di Battista, P. Eades, R. Tamassia, and I. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comp. Geometry: Theory and Applications*, 4(5):235–282, 1994.
6. U. Fößmeier, G. Kant, and M. Kaufmann. 2-visibility drawings of planar graphs. In North [12], pages 155–168.
7. U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In Brandenburg [4], pages 254–266.
8. G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996.
9. T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Teubner/Wiley & Sons, Stuttgart/Chicester, 1990.
10. K. Miriyala, S. Hornick, and R. Tamassia. An incremental approach to aesthetic graph layout. In *6th Intl. IEEE Workshop Computer-Aided Software Eng.*. 1993.
11. K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *J. Visual Languages and Computing*, pages 183–210, June 1995.
12. S. North, editor. *Symposium on Graph Drawing 96*, volume 1190 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.
13. A. Papakostas, J. M. Six, and I. Tollis. Experimental and theoretical results in interactive orthogonal graph drawing. In North [12], pages 371–386.
14. A. Papakostas and I. Tollis. Issues in interactive orthogonal graph drawing. In Brandenburg [4], pages 419–430.
15. A. Papakostas and I. Tollis. High-degree orthogonal drawings with small grid-size and few bends. In *5th Workshop on Algorithms and Data Structures*, Lecture Notes in Computer Science. Springer Verlag, 1997. To appear.
16. A. Papakostas and I. Tollis. A pairing technique for area-efficient orthogonal drawings. In North [12], pages 355–370.
17. P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientation of planar graphs. *Discrete Computational Geometry*, 1:343–353, 1986.
18. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal of Computing*, 16(3):421–444, 1987.
19. R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Systems, Man and Cybernetics*, 18(1), 1988.
20. R. Tamassia and I. Tollis. A unified approach to visibility representations of planar graphs. *Discrete Computational Geometry*, 1:321–341, 1986.
21. R. Tamassia and I. Tollis, editors. *DIMACS International Workshop, Graph Drawing 94*, volume 894 of *Lecture Notes in Computer Science*. Springer Verlag, 1995.