Journal of Graph Algorithms and Applications

http://www.cs.brown.edu/publications/jgaa/

vol. 4, no. 3, pp. 75–103 (2000)

Techniques for the Refinement of Orthogonal Graph Drawings

Janet M. Six *Konstantinos G. Kakoulis* *Ioannis G. Tollis*

CAD & Visualization Lab
Department of Computer Science
The University of Texas at Dallas
P.O. Box 830688, EC 31
Richardson, TX 75083-0688
{janet,kostant,tollis}@utdallas.edu

Abstract

Current orthogonal graph drawing algorithms produce drawings which are generally good. However, many times the quality of orthogonal drawings can be significantly improved with a postprocessing technique, called *refinement*, which improves aesthetic qualities of a drawing such as area, bends, crossings, and total edge length. Refinement is separate from layout and works by analyzing and then fine-tuning the existing drawing in an efficient manner. In this paper we define the problem and goals of orthogonal drawing refinement, review measures of a graph drawing's quality, and introduce a methodology which efficiently refines any orthogonal graph drawing. We have implemented our techniques in C++ and conducted experiments over a set of drawings from five well known orthogonal drawing systems. Experimental analysis shows our techniques to produce an average 37% improvement in area, 23% in bends, 25% in crossings, and 37% in total edge length.

Communicated by G. Liotta and S. H. Whitesides: submitted October 1998; revised November 1999.

Research supported in part by NIST, Advanced Technology Program grant number 70NANB5H1162 and by the Texas Advanced Research Program under Grant No. 009741-040.

1 Introduction

Graphs are used in numerous applications to represent many information structures: telecommunication networks, entity-relationship diagrams, data flow charts, object-oriented software design and much more [34]. *Graph Drawing* studies the automated layout of these structures onto a two or three dimensional space where each node is represented by a polygon or circle and each edge by one or more contiguous line segments. This visualization can take form in one of many conventions: e.g., polyline, straight-line, orthogonal planar, or upward [6, 7].

Orthogonal graph drawings represent nodes with boxes and edges with polygonal chains of horizontal and vertical line segments which reside on an underlying grid. A few key applications for this style of graph drawing are project management, PERT diagramming, object-oriented analysis, database navigation, and VLSI circuit design. Much research has been conducted in this area and various algorithms exist to produce orthogonal drawings of planar [1, 12, 18, 30, 31, 33], general maximum degree four [1, 25, 28], and general higher degree graphs [2, 16, 23]. An extensive experimental study was conducted by Di Battista et. al. [8] where four general purpose orthogonal drawing algorithms were implemented and compared with respect to area, bends, crossings, edge length, and running time.

Many papers have suggested ways of evaluating the "goodness" of a graph drawing (e.g., [9, 11, 19, 22, 32]). Ding and Mateti [9] formalize the quality of data structure diagrams with nine categories: visual complexity, regularity, symmetry, consistency, modularity, size, shape, separation, and tradition. Visual complexity is the measure of ease with which a structure is communicated and includes the factors ambiguity, geometric complexity, and recognizability. Ding and Mateti also present the importance of semantic and geometric recognizability. Of course graph drawers endeavor to develop algorithms which have high semantic recognizability, but this goal is very difficult since a set of humans may have multiple interpretations of any one drawing. The authors of [9] also propose that the geometric complexity (e.g. area, crossings, bends, edge lengths) of drawings should be low, respecting the concept that the simpler a drawing appears, the simpler it will be to interpret. The last of Ding and Mateti's categories addresses a very important issue which the standard quality measures of a graph drawing do not capture: tradition. People understand a drawing better if the information is presented in a manner which is familiar. This is very difficult to qualitatively measure in a general way since people of different backgrounds may expect to see information in alternate ways.

Tamassia, Di Battista, and Batini present an extensive study of the readability of diagrams in [32]. They suggest several specific quantitative measures of quality which include: mimimization of area, balance of the drawing with respect to the horizontal or vertical axis, minimization of bends, maximization of the number of convex faces, minimization of crossings, minimization of the difference in size of the largest and smallest nodes, minimization of the total length of edges, minimization of the maximum edge length, symmetry of children in hierarchies, uniform distribution of nodes, and the upwardness of hierarchic structures. The semantics of the drawing are addressed with the following set of constraints: place a set of nodes in the center of a drawing, specify certain nodes to be a specific size, place a set of nodes on the exterior of the drawing area, cluster a set of vertices, draw a set of nodes in a specific shape and place a sequence of vertices along a straight line.

The optimization of many of the above aesthetics and constraints is known to be NP-Hard [6]. Complicating this issue is the experience that maximizing one particular quality of a drawing causes another to be significantly poor since some of these qualities work against each other. Therefore most algorithms try to layout the graph in a manner which is good for some set of aesthetics.

Current orthogonal graph drawing algorithms produce drawings which are generally good. However, many times the quality of orthogonal drawings can be significantly improved with a postprocessing step. *Refinement* is a postprocessing methodology which can be applied to any orthogonal drawing. It improves the quality of a drawing by first analyzing it and then fine-tuning the drawing while keeping the majority of the layout intact. The result is a new drawing which has improved aesthetic qualities including area, bends, crossings, and edge length. Previous work includes compaction strategies [2, 31, 33] and movement of stranded nodes [13]. However, the scope of these postprocessing techniques is limited. A more comprehensive methodology is needed to further improve the aesthetic qualities of graph drawings. We presented a preliminary version of our refinement techniques in [29]. At the same conference, Fößmeier, Heß and Kaufmann discuss four techniques which reduce area, bends, and edge length, [15], yet they present no experimental results. However, most of the benefits they discuss would also be obtained by our techniques. We have focused on the development and implementation of several efficient refinement modules which work on *any* orthogonal drawing (including degree greater than four). An example of a drawing before and after refinement is shown in Figure 1.

There are two types of refinement: *interactive* and *static*. During the interactive refinement of drawings we must maintain the user's mental map [22] and are allowed to make only minimal changes. The requirements for this kind of refinement methodology are very similar to those of interactive graph drawing algorithms [3, 5, 14, 21, 22, 24, 26]. The fact that the user has already seen a drawing of a graph means that the refinement techniques must not make changes so drastic that pieces of the drawing are not recognizable. Static refinement fine tunes drawings for which we do not have to maintain the user's mental map: we are free to make any change in the embedding. Our tool performs static refinement on any orthogonal graph drawing. Using a subset of the modules, our tool can also perform interactive refinement. Certainly refinement cannot be a cure for a very poor layout because this would require the essential invocation of some other layout algorithm. Refinement fine-tunes an existing drawing by improving some layout qualities.

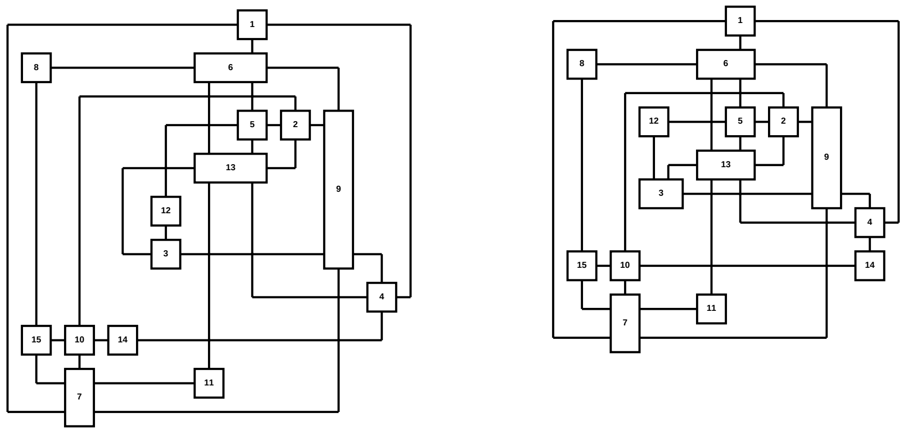


Figure 1: A graph as drawn with the GIOTTO component of Brown University's Graph Drawing Server on the left. The same drawing after refinement on the right (same scale). There is a 29% improvement in area, 13% improvement in the number of bends, 11% in the number of crossings, and 24% in the total edge length.

Our refinement techniques produced a significant 23% to 37% average improvement for each of the generally accepted quality characteristics area, bends, crossings, and total edge length in experiments over drawings from five algorithms. Since different applications require different classes of drawings and therefore need to focus on varying kinds of refinement, our system has the flexibility to vary the types and order of refinement modules called, so that a user may refine drawings in a manner specific for a particular application.

The remainder of this paper is organized as follows: we present techniques for the refinement of orthogonal graph drawings in Section 2, implementation details and the results of an experimental study in Section 3, and conclusions and future work in Section 4.

2 Refinement

During a survey of orthogonal drawings from a variety of sources, we repeatedly observed extra area, bends, edge crossings, and edge length caused by U-Turns in edges (as described in [21]), superfluous bends, poor placement of degree two nodes, two incident edges of a node crossing, nodes stranded very far from their neighbors, and unused space inside the drawing. See Figure 2 for examples.

Specifically, **U-Turns** are three contiguous edge segments which form a "U"

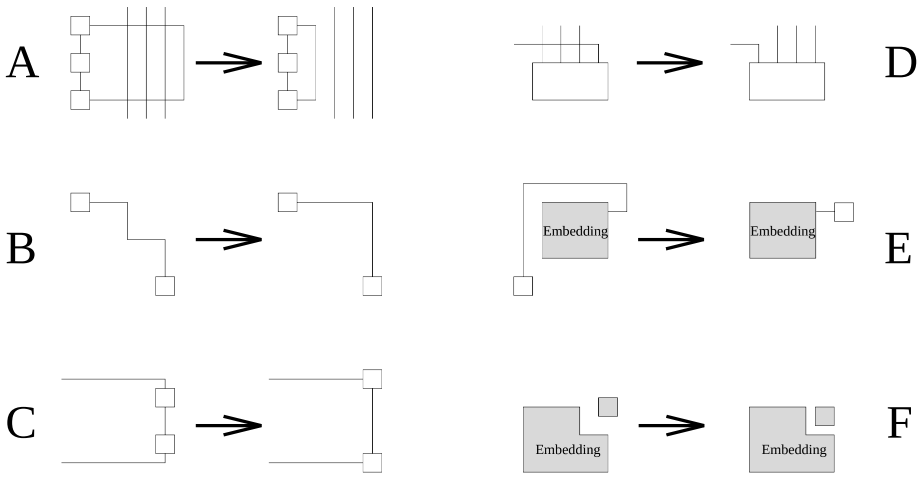


Figure 2: Examples of the problems we solve with a refinement technique: A: U-Turns, B: superfluous bends, C: poor placement of degree two nodes, D: self-crossings, E: stranded nodes, and F: extra area.

shape with the middle segment pulled far from the source and target nodes of those three segments. **Superfluous bends** are those which exist even if there is room in the drawing for an edge routing with fewer bends. Clearly U-Turns and superfluous bends can occur multiple times in edges which have four or more segments. Also, both U-Turns and superfluous bends can appear in the same edge. Poorly placed **degree two** nodes are those which are neither on a bend nor in the midst of its two incident edges. **Self crossings** are those which occur between two edges incident to the same node. Self crossings are divided into two categories: *near* and *far* self crossings. Near self crossings are those which appear north, south, east, or west of the node. Far self crossings appear northeast, northwest, southeast, or southwest of the node. A **stranded node** is a degree one node which could be placed closer to its neighbor.

Fixing a set of the above defined problems with a sequence of refinement steps will certainly reduce the visual complexity of the drawings, however we take our methodology one level further. We reduce the given graph into a simpler one. First we absorb all chains of degree two nodes into a super edge and then denote each degree one node to be a super node and determine the minimum distance needed between it and its neighbor (as is also done in [5, 23]). For example, the edge crossing between edges $(3, 13)$ and $(5, 12)$ in the left drawing of Figure 1 would not be a self crossing as defined above, but obviously it could be removed. After the reduction then the crossing between the super edge $(3, 13)$ and the edge $(3, 5)$ is discovered as a self crossing and can be removed. All refinement operations are performed on this simplified graph. The refinement techniques have been implemented to acknowledge the presence of super nodes and edges and place them in some appropriate manner. After refinement is complete the super edges are expanded in order to restore the graph to its original topology. The reduction operations allow our methodology to detect more of the above problems and therefore produce better quality drawings.

Procedure: Refine-Orthogonal-Graph-Drawing
Input: An orthogonal graph drawing, Γ, of a graph, G
Output: A new orthogonal drawing, Γ', of G with a lower visual complexity

1. Build the abstracted graph, G', of G, such that
   (a) Each chain of degree two nodes is abstracted into a super edge.
   (b) Each degree one node is denoted to be a super node and the minimum necessary distance between it and its lone neighbor is calculated. The minimum distance is directly proportional to the number of absorbed degree two nodes in the lone incident edge.
2. For each edge, e, in G'
   (a) If e contains a sequence of three edge segments which form a U-Turn edge, then pull in the middle segment of that sequence so that it is as close as possible to the tips of the U.
   (b) If e contains a sequence of three edge segments which have an extra bend and there exists room for a lower bend edge routing in the drawing then replace the current routing with the lower bend solution. This technique is very similar to the bend-stretching transformations of [33].
3. For each node, v, in G'
   (a) If v has a near self crossing, expand the node by one row or column in the appropriate direction and move the attachment point of the trouble edge to that new row or column. Place any abstracted degree two nodes so that they do not occlude any node or edge. If v has a far self crossing and is degree two, then place the node at the location of the crossing. Otherwise add one row or column and break the crossing into a knock-knee [20] edge routing. Both of these far self crossing solutions swap the attachment points of the crossing edges at v so that neither of the neighbor nodes is moved.
   (b) If v is supernode and the distance to its lone neighbor is greater than the minimum distance calculated in Step 1(b), (i.e. a stranded node) place the supernode as close to the neighbor node as space allows in Γ.

4. For each superedge, e, expand e to restore the degree two nodes and verify that each degree two node is either on a bend or in the midst of its two incident edges.

5. Perform a VLSI layout compaction [10, 17, 20] to remove extra space in the drawing. □

Many improvements may be made without increasing the area of the drawing, but allowing the addition of more area may enable refinement to significantly improve other aesthetics of the drawing. For example, adding a row or column may be necessary to remove a self crossing. However this allowance should be according to user requirements and must be parametrically defined. It is possible that more area, bends, crossings, or edge length will be added with some refinement step, however we have found with our experimental study that we still gain improvements in these quality measures globally. However, we still recommend that this refinement methodology be implemented with a set of parametric options tailored to user requirements. Also, the separate refinement techniques may be performed in any order.

Refinement is an evolving methodology: we are planning to implement additional modules for improving orthogonal drawings as we discover and develop further techniques.

3 Implementation and Experimental Results

3.1 Implementation

Refine-Orthogonal-Graph-Drawing has been implemented in GNU C++ Version 2.7.2 with Tom Sawyer Software's Graph Layout Toolkit Version 2.3.1 (GLT) and a TCL/TK graphical user interface. A set of experiments has been run on a Sparc 5 running Sun OS 4.3.1.

Many interesting and challenging issues were addressed during the implementation of our refinement procedure. First we needed a mechanism to search the space within the given drawing to move pieces of the drawing without occluding uninvolved nodes and edges. We represent the space of the drawing with a dynamic orthogonal grid structure in which rows and columns may be added at any point within the space. Nodes and edges of the drawing are represented with grid segments.

As mentioned in the previous section, we recommend the use of parametric options for allowing or disallowing the addition of area, bends, crossings, or edge length. Furthermore, it is not necessary for these parametric options to be binary. For example, adding x amount of area is allowable if y crossings are removed, where x and y are some thresholds. We do not expect that any single set of parametric options would be appropriate for every application. Therefore, the definition of this set of options should be decided by the implementor. In the experimental study presented in Section 3.2 we used a set of parametric options which would allow the addition of area, bends, crossings, and edge length in order to test the strength and weaknesses of the entire set of refinement techniques. We also allow the movement of edge attachment points. As shown with the results of this experimental study, Refine-Orthogonal-Graph-Drawing overall improves these aesthetics although they may be temporarily increased during a single step.

Each of the refinement modules can be viewed as a local search technique. The module which shortens U-Turn edges searches the grid starting at the row or column next to the endpoint of the first or third edge segment (whichever is placed closer to the middle edge segment) looking for a placement which does not occlude any nodes or edges. We search the grid toward the old placement until sufficient space is found. At worst, this will be the old placement. See Figure 3. It is important to note that the edges involved in the U-Turn may

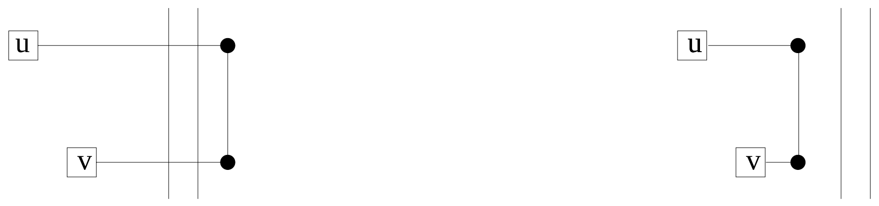


Figure 3: Fixing a U-Turn edge. The left illustration shows the nodes and edges in their original positions. The endpoints of the middle segment are shown with circles and the source and target of the U-Turn with boxes. The right illustration shows the final placement.

actually represent a chain of degree two nodes, therefore we must detect that situation and be sure to place those nodes only where there is sufficient space in the grid. This means that none of the relocated edge segments (or nodes, if this edge represents a chain of degree two nodes) can overlap any previously placed node or edge. Also, we examine each set of three contiguous segments in each edge so that we can catch more of the U-Turns. This is especially important when we are dealing with degree two chains. It is possible that we can remove bends with this refinement module. See Figure 4. For example, if there is room on the top side of node v in Figure 4 (a) for the attachment of edge (u, v) we can remove one bend. Of course, if this edge is a super edge, we must make sure that there is room for the placement of each of the degree two nodes in the grid. The illustration in Figure 4 (b) shows how multiple U-turns in a single edge can also be fixed.

The superfluous bends module examines each set of three contiguous edge segments in every edge. For each set, call one endpoint of the middle edge

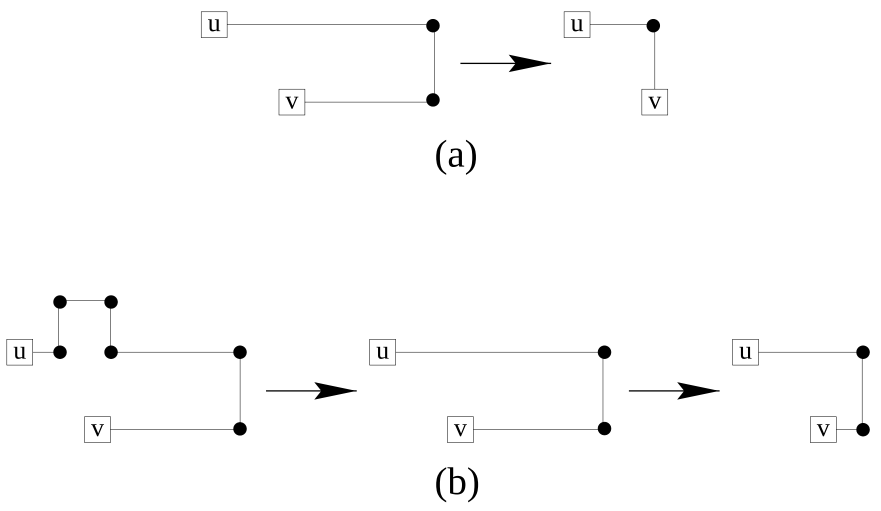


Figure 4: Removing bends with the U-Turn module. The illustration marked (a) demonstrates how one bend of a U-turn can be removed. The illustration marked (b) demonstrates how an edge with multiple U-turns is handled by Refine-Orthogonal-Graph-Drawing.

segment x and the other y. Define a and b to be the points shown in Figure 5. If space in the drawing allows, then place x at a or y at b. No occlusions are allowed. Again, it is possible that superfluous bends may appear multiple times in an edge, this is the reason for examining each set of three contiguous edge segments.

The self crossing module removes crossings as specified in Step 3 of our refinement procedure. Since the grid is dynamic, we insert the new gridlines inside the node to fix a near self crossing on the appropriate side and that automatically forces the node to grow. It is possible that a node will be grown multiple times. For far self crossings of a degree three or higher node, a row or column is inserted at that crossing (see Figure 6 for examples). If adding a row increases the size of some node, but adding a column does not, then our refinement procedure will use the solution which adds a column. The first far self crossing solution is for the case where the node is degree two while the second solution is for higher degree nodes. If the node is degree two, our procedure for refinement searches the space in the grid at the self crossing to determine if the there is enough space for the node to be placed there. If so, the node is moved and the endpoints of the incident edges are updated at the newly moved node. The positions of the neighbors are always maintained. The second solution adds two bends and therefore the superfluous bends refinement module

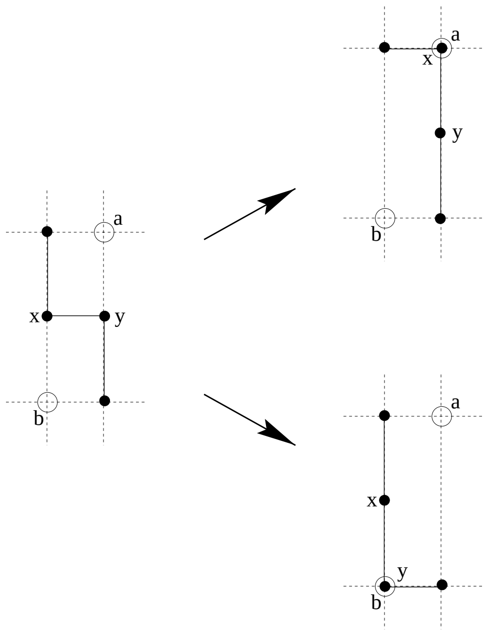


Figure 5: Superfluous bend solutions. The old edge routing is shown with a dashed line in the two illustrations on the right.

should be run on the new drawing to remove these extra bends if possible. The user has a parametric option for this module to allow the addition of rows and columns as necessary to avoid adding any crossings or to allow crossings to remain. This option gives the user the ability to decide to give priority to area or crossing reduction. We believe that reducing the number of crossings is paramount and chose the first parametric option for our experiments. In part, this decision was influenced by the study presented in [27] which showed the number of crossings to have a very significant effect on quality. This is certainly not to say that avoiding crossings will always cause the addition of area. In fact we found in our experimental study that in many drawings, the area will still be reduced while avoiding crossings. Also, note that the attachment points of the two crossing edges may be swapped so that the self crossing is removed while the positions of the neighbors are not changed. If this movement of the attachment points is not acceptable, the user can set a parametric option preventing this action.

The stranded node module searches the grid from the placement of the neighbor towards the old placement of the stranded node in order to find available space for the stranded node to be moved. At worst, this will be the old place-

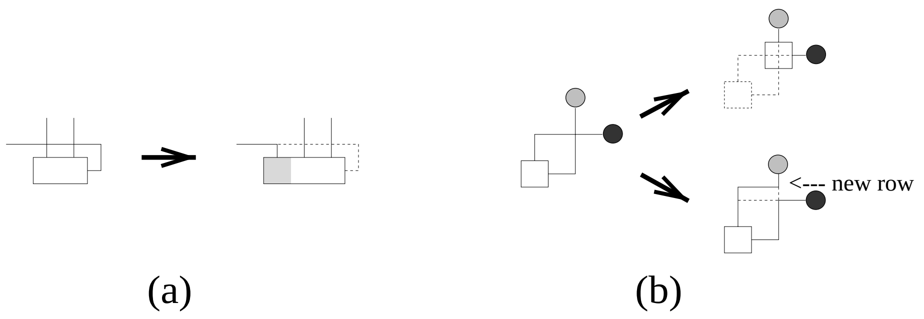


Figure 6: Near(a) and far(b) self crossing solutions. Old node placements and edge routings are shown with dashed lines.

ment. If the lone incident edge of the stranded node represents a chain of degree two nodes, then this process is iteratively carried out for degree two node from the neighbor node (i.e. the node with degree higher than two) to the stranded node. See Figure 7.

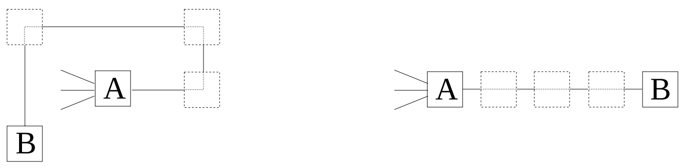


Figure 7: Fixing stranded node B with an incident edge that represents a degree two chain. The refined solution appears on the right. Notice that not only are we placing related nodes closer to each other, but also unwrapping a poorly routed set of edges.

The module which fixes poor placement of degree two nodes, first expands the super edges and then visits each newly restored degree two node to verify that it lies either on a bend or in the midst of its two incident edges. See Figure 8. These refinements reduce the number of bends and produce drawings with a better distribution of node placements for the incident edges. Since the refinement modules described in this paper can be applied in any order, if there are any more refinements to be done after the completion of this module, any degree two chains should be reduced again.

The implementation of the compaction module is inspired by one dimensional VLSI layout compaction [10, 17, 20]. A one dimensional graph-based compaction is performed once each for the horizontal and vertical directions. Compaction

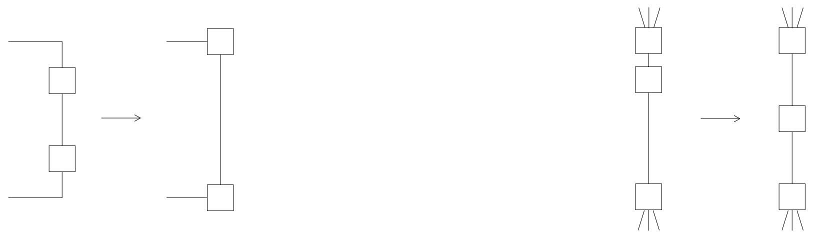


Figure 8: Refining poor placement of degree two nodes. The illustration on the left shows the movement of degree two nodes to remove bends while the right demonstrates a node placement with better distribution.

is a NP-Hard problem [20] and therefore efficient one-dimensional compaction techniques are not guaranteed to produce optimal solutions. Similar types of compaction modules have also been successfully used in [2, 31, 33].

It is recognized that different users may want different types of drawings and need to refine a specific aesthetic quality. Also different algorithms merit the use of different types of refinement: classes of orthogonal drawing algorithms inherently have strengths and weaknesses with respect to different aesthetic criteria. Static orthogonal graph drawing algorithms either planarize and then embed with a planar algorithm or proceed in an essentially incremental fashion. While the problems described in the previous section occur with all of the orthogonal algorithms surveyed, some types of problems occur more frequently in a particular class of orthogonal algorithms. For example, planarization algorithms have a tendency to have some very long edges and place nodes far away from their neighbors. Incremental algorithms tend to have superfluous bends. Our refinement methodology automatically detects these problems and fixes them regardless of the class of algorithm used to create the drawing. Furthermore, the quality of a drawing before refinement depends heavily on a chosen algorithm, and even more heavily on the implementation of that algorithm. Our refinement methodology fixes problems caused by both the algorithms and their implementations.

Our implementation communicates with the user via a graphical user interface which allows the user to perform a desired set of refinements in any order. This flexibility adds power to refinement in that the user can refine any orthogonal drawing in a manner which is application specific. Figure 9 shows the user interface of our tool. Note the nine square buttons on the left side of the interface. These buttons invoke the refinement modules. Our design and implementation allows the user to choose the types and order of refinement to be performed. For example, if the user would like to remove only the self crossings, then that button is chosen. The refined drawing quickly appears on the can-

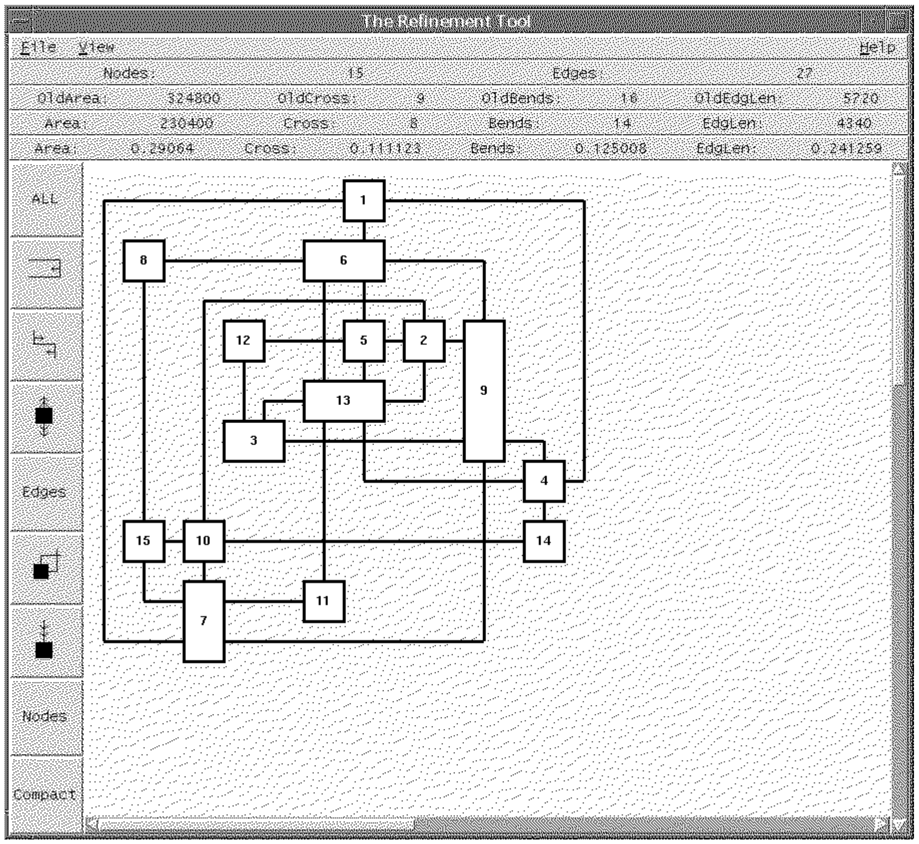

```
Nodes:        15                  Edges:        27
OldArea:  324800   OldCross:   9   OldBends:   16   OldEdgLen:   5720
Area:     230400   Cross:      8   Bends:      14   EdgLen:      4340
Area:    0.29064   Cross: 0.111123 Bends: 0.125008  EdgLen:  0.241259
```

Figure 9: The Graphical User Interface for The Refinement Tool. Buttons for individual and combined refinement types appear on the left while statistics for the drawing appear at the top. The user also has options to save postscript images and view the drawings at different scales. This functionality is contained within the menu bar. In this example, refinement reduces the area by 29%, crossings by 11%, bends by 12%, and total edge length by 24%.

vas and the statistical information at the top of the interface shows how many crossings were removed. The user can also choose to perform the refinement modules which deal with just the edges by selecting the “Edges” button. For the quickest application of all the refinement modules, the user chooses “ALL”. The refinements are performed in the same order as their respective buttons appear on the interface. This ordering may easily be configured for each user: the individual techniques work in the same manner although the final outcome may be different.

We have designed and implemented all of our refinement techniques to take linear time with respect to the number of grid segments. Typically the number of grid segments is bounded by the number of nodes and edges and hence refinement takes $O(n + m)$ time, where n is the number of nodes and m is the number of edges.

3.2 Experimental Results

We have conducted a set of experiments with an implementation of our refinement procedure. The source drawings are 1400 layouts of the graphs used in the experimental study of [8], which we will refer to as the Rome Graphs, (available at `http://www.inf.uniroma3.it/people/gdb/wp12/ LOG.html`) which range in size from 10 to 65 nodes and are produced by the Bend-Stretch, Column, Giotto, and Pair algorithms as implemented at Brown University's Graph Drawing Server [4] (`http://loki.cs.brown.edu:8081/graphserver/`), and by GLT. Each drawing was given as input to our refinement implementation and data collected as to the improvements made in area, bends, edge crossings, and total edge length for each drawing. A table summarizing the average percent improvement for this set of aesthetics over drawings from the five layout algorithms is given in Figure 10.

In the table, the left column of percentages for each algorithm represents the average improvement over all drawings. This includes the drawings for which our techniques are unable to improve the drawing with respect to that aesthetic. The second column of percentages for each layout technique represents the average percent improvement for those drawings which our approach was able to strictly improve. The row marked δ, represents the average percent improvement made with respect to all four aesthetics for that particular algorithm. Δ represents the average percent improvement made for area, bends, crossings or edge length over all of the experimental source drawings. Note that refinement acts on the input drawing which is produced by a specific implementation of a chosen algorithm. As such, refinement improves aesthetic qualities caused by possible problems of both a chosen algorithm and the implementation of that algorithm.

Our implementation of refinement makes a 37% improvement on average in both area and total edge length. This huge improvement is due largely to the modules of our techniques which shorten long edges. As it is well known in the VLSI circuit design field, the area is usually dominated by the amount of

	Bend-Stretch		Column		Giotto		Pair		GLT		Δ	
	All	Better	All	Better	All	Better	All	Better	All	Better	All	Better
Area	29	32	53	53	7	15	61	61	33	37	37	40
Bends	18	21	45	45	2	12	42	43	10	25	23	29
Crossings	10	29	49	51	2	18	39	45	23	30	25	35
Edge Length	31	32	55	55	22	24	53	53	22	26	37	38
δ	22	29	51	51	8	17	49	51	22	30		

Figure 10: Summary of results from experimental study.

wiring. Likewise, the area of graph drawings is often dominated by the edge lengths. So our methods which shorten the total edge length also inherently decrease the area. Hence we see a proportional improvement in both area and total edge length. In addition the area is also reduced by the compaction phase. Although several orthogonal layout algorithms already use a compaction phase, our compaction has such a significant effect since other phases of refinement have simplified the drawing by reducing its geometric complexity. Therefore, pieces of the drawing have more freedom to move and can therefore be compacted more efficiently.

Our experiments also show about 24% improvement on average with respect to bends and crossings. This is due to the modules which particularly refine those elements.

Refinement significantly improves drawings created by each of these orthogonal algorithms. As mentioned earlier, every layout algorithm has a set of strengths and weaknesses. These strengths and weaknesses with respect to area, bends, crossings and total edge length are apparent in the numbers collected for each algorithm in our experiments. Giotto is the most evolved implementation of the Graph Drawing Server (GDS) and thus refinement has a lesser impact on these drawings. One of the main steps of Giotto finds the minimum number of bends of the embedded planarized graph [33]. Our refinement improved the number of bends by an average 2%. Improvement is possible since some of our refinement modules slightly modify the embedding. When our tool was able to improve the number of bends, the improvement was on the average 12%, with some improvements up to 33%. The planarization step of Giotto causes some nodes to be placed far from their neighbors, hence we see a more significant improvement of 22% with respect to total edge length.

Likewise we notice similar behavior with GLT. Their implementation allows each edge to have at most one bend. So the average improvement in the number of bends is 10% compared to the average 23%. Column and Pair drawings experience very significant improvements of each aesthetic. Especially notice the average 45% and 42% improvements in the number of bends and the average 55% and 53% improvements in total edge length. This is related not only to the nature of these algorithms, but also to the implementation of these algorithms in the GDS. Instead of dividing input graphs into biconnected components and performing a layout on each component, the GDS implementations of Column and Pair augment graphs to make them biconnected and then perform the layout on the augmented graph. The augmenting edges are removed during the final step and the resulting drawing shows only the input graph. This implementation decision has increased the geometric complexity of many Column and Pair drawings. Our refinement techniques provide a 10% to 31% average improvement of Bend-Stretch drawings for all aesthetics considered.

It is important to note the difference between the "All" percentages and the "Better" percentages. All five of these orthogonal algorithms produce good drawings: sometimes the number of crossings and bends is already very low, or even optimal in a few cases. Of course, the refinement tool cannot reduce the number of bends in a drawing which already has the lowest possible number of bends. Also, some layouts do not allow the embedding to be refined much.

Example drawings from Bend-Stretch, Column, Giotto, Pair, and GLT along with their refined drawings are included in Figures 11 - 19. Another such example for Giotto appears in Figure 1.

4 Conclusions and Future Work

In this paper we presented efficient postprocessing techniques to improve aesthetic qualities of any orthogonal graph drawing. Specifically, we focused on reducing the area, number of bends, number of crossings and total edge length. An experimental study conducted over a set of drawings from five well known algorithms produced very good results. An average 37% improvement was made in area, 23% in bends, 25% in crossings, and 37% in edge length.

Refinement is an evolving methodology. We plan to implement more modules of refinement as we develop further techniques to improve orthogonal graph drawings. Also, we plan to further enhance the graphical user interface of our refinement tool so that the refinement process can be even more tailored to an individual user's needs. More parametric options will be added. Further, we would like to capture the user's modal interactions with the nodes and edges so that we can further improve the quality of given drawings to some application-specific standard.

Acknowledgments

The authors would like to thank Uğur Doğrusöz and Therese Biedl for their technical advice. We are also grateful to Stina Bridgeman, Roberto Tamassia and the Graph Drawing group at Brown University for the Graph Drawing Server which is a great resource. We also wish to thank the referees for several useful suggestions.

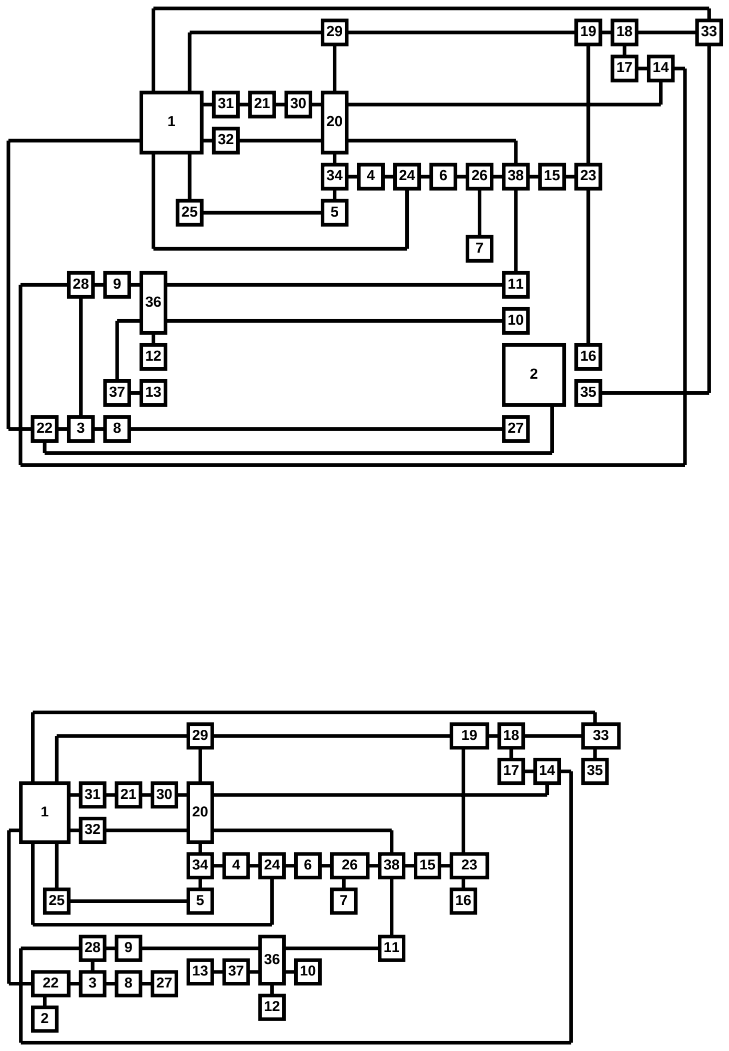


Figure 11: One of the Rome graphs as drawn with Bend-Stretch on the top. The same drawing after refinement on the bottom (same scale). There is a 34% improvement in area, 24% in the number of bends, 33% in the number of crossings, and 39% in the total edge length.

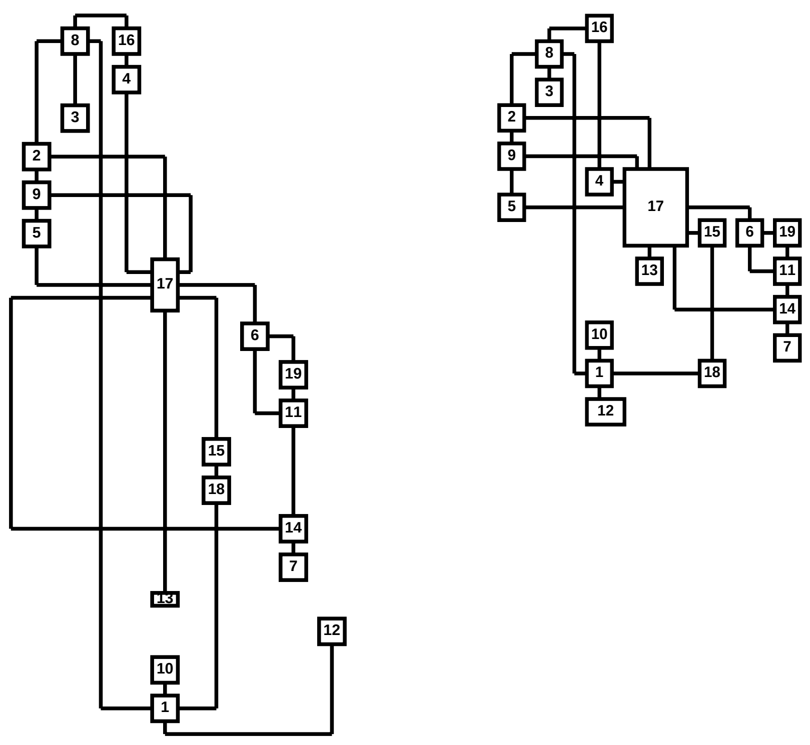
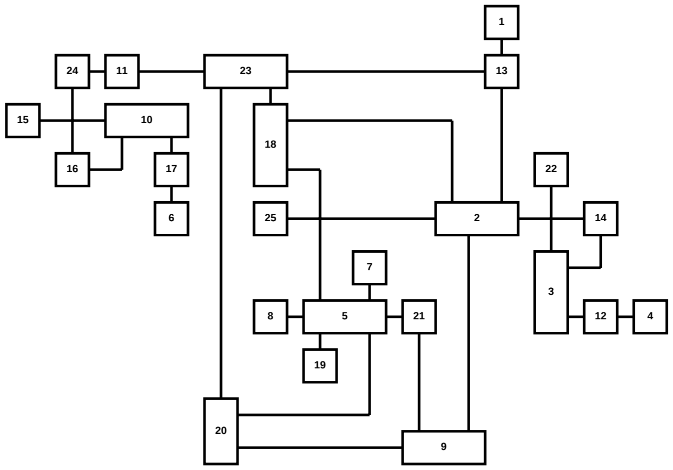


Figure 12: One of the Rome graphs as drawn with Column on the left. The same drawing after refinement on the right (same scale). There is a 47% improvement in area, 42% improvement in the number of bends, 40% in the number of crossings, and 56% in the total edge length.



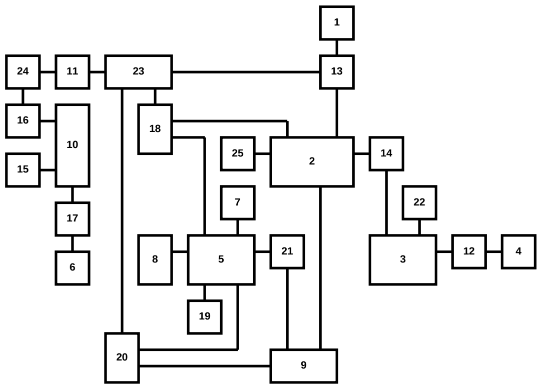


Figure 13: One of the Rome graphs as drawn with GLT on the top. The same drawing after refinement on the bottom (same scale). There is a 34% improvement in area, 40% improvement in the number of bends, 100% in the number of crossings, and 36% in the total edge length.

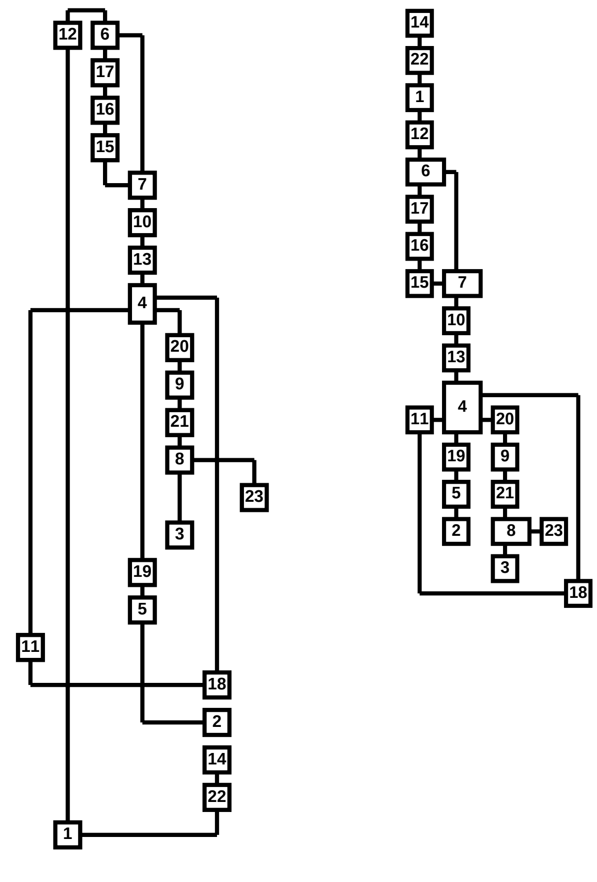


Figure 14: One of the Rome graphs as drawn with Column on the left. The same drawing after refinement on the right (same scale). There is a 46% improvement in area, 73% improvement in the number of bends, 100% in the number of crossings, and 68% in the total edge length.

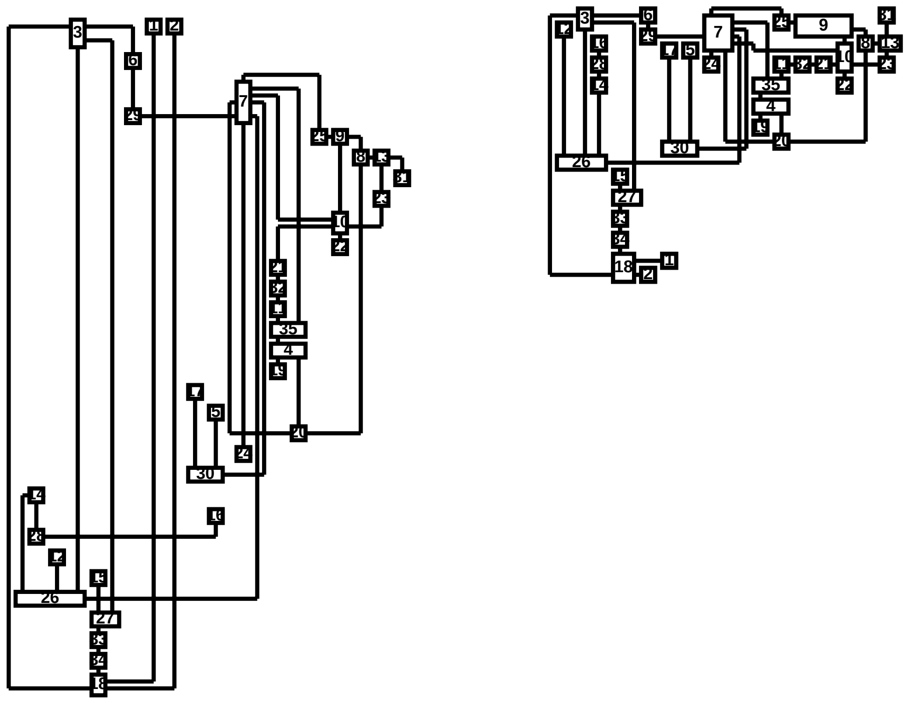
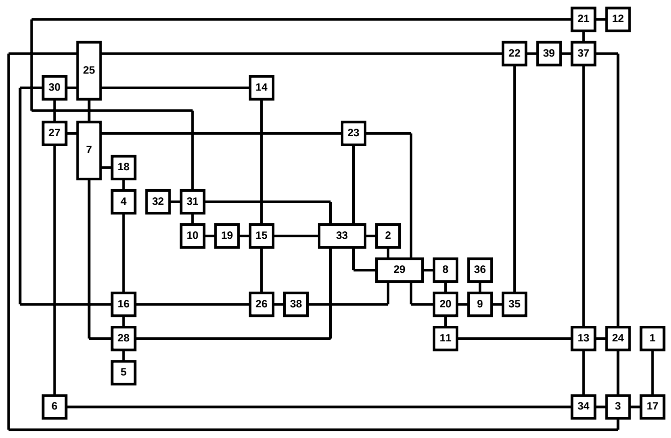


Figure 15: One of the Rome graphs as drawn with PAIR on the left. The same drawing after refinement on the right (same scale). There is a 65% improvement in area, 35% improvement in the number of bends, 56% in the number of crossings, and 66% in the total edge length.



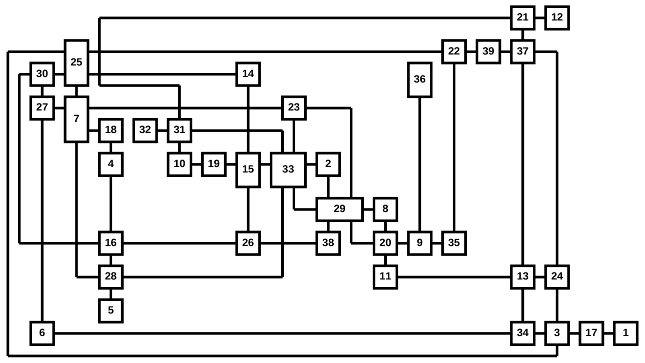


Figure 16: One of the Rome graphs as drawn with Giotto on the top. The same drawing after refinement on the bottom (same scale). There is a 19% improvement in area, 6% improvement in the number of bends, 20% in the number of crossings, and 17% in the total edge length.

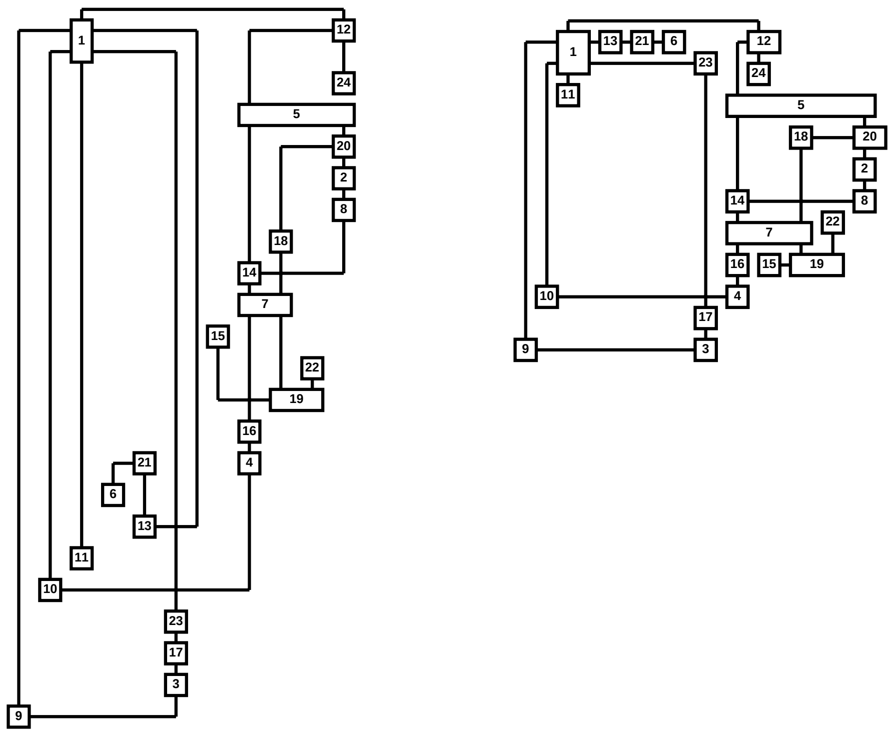


Figure 17: One of the Rome graphs as drawn with PAIR on the left. The same drawing after refinement on the right (same scale). There is a 49% improvement in area, 64% improvement in the number of bends, 50% in the number of crossings, and 60% in the total edge length.

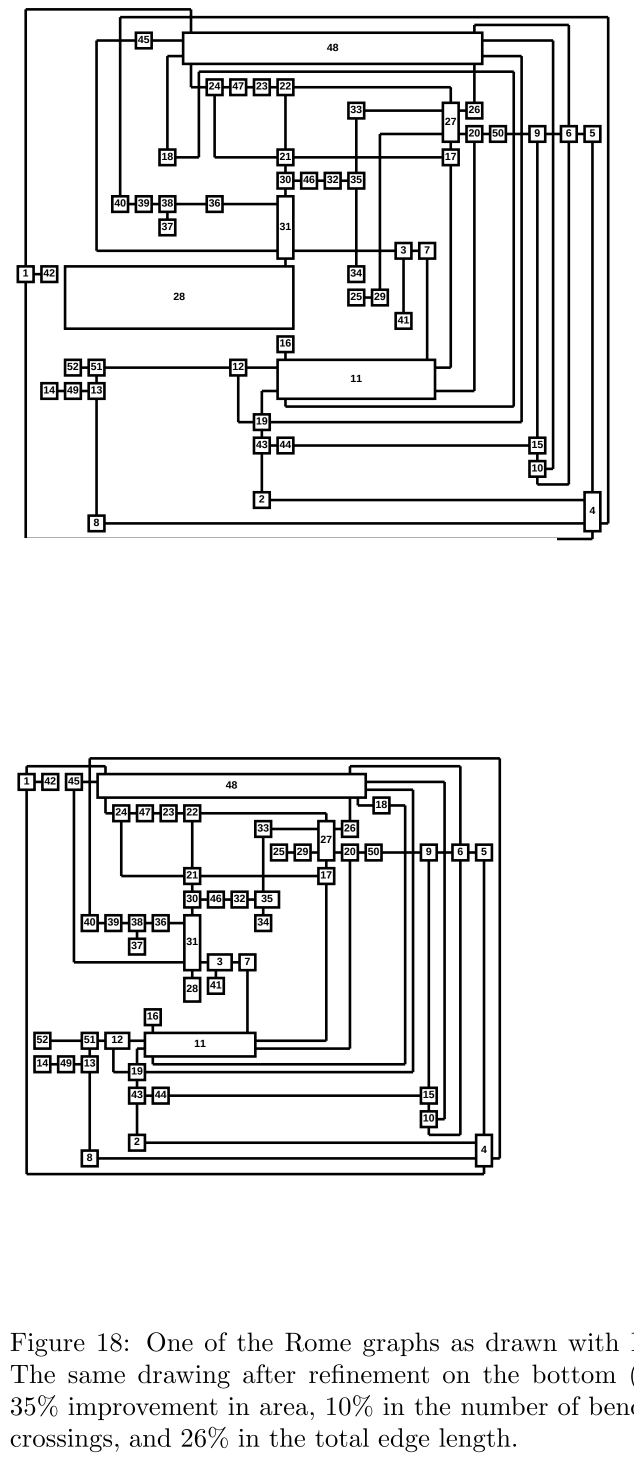


Figure 18: One of the Rome graphs as drawn with Bend-Stretch on the top. The same drawing after refinement on the bottom (same scale). There is a 35% improvement in area, 10% in the number of bends, 43% in the number of crossings, and 26% in the total edge length.

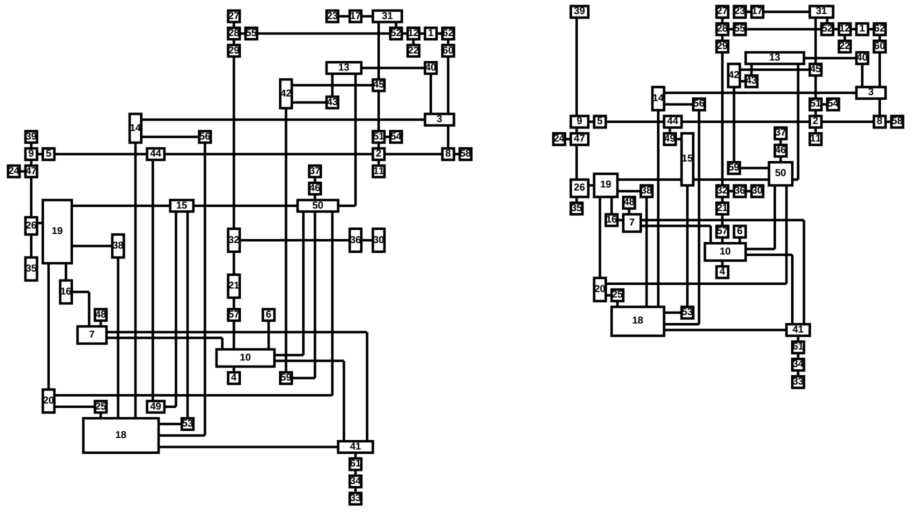


Figure 19: One of the Rome graphs as drawn with GLT on the left. The same drawing after refinement on the right (same scale). There is a 42% improvement in area, 30% improvement in the number of bends, 36% in the number of crossings, and 41% in the total edge length.

References

[1] T. Biedl and G. Kant, A Better Heuristic for Orthogonal Graph Drawings, *Computational Geometry: Theory and Applications*, 9(1998), 1998, pp. 159-180.

[2] T. C. Biedl, B. P. Madden and I. G. Tollis, The Three-Phase Method: A Unified Approach to Orthogonal Graph Drawing, *Proc. GD '97, LNCS 1353*, Springer-Verlag, 1997, pp. 391-402. Also available at `http://www.utdallas.edu/~tollis`.

[3] S. Bridgeman, J. Fanto, A. Garg, R. Tamassia and L. Vismara, Interactive Giotto: An Algorithm for Interactive Orthogonal Graph Drawing, *Proc. GD '97, LNCS 1353*, Springer-Verlag, 1997, pp. 303-308.

[4] S. Bridgeman, A. Garg and R. Tamassia, A Graph Drawing and Translation Service on the WWW, *Proc. GD '96, LNCS 1190*, Springer-Verlag, 1997, pp. 45-52.

[5] R. F. Cohen, G. Di Battista, R. Tamassia and I. G. Tollis, Dynamic Graph Drawings: Trees, Series-Parallel Digraphs, and Planar *ST*-Digraphs, *SIAM J. Computing*, 24(5), October 1995, pp. 970-1001.

[6] G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, Algorithms for Drawing Graphs: An Annotated Bibliography, *Computational Geometry: Theory and Applications*, 4(5), 1994, pp. 235-282. Also available at `http://www.utdallas.edu/~tollis`.

[7] G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, 1999.

[8] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari and F. Vargiu, An Experimental Comparison of Four Graph Drawing Algorithms, *Computational Geometry: Theory and Applications*, 1997, pp. 303-325. Also available at `http://www.cs.brown/~rt`.

[9] C. Ding and P. Mateti, A Framework for the Automated Drawing of Data Structure Diagrams, *IEEE Transactions on Software Engineering*, 16(5), 1990, pp. 543-557.

[10] J. Doenhardt and T. Lengauer, Algorithmic Aspects of One Dimensional Layout Compaction, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5), 1987, pp. 863-879.

[11] C. Esposito, Graph Graphics: Theory and Practice, *Computers and Mathematics with Applications*, 15(4), 1988, pp. 247-253.

[12] S. Even and G. Granot, Rectilinear Planar Drawings with Few Bends in Each Edge, Tech. Report 797, CS Dept., Technion, Israel Inst. of Tech., 1994.

[13] Jody Fanto, Postprocessing of GIOTTO drawings, Student project, Brown University, Summer 1997.

[14] U. Fößmeier, Interactive Orthogonal Graph Drawing: Algorithms and Bounds, *Proc. GD '97, LNCS 1353*, Springer-Verlag, 1997, pp. 111-123.

[15] U. Fößmeier, C. Heß and M. Kaufmann, On Improving Orthogonal Drawings: The 4M-Algorithm, *Proc. GD '98, LNCS 1547*, Springer-Verlag, 1998, pp. 125-137.

[16] U. Fößmeier and M. Kaufmann, Algorithms and Area Bounds for Nonplanar Orthogonal Drawings, *Proc. GD '97, LNCS 1353*, Springer-Verlag, 1997, pp. 134-145.

[17] M. Y. Hsueh, *Symbolic Layout and Compaction of Integrated Circuits*, Ph.D. Thesis, University of California at Berkeley, Berkeley, CA, 1979.

[18] G. Kant, Drawing Planar Graphs Using the Canonical Ordering, *Algorithmica*, 16(1), 1996, pp. 4-32.

[19] C. Kosak, J. Marks and S. Shieber, Automating the Layout of Network Diagrams with Specified Visual Organization, *IEEE Transactions on Systems, Man, Cybernetics*, 24(3), 1994, pp. 440-454

[20] Thomas Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley and Sons, 1990.

[21] K. Miriyala, S. W. Hornick and R. Tamassia, An Incremental Approach to Aesthetic Graph Layout, *Proc. Int. Workshop on Computer-Aided Software Engineering (Case '93)*, 1993, pp. 297-308.

[22] K. Misue, P. Eades, W. Lai and K. Sugiyama, Layout Adjustment and the Mental Map, *J. of Visual Languages and Computing*, June 1995, pp. 183-210.

[23] A. Papakostas, *Information Visualization: Orthogonal Drawings of Graphs*, Ph.D. Thesis, University of Texas at Dallas, 1996.

[24] A. Papakostas, J. M. Six and I. G. Tollis, Experimental and Theoretical Results in Interactive Orthogonal Graph Drawing, *Proc. GD '96, LNCS 1190*, Springer-Verlag, 1997, pp. 371-386. Also available at `http://www.utdallas.edu/∼tollis`.

[25] A. Papakostas and I. G. Tollis, Algorithms for Area-Efficient Orthogonal Drawings, *Computational Geometry: Theory and Applications*, 9(1998) 1998, pp. 83-110. Also available at `http://www.utdallas.edu/~tollis`.

[26] A. Papakostas and I. G. Tollis, Issues in Interactive Orthogonal Graph Drawing, *Proc. GD '95, LNCS 1027*, Springer-Verlag, 1995, pp. 419-430. Also available at `http://www.utdallas.edu/~tollis`.

[27] H. Purchase, Which Aesthetic has the Greatest Effect on Human Understanding, *Proc. of GD '97, LNCS 1353*, Springer-Verlag, 1997, pp. 248-261.

[28] M. Schäffter, Drawing Graphs on Rectangular Grids, *Discr. Appl. Math.*, 63(1995), pp. 75-89.

[29] J. M. Six, K. Kakoulis and I.G. Tollis, Refinement of Orthogonal Graph Drawings, *Proc. GD '98, LNCS 1547*, Springer-Verlag, 1998, pp. 302-315. Also available at `http://www.utdallas.edu/~tollis`.

[30] J. Storer, On Minimal Vertex-Cost Planar Embeddings, *Networks* 14(1984), pp. 181-212.

[31] R. Tamassia, On Embedding a Graph in the Grid with the Minimum Number of Bends, *SIAM J. Comput.*, 16(1987), pp. 421-444.

[32] R. Tamassia, G. Di Battista and C. Batini, Automatic Graph Drawing and Readability of Diagrams, *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1), 1988, pp. 61-79.

[33] R. Tamassia and I. G. Tollis, Planar Grid Embeddings in Linear Time, *IEEE Trans. on Circuits and Systems CAS-36*, 1989, pp. 1230-1234.

[34] I. G. Tollis, Graph Drawing and Information Visualization, *ACM Computing Surveys*, 28A(4), December 1996. Also available at `http://www.utdallas.edu/~tollis/`.