Experimental and Theoretical Results in Interactive Orthogonal Graph Drawing[*]

Achilleas Papakostas, Janet M. Six and Ioannis G. Tollis

Department of Computer Science
The University of Texas at Dallas
Richardson, TX 75083-0688
email: {papakost,janet,tollis}@utdallas.edu

Abstract. *Interactive Graph Drawing* allows the user to dynamically interact with a drawing as the design progresses while preserving the user's mental map. This paper presents a theoretical analysis of Relative-Coordinates and an extensive experimental study comparing the performance of two interactive orthogonal graph drawing scenaria: *No-Change*, and *Relative-Coordinates.* Our theoretical analysis found that the Relative-Coordinates scenario builds a drawing that has no more than $3n - 1$ bends, while the area of the drawing is never larger than $2.25n^2$. Also, no edge has more than 3 bends at any time during the drawing process. To conduct the experiments, we used a large set of test data consisting of 11,491 graphs (ranging from 6 to 100 nodes) and compared the behavior of the above two scenaria with respect to various aesthetic properties (e.g., area, bends, crossings, edge length, etc) of the corresponding drawings. The Relative-Coordinates scenario was a winner over No-Change under any aesthetic measure considered in our experiments. Moreover, the practical behavior of the two scenaria was considerably better than the established theoretical bounds, in most cases.

1 Introduction and Preliminaries

Graph drawing addresses the problem of automatically generating geometric representations of abstract graphs or networks. For a survey of graph drawing algorithms and other related results see the annotated bibliography of Di Battista, Eades, Tamassia and Tollis [4]. An *orthogonal* drawing is a drawing in which vertices are represented by points on integer coordinates and edges are represented by polygonal chains consisting of horizontal and vertical line segments. Various algorithms have been introduced to produce orthogonal drawings of planar [2, 6, 10, 20, 22] or general [1, 2, 16, 19] graphs of maximum degree four, and maximum degree three [10, 16, 17]. All these algorithms run in linear time, except for the algorithm in [20]. For drawings of general graphs, the required area can be as little as $0.76n^2$ [16], the total number of bends is no more than $2n + 2$ [2, 16], and at most two bends can be on the same edge [2, 16].

[*] Research supported in part by NIST, Advanced Technology Program grant number 70NANB5H1162.

Upper and lower area bounds have been proved for orthogonal drawings of general graphs. Leighton [11] presented an infinite family of planar graphs which require area $\Omega(n \log n)$. Independently, Leiserson [12] and Valiant [25] showed that every planar graph of degree three or four has an orthogonal drawing with area $O(n \log^2 n)$. Valiant [25] showed that the orthogonal drawing of a general (non-planar) graph of degree three or four requires area no more than $9n^2$, and described families of graphs that require area $\Omega(n^2)$.

Many graph drawing algorithms have been implemented, and a number of results on their practical behavior have appeared in literature. Himsolt [8] presented his experimental findings when he compared the performance of twelve graph drawing algorithms (most of which were specialized for trees, graph grammars, Petri nets and planar graphs) in the GraphEd environment [9]. An extensive experimental work appeared in [5] where four general-purpose orthogonal graph drawing algorithms [2, 16, 21, 22] were implemented and compared with respect to their performance on various aesthetic properties including crossings, bends, area, running time etc. Their results have a great statistical value mainly because they are based on a very large data set including 11,582 graphs.The sizes of these graphs range from 10 to 100 vertices and are taken from "real-life" software engineering and database applications.

In most graph drawing algorithms a graph is given as an input and a drawing of this graph is produced. If an insertion (or deletion) is performed on the graph, then we have a "new" graph. Running the drawing algorithm again will result in a new drawing, which might be vastly different from the previous one. Obviously this is a waste of human resources to continually re-analyze the entire drawing and also of computational resources to re-compute the entire layout after each modification. Therefore it is important to efficiently produce a series of drawings which evolve with the structure while preserving the user's *mental map* [15]. The first systematic approach to dynamic graph drawing appeared in [3]: the target was to perform queries and updates on an implicit representation of the drawing, while maintaining its planarity. The insertion of a single edge however, could cause a planar graph to drastically change embedding, or even become non planar. An incremental approach to orthogonal graph drawing was presented in [14], where the focus was on routing edges efficiently without disturbing existing vertices or edges.

In [18] we discussed various features of interactive graph drawing systems, we introduced four scenaria for interactive graph drawing and presented a theoretical analysis of the performance of the No-Change scenario. All four scenaria were based on the assumption that the underlying drawing was orthogonal and the maximum degree of any vertex was four at the end of an update operation. The basic property of the No-Change scenario is that an update operation (i.e., a vertex insertion) does not alter the coordinates of any vertex or bend within the current drawing, since any vertex insertion or edge routing takes place around it. The analysis of the No-Change scenario of [18] is based on the assumption that the drawn graph is connected at all times.

Theorem 1. *[18] The "No-Change scenario"produces drawings with the following properties:*

1. *every insertion operation takes constant time,*
2. *every edge has at most three bends,*
3. *the total number of bends at any time t is at most $2.66n(t) + 2$, where $n(t)$ is the number of vertices of the drawing at time t,*
4. *the area of the drawing at any time t is no more than $(n(t) + n_4(t))^2 \leq 1.77n(t)^2$, where $n_4(t)$ is the number of vertices of local degree four which have been inserted up to time t, and*
5. *the upper bounds for both area and bends are tight.*

In this paper, we first analyze the performance of the Relative-Coordinates scenario by using linear programming to prove upper bounds on the area and the number of bends. More specifically, we show that an interactive graph drawing system under the Relative-Coordinates scenario builds a drawing that has no more than $3n(t) - 1$ bends, while requiring at most $2.25n(t)^2$ area. Moreover, no edge has more than 3 bends at any time during the drawing process. Then we compare the performance of the No-Change and Relative-Coordinates scenaria on a set of 11,491 "real-life" maximum degree four graphs, which were taken from the database of graphs used in [5]. Our experiments compare the quality of the drawings produced by the two scenaria, based on the following aesthetic measures: area, number of bends, number of crossings, aspect ratio, average edge length and maximum edge length. We also include results on the average number of new rows, columns and bends that are introduced in any drawing for different types of vertex insertions. Our experiments revealed:

- The practical behavior of the two scenaria is much better than their established theoretical bounds, in most cases.
- The Relative-Coordinates scenario exhibits better performance than No-Change under any aesthetic measure considered.

2 Analysis of the Relative-Coordinates Scenario

In this scenario, every time a new vertex is about to be inserted into the current drawing, the system makes a decision about the coordinates of the vertex and the routing of its incident edges. New rows and columns may be inserted anywhere in the current drawing in order for this routing to be feasible. The coordinates of the new vertex (say v) as well as the locations of the new rows and/or columns will depend on the following:

- v's degree (at the time of insertion).
- How many of v's adjacent vertices allow the insertion of a new incident edge towards the same direction (i.e., up, down, right, or left of the vertex).
- How many of v's adjacent vertices allow a new incident edge towards opposite directions.
- Whether or not the required routing of edges can be done utilizing segments of existing rows or columns that are free (not covered by an edge).
- Our optimization criteria.

When we use the Relative-Coordinates scenario in an interactive system, we can start from an existing drawing of a graph or from scratch, that is from an empty graph. In either case, we assume that the insertion of any vertex/edge under this scenario will not increase the number of connected components of the current graph. The only exception to this is when a single vertex is inserted into a currently empty graph. Any other vertex inserted during an update step will be connected to at least one other vertex of the current drawing. Let us assume that v is the next vertex to be inserted in the current graph during an update step. The number of vertices in the current graph that v is connected to, is called the *local degree* of v, and is denoted by $local\_degree(v)$.

From the discussion above it follows that we only consider the cases where an inserted vertex has local degree one, two, three or four, except for the first vertex inserted in an empty graph. If the user wishes to insert a new vertex that has local degree zero, then this vertex is placed in a temporary location and it will be inserted automatically in the future, when some newer vertices increase its (local) degree. Assume that vertex v is about to be inserted into the current graph. For each one of the vertices of the current drawing that is adjacent to v, the system checks the possible directions around these vertices that new edges may be inserted or routed. The target is to minimize the number of new rows or columns that have to open up in the current drawing, as well as the number of bends that appear along the routed edges.

There are many different cases because there are many possible combinations. In the example shown in Fig. 1a vertices u_1 and u_2 have a free edge (i.e., grid edge not covered by a graph edge) up and to the right respectively. In this case no new rows/columns are needed for the insertion of vertex v and no new bends are introduced. On the other hand however, in the example shown in Fig. 1b all four vertices u_1, u_2, u_3 and u_4 have pairwise opposite direction free edges. The insertion of new vertex v requires the insertion of three new rows and three new columns in the current drawing. Additionally, eight bends are introduced. Vertices u_1, u_2, u_3 and u_4 have general positions in Fig. 1b, and we can see that edge (v, u_4) has four bends. We can avoid the 4-bend edge, if we insert vertex v in the way shown in Fig. 1c. The total number of new rows, columns and bends is still the same, but the maximum number of bends per edge is now three. For a more even distribution of the bends of the edges adjacent to vertex v, we may choose to insert it in the way shown in Fig. 1d, where every edge has exactly two bends (three new rows and three new columns are still required). Notice, though, that the approach described in Fig. 1d for inserting vertex v, is not always possible (e.g., we cannot have this kind of insertion if vertices u_1, u_2, u_3 and u_4 are in the same row or column). At this point it is important to note that any one of the solutions presented in Fig. 1 (b, c and d) are acceptable: all add the same number of rows, columns and *total* number of bends. The characteristic of placing a maximum of three bends per edge is attainable, but ultimately is up to the implementor. The *total* number of bends added per insertion will always remain the same under this scenario.

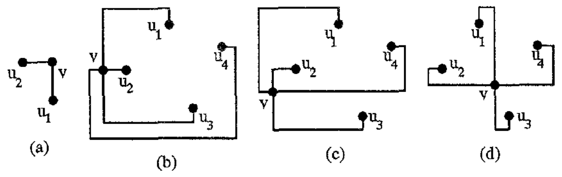

Fig. 1. Insertion of v: (a) no new row or column is required, (b),(c) and (d) three new rows and three new columns are required, with a maximum of 4 bends per edge in (b), 3 bends per edge in (c), and 2 bends per edge in (d).

Let v be the next vertex to be inserted. There are many cases, if one is interested in an exhaustive analysis. However, it is relatively easy to come up with all the cases for each insertion. Here, we distinguish the following main cases for vertex v:

1. v has local degree one. If u is the vertex of the current drawing that is adjacent to v, we draw an edge between u and v. Edge (u, v) uses a direction (up, right, bottom, or left) that is not taken by some other edge incident to u. This is depicted in Fig. 2a, and this insertion requires at most either a new row or a new column. No new bend is inserted.
2. v has local degree two. In the best case, the insertion requires no new rows, columns or bends as shown in Fig. 1a. In the worst case, though, two new rows and one new column, or one new row and two new columns (see Fig. 2b), and three new bends might be inserted.
3. v has local degree three. In the worst case, the insertion requires a total of four new rows and columns, and five new bends. In Fig. 2c we show an example of such an insertion that requires one new row, three new columns and five new bends.
4. v has local degree four. The worst case requires a total of six new rows and columns, and eight new bends. We have already discussed an example, which is depicted in Fig. 1c. In Fig. 2d we show another case, where two new rows, four new columns and eight new bends are introduced. Note that no more than four new rows or columns may be introduced when v has local degree four.

As discussed in the previous section, single edge insertions can be handled using techniques from global routing [13] or the technique of [14]. The easiest way to handle deletions is to delete vertices/edges from the data structures without changing the coordinates of the rest of the drawing. Occasionally, or on demand, the system can perform a linear-time compaction similar to the one described in [22], and refresh the screen.

In the rest of this section we assume that, when we use the interactive graph drawing scheme under the Relative-Coordinates scenario, we start from scratch. According to the discussion in the beginning of this section, the Relative-

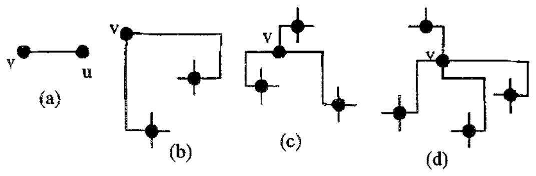

Fig. 2. Inserting v when its local degree is (a) one, (b) two, (c) three, and (d) four.

Coordinates scenario guarantees that the graph that is being built is always connected after any vertex insertion. Let $n_1(t)$, $n_2(t)$, $n_3(t)$ and $n_4(t)$ denote the number of vertices of local degree one, two, three and four, respectively, that have been inserted up to time t.

Theorem 2. *An interactive graph drawing system under the "Relative Coordinates scenario" produces drawings with the following properties:*

1. *after each vertex insertion, the coordinates of any vertex or bend of the current drawing may shift by a total amount of at most 6 units along the x and y axes,*
2. *there are at most 3 bends along any edge of the drawing,*
3. *the total number of bends is at most $3n(t) - 1$, and*
4. *the area of the drawing is at most $2.25n(t)^2$,*

where $n(t)$ is the number of vertices that have been inserted up to time t.

Sketch of Proof. The first property follows from the definition of the Relative-Coordinates scenario and from the fact that at most 6 new rows and new columns might open anywhere in the current drawing (see Figs. 1b, 1c, 1d, 2d) as a result of a vertex insertion. Figures 1 and 2 cover the worst cases in terms of rows, columns and bends required for a single vertex insertion, and for all possible local degrees of the inserted vertex. From these figures we observe the following: First there can be at most 3 bends along any edge of the drawing (see Fig. 1c). Second, the bends along an edge are introduced at the time of insertion of the vertex that is incident to that edge.

From Figs. 1 and 2 and from the discussion above, it follows that at most 3 new bends are introduced when a vertex of local degree 2 is inserted, at most 5 new bends when a vertex of local degree 3 is inserted, and at most 8 new bends when a vertex of local degree 4 is inserted. No new bend is introduced when a vertex of local degree 1 is inserted. In other words, if $B(t)$ is the total number of bends at time t, it holds that:

$$B(t) \leq 3n_2(t) + 5n_3(t) + 8n_4(t)$$

We want to compute the maximum value that $B(t)$ can take, in order to establish an upper bound on the number of bends of the drawing at time t. This is equivalent to solving the following linear program:

$$\text{maximize} \ : \ 3n_2 + 5n_3 + 8n_4$$

under the following constraints:

$$\begin{gathered} n_1 \geq 1 \\ n_1 + n_2 + n_3 + n_4 = n - 1 \\ n_1 + 2n_2 + 3n_3 + 4n_4 \leq 2n \end{gathered}$$

The first constraint is an inequality on the number of local degree one insertions, the second is an equation on the number of vertices, and the third is an inequality on the number of edges of the graph, after n vertices have been inserted. Recall that the first vertex to be inserted has local degree 0, since it does not have any edges yet.

Solving this linear program with all three constraints leads to a non-integral solution. If we ignore the first constraint, the new linear program has an integral solution and the objective function is maximized (to $3n + 2$) when $n_1 = n_3 = 0$, $n_2 = n - 2$, and $n_4 = 1$. This solution implies that maximizing the number of bends depends solely on the number of vertices of local degree two and four. If we take into account the fact that the first two vertices inserted in an empty graph have local degrees 0 and 1 respectively, what we really have is that $n_2 = n - 3$ and $n_4 = 1$. This is the same solution as the one obtained from the first linear program after relaxing the solution into an integral one. We can also see that any other combination of values for n_2 and n_4 when $n_2 + n_4 = n - 2$ gives more than $2n - 1$ edges (recall that one edge is introduced by the second vertex, which has local degree 1). From the above analysis, it follows that the upper bound on the number of bends is $3n - 1$.

Regarding the area of the drawing at time t, we can infer from Figs. 1 and 2 that:

- when a vertex with local degree one is inserted, either a new row or a new column is required,
- when a vertex with local degree two is inserted, either two new rows and one new column are required, or one new row and two new columns are required,
- when a vertex with local degree three is inserted, we need a total of at most four new rows and new columns, and
- when a vertex with local degree four is inserted, we need a total of at most six new rows and new columns.

Let $h(t)$ and $w(t)$ denote the height and the width, respectively, of the drawing at time t. Then it holds that:

$$h(t) + w(t) \leq n_1(t) + 3n_2(t) + 4n_3(t) + 6n_4(t) \leq 2n(t) + n_2(t) + n_3(t) + 2n_4(t)$$

since $n_1(t) + 2n_2(t) + 3n_3(t) + 4n_4(t) \leq 2n(t)$. We want to maximize $h(t) + w(t)$. If we just multiply both sides of the last inequality (i.e., the one on the edges of the graph) by $\frac{3}{2}$, we obtain $h(t) + w(t) \leq 3n(t)$. However, this solution does not give us the values of the variables (i.e., $n_1(t)$, $n_2(t)$, etc), for which this upper bound is achieved. For this reason, we formulate this problem as a linear program, where the expression to be maximized is: $2n + n_2 + n_3 + 2n_4$, and the constraints are exactly the same as the ones in the above linear program.

Solving this new linear program, we have that $h(t) + w(t)$ is maximized when $n_1 = n_3 = 0$, $n_2 = n - 2$, and $n_4 = 1$, exactly as in the linear program we studied above, for the number of bends. According to the analysis we did for that linear program, these results really mean that $n_1 = 1$ (the second vertex to be inserted), $n_2 = n - 3$, and $n_4 = 1$. The maximum value of expression $2n + n_2 + n_3 + 2n_4$ that we wanted to maximize is now $3n$. This means that $h(t) + w(t) \leq 3n(t)$. It also holds that $h(t) \times w(t)$ is maximized when $h(t) = w(t) = \frac{h(t)+w(t)}{2} \leq 1.5n(t)$. In this case, the area of the drawing can be at most $2.25n(t)^2$. □

Let us have a look at the expression giving the number of bends that we maximized in the linear program of the proof of Theorem 2. One might be tempted to believe that this expression is maximized when $n_4(t)$ is maximized (and this happens when $n_4(t) = \frac{n(t)}{3}$, if the graph is always connected). The result of the linear program was quite revealing. We discovered that this expression is maximized only under the following insertion sequence: insert the first two vertices with local degrees 0 and 1 respectively, followed by $n - 3$ vertices of local degree two, and conclude with the insertion of exactly one vertex of local degree four. In order to refresh the drawing after each update, the coordinates of every vertex/bend affected must be recalculated. Hence, it would take linear time.

3 Experimental Comparison of the Two Interactive Scenaria

3.1 Implementation and Experiments

Both the No-Change and Relative-Coordinates algorithms have been implemented in C++ (GNU C++ version 2.6.0) on a SPARC 5 running SUN OS Release 4.1.3. Our implementations are running on top of Tom Sawyer Software's Graph Layout Toolkit version 2.2 [23, 24]. We converted 11,491 of the graphs used in the experimental analysis of [5] for our set of experiments. This database of graphs is available by anonymous ftp from `infokit.dis.uniroma1.it:public`. A small number of edges and vertices were discarded in order to make all vertices maximum degree four. For each graph, *arbitrarily selected* vertices and their incident edges were inserted one at a time into an initially empty structure. The graph was connected at all times. The following standard measures of quality were determined for each drawing:

- Area: The area of the smallest box which can bound the drawing.
- Bends: The total number of bends.
- Crossings: The total number of crossings.
- Aspect Ratio: The width divided by the height of the drawing.
- Average Edge Length: The sum of all edge lengths divided by the number of edges in the drawing.
- Maximum Edge Length: The length of the single longest edge in the drawing.

In addition to these standard measures, we also count the number of rows, columns and bends added with each type of insertion (local degree one, two, three or four). These factors show behavior specific to interactive orthogonal drawing algorithms. All of these measures are plotted against the final number of vertices in the graph.

3.2 Performance Analysis Under Various Quality Measures

Our experimental results show that both No-Change and Relative-Coordinates scenaria behave better than their theoretical upper bounds. The most important aspect of our experimental work is the observation that the performance of the Relative-Coordinates approach for graph drawing is considerably better than that of No-Change with respect to the aesthetic measures we considered. Although both algorithms respected their theoretical bounds, Relative-Coordinates' average case behavior was consistently better than that of No-Change. In Fig. 3 we show the drawings of one graph from our experimental set with the same random order of vertex insertion, when drawn under the two different scenaria. As is evidenced in the pair of drawings, each scenario has its own distinctive style. The No-Change scenario placed the first two vertices at the top left corner of the drawing and grew in a south-easterly direction. Relative-Coordinates maintained the general shape of the drawing, but inserted a small number of rows and columns in order to facilitate the placement of the new vertex and the routing of its incident edges. Some of our experimental findings are summarized in Fig. 4. More specifically, we have:
Area: The area of graphs laid out by Relative-Coordinates is consistently smaller than that of No-Change. The theoretical upper bound of the No-Change scenario is $1.77n^2$ while the behavior is closer to $\frac{n^2}{2}$. Likewise, the theoretical upper bound of the Relative-Coordinates scenario is $2.25n^2$ and the experimental behavior is closer to $\frac{n^2}{4}$.
Bends: Relative-Coordinates produces drawings with fewer bends than No-Change. This happens because, under Relative-Coordinates, newly inserted vertices are expected to be placed closer to their adjacent vertices. In addition, the first invariant of No-Change forces us to place bends in a significant percentage of all degree one insertions. In other words, if a low or no bend edge exists, Relative-Coordinates will use it, but No-Change may not in order to comply with its own invariants. The theoretical upper bound for No-Change is $2.66n+2$ while it behaves more like $\frac{n}{2}$. Likewise the theoretical upper bound for Relative-Coordinates is $3n-1$ and it behaves closer to $\frac{n}{4}$.
Crossings: Again Relative-Coordinates performs significantly better than No-Change and this behavior is expected. Since the No-Change scenario allows no coordinate of any vertex or bend to change, the scenario must comply with its own invariants, so new edges are often forced to cross many old edges to reach their incident vertices. Contrariwise, Relative-Coordinates allows some change (in the form of row and column additions anywhere within the drawing) and places vertices closer to their adjacent vertices.
Aspect Ratio: As can be seen in the plot, No-Change and Relative-Coordinates

behave in a very similar fashion. Both algorithms produce rather squarish drawings. It is important to note that certain modifications within the implementation will allow different behaviors: either algorithm could produce more rectangular drawings to comply with some requirement.
Average Edge Length: The Average Edge Length and Maximum Edge Length plots show a very important difference between the two scenaria. By the very nature of No-Change, newer vertices are forced to be far from their adjacent vertices if they were inserted at a much earlier point in the lifetime of the drawing. This factor, of course, causes the average edge length to be high. Relative-Coordinates adds a reasonable number of rows and columns as necessary to allow "good" placement of new vertices close to their adjacent vertices and edges with few bends.
Maximum Edge Length: The No-Change Algorithm produces long edges when a new vertex is connected to another vertex which was placed at, or near, the beginning of the lifetime of the graph. This is also an expected result.

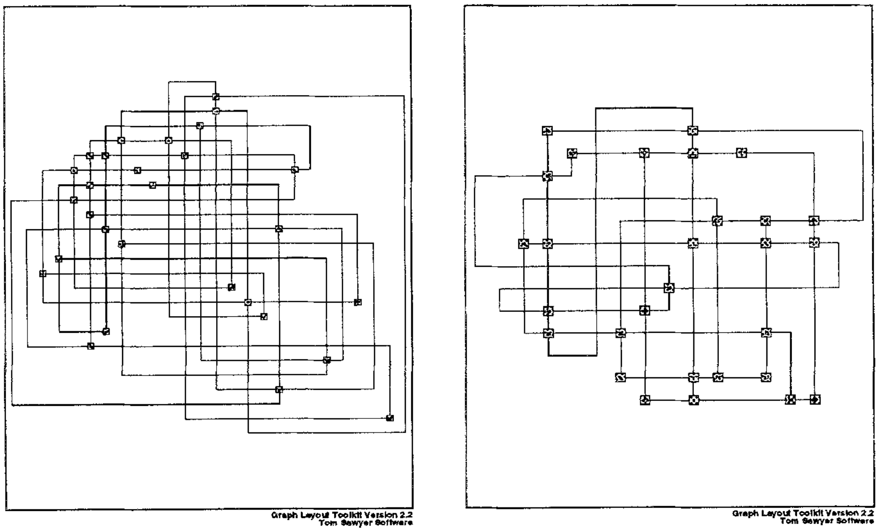
Graph Layout Toolkit Version 2.2
Tom Sawyer Software
Graph Layout Toolkit Version 2.2
Tom Sawyer Software

Fig. 3. Drawings of the same 29-vertex graph: No-Change on the left, Relative-Coordinates on the right.

At this point it is important to notice that Relative-Coordinates performs well even as a non-interactive algorithm. Although it is unfair to directly compare our results with those in [5] because we limit the test graphs to degree four, it is interesting that the average area of a Relative-Coordinates drawing is only slightly larger than the Giotto drawings. The same phenomenon is observed with crossings and average edge length. For these experiments we used an arbitrary

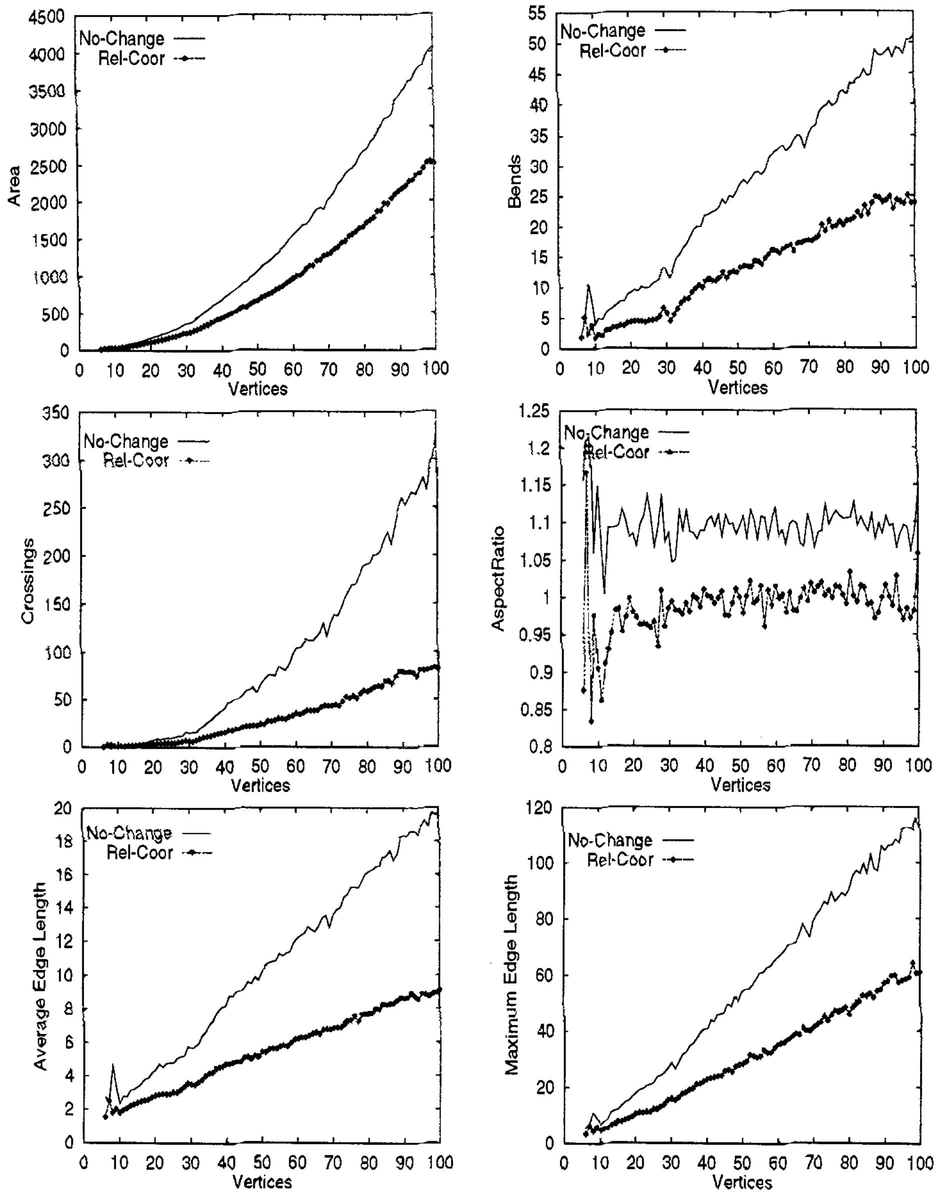

Fig. 4. Graphs of our experimental findings.

insertion sequence. If we find a "good" insertion sequence then we should obtain even better results. An additional refinement phase should produce data curves (representing area, bends, crossings, average and maximum edge lengths) that are very similar to those of Giotto.

3.3 Performance After a Single Update Step

Our second set of experimental data is pertinent to the interactive nature of the two scenaria. Our data show that Relative-Coordinates consistently outperforms No-Change in any measure considered. The plots of our results are shown in Fig. 5 and Fig. 6.

Rows and Columns Added per Degree One Insertion: No-Change adds at most one new row and one new column per degree one insertion while Relative-Coordinates adds either one new row or one new column. Therefore we expected the average number of rows and columns for degree one insertions to be slightly higher for No-Change. The collected data reflects this characteristic. The average number of rows and columns added per degree one insertion is $0.551 + 0.605 = 1.156$ for No-Change and $0.515 + 0.485 = 1.000$ for Relative-Coordinates.

Bends Added per Degree One Insertion: No-Change adds a bend to the edge if it is placed to the North or West of the old vertex while Relative-Coordinates never adds a bend during a degree one insertion. The average number of bends for No-Change is 0.118 and 0 for Relative-Coordinates.

Rows and Columns Added per Degree Two Insertion: No-Change adds at most one new row and one new column per degree two insertion. On average, No-Change adds $0.926 + 0.953 = 1.879$ rows and columns per degree two insertion. Notice that the sum is very close to the worst case. Relative-Coordinates adds a total of three new rows and columns in the worst case and the average is a much better $0.377 + 0.435 = 0.812$.

Bends Added per Degree Two Insertion: In the worst case, No-Change adds four bends while Relative-Coordinates adds three. The average number of bends inserted is 2.050 for No-Change and 0.812 for Relative-Coordinates. This number is so low because Relative-Coordinates makes good use of any free rows, columns and open degrees of freedom.

Rows and Columns Added per Degree Three Insertion: Theoretically at most two new rows and two new columns are added for a No-Change degree three insertion, and at most a total of four new rows and columns are needed for Relative-Coordinates. In our experiments we found that an average of $1.102 + 1.135 = 2.237$ rows and columns are added in the No-Change implementation and only $0.707 + 0.740 = 1.447$ rows and columns in Relative-Coordinates.

Bends Added per Degree Three Insertion: In the worst case, No-Change adds five bends and the average behavior is 3.205. Relative-Coordinates takes better advantage of available rows and columns and has an average of 2.447.

Rows and Columns Added per Degree Four Insertion: Degree four insertions are handled very similarly by both No-Change and Relative-Coordinates.

At most, a total of six new rows and columns are added by each algorithm. According to the data an average of $1.660+1.730 = 3.390$ rows and columns is added by the No-Change algorithm. Relative-Coordinates adds $1.343 + 1.373 = 2.716$.
Bends Added per Degree Four Insertion: Both algorithms add eight bends in the worst case, and No-Change has been experimentally found to produce an average of 5.022 bends per degree four insertion while Relative-Coordinates introduces an average of 3.987 bends.
Remember in Sect. 2 we proved that the area and number of bends within a Relative-Coordinates drawing is contingent on the number of degree two insertions. Therefore it is quite interesting and important to note that the experimental behavior of degree two insertions is so much better than the worst case. This explains why the behavior (with respect to area and bends) of Relative-Coordinates is so much better.

4 Conclusions and Open Problems

The Relative-Coordinates scenario maintains the general shape of the current drawing after an update (vertex/edge insertion/deletion) takes place, and does not affect the number of bends of the current drawing even if the update operation is a vertex insertion. We used linear programming in order to establish an upper bound for the performance of this scenario. A comparison of the practical behavior of the No-Change and Relative-Coordinates scenaria was presented. The two scenaria were tried on a very large set of "real-life" graphs, and results were reported with respect to their performance under various aesthetic measures. The Relative-Coordinates scenario was consistently better, while both scenaria respected their theoretical bounds.

It is an interesting open problem to develop a theory that enables the efficient insertion, movement or deletion of more than one vertex simultaneously (that is a block of vertices) in the current drawing. Also, techniques for interactive graph drawing in other standards (straight line, polyline, etc.) are needed, and should be explored.

References

1. Therese Biedl, Embedding Nonplanar Graphs in the Rectangular Grid, *Rutcor Research Report 27-93*, 1993.
2. T. Biedl and G. Kant, A Better Heuristic for Orthogonal Graph Drawings, *Proc. 2nd Ann. European Symposium on Algorithms (ESA '94), Lecture Notes in Computer Science, vol. 855*, pp. 24-35, Springer-Verlag, 1994.
3. R. Cohen, G. DiBattista, R. Tamassia, and I. G. Tollis, Dynamic Graph Drawings:Trees, Series-Parallel Digraphs, and Planar st-Digraphs, *SIAM Journal on Computing*, vol. 24, no. 5, pp. 970-1001, 1995.
4. G. DiBattista, P. Eades, R. Tamassia and I. Tollis, Algorithms for Drawing Graphs: An Annotated Bibliography, *Computational Geometry: Theory*

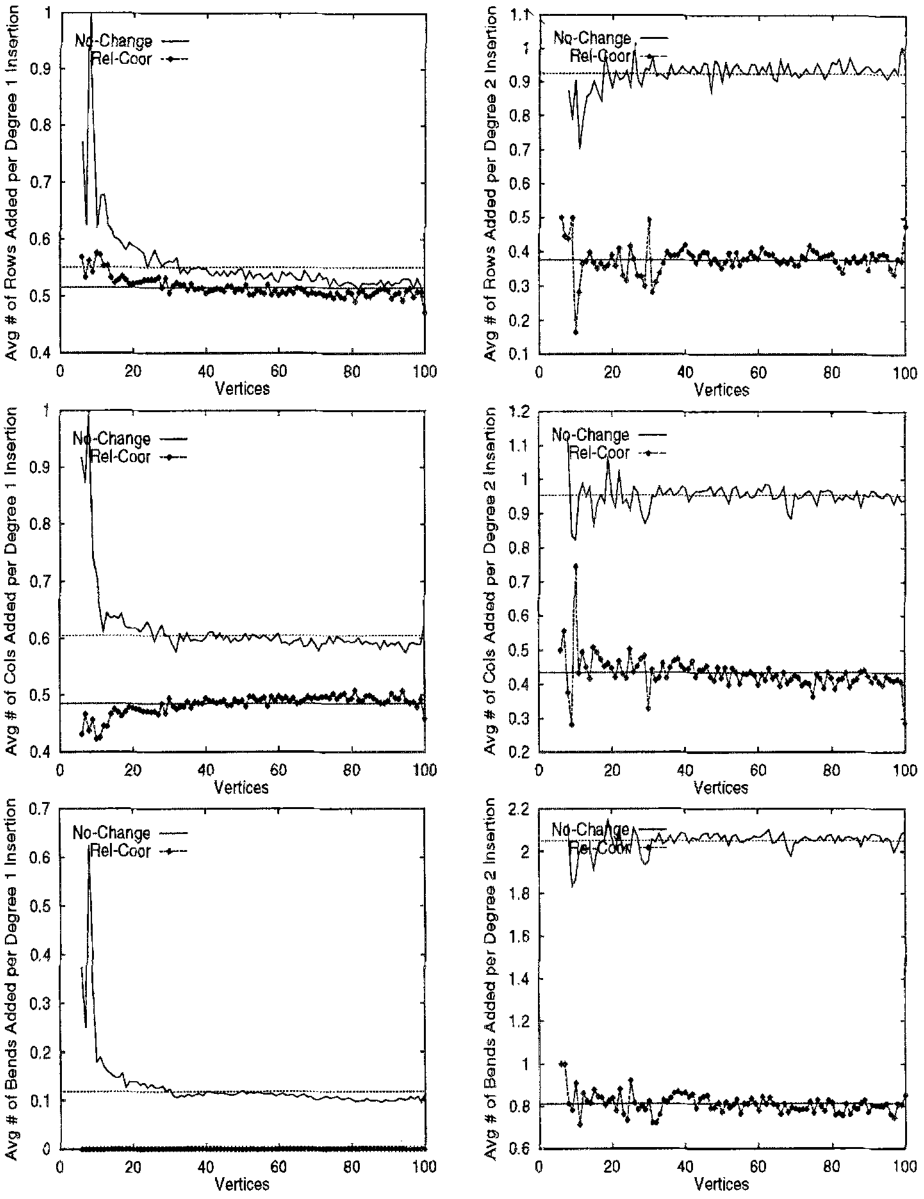

Fig. 5. Average number of rows, columns and bends added per degree one and two vertex insertion.

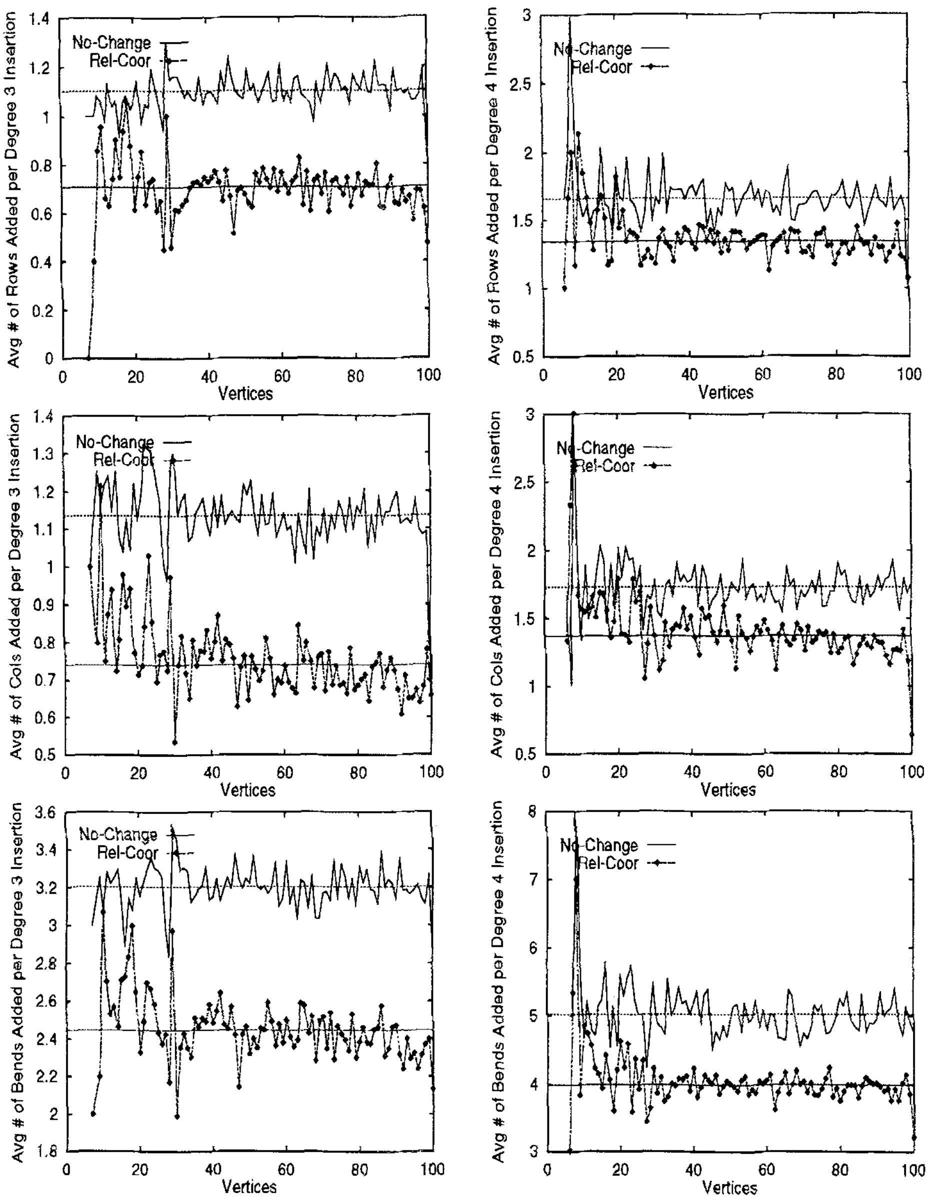

Fig. 6. Average number of rows, columns and bends added per degree three and four vertex insertion.

and Applications, vol. 4, no 5, 1994, pp. 235-282. Also available via anonymous `ftp` from `ftp.cs.brown.edu`, `gdbiblio.tex.Z` and `gdbiblio.ps.Z` in `/pub/papers/compgeo`.

5. G. DiBattista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari and F. Vargiu, An Experimental Comparison of Three Graph Drawing Algorithms, *Proc. of ACM Symp. on Computational Geometry*, pp. 306-315, 1995. The version of the paper with the four algorithms can be obtained from `http://www.cs.brown/people/rt`.
6. S. Even and G. Granot, Rectilinear Planar Drawings with Few Bends in Each Edge, *Tech. Report 797, Comp. Science Dept.*, Technion, Israel Inst. of Tech., 1994.
7. S. Even and R.E. Tarjan, Computing an st-numbering, *Theor. Comp. Sci. 2* (1976), pp. 339-344.
8. Michael Himsolt, Comparing and evaluating layout algorithms within GraphEd, *J. Visual Languages and Computing*, vol. 6, no. 3. pp.255-73, 1995.
9. Michael Himsolt, GraphEd: a graphical platform for the implementation of graph algorithms, *Proc. DIMACS Workshop GD '94, Lecture Notes in Comp. Sci. 894*, Springer-Verlag, 1994, pp. 182-193.
10. Goos Kant, Drawing planar graphs using the lmc-ordering, *Proc. 33th Ann. IEEE Symp. on Found. of Comp. Science*, 1992, pp. 101-110.
11. F. T. Leighton, New lower bound techniques for VLSI, *Proc. 22nd Ann. IEEE Symp. on Found. of Comp. Science*, 1981, pp. 1-12.
12. Charles E. Leiserson, Area-Efficient Graph Layouts (for VLSI), *Proc. 21st Ann. IEEE Symp. on Found. of Comp. Science*, 1980, pp. 270-281.
13. Thomas Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley and Sons, 1990.
14. 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.
15. 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.
16. A. Papakostas and I. G. Tollis, *Algorithms for Area-Efficient Orthogonal Drawings, Technical Report UTDCS-06-95*, The University of Texas at Dallas, 1995.
17. A. Papakostas and I. G. Tollis, Improved Algorithms and Bounds for Orthogonal Drawings, *Proc. DIMACS Workshop GD '94, LNCS 894*, Springer-Verlag, 1994, pp. 40-51.
18. A. Papakostas and I. G. Tollis, Issues in Interactive Orthogonal Graph Drawing, *Proc. of GD '95, LNCS 1027*, Springer-Verlag, 1995, pp. 419-430.
19. Markus Schäffter, Drawing Graphs on Rectangular Grids, *Discr. Appl. Math.* 63 (1995), pp. 75-89.
20. J. Storer, On minimal node-cost planar embeddings, *Networks* 14 (1984), pp. 181-212.
21. R. Tamassia, On embedding a graph in the grid with the minimum number of bends, *SIAM J. Comput.* 16 (1987), pp. 421-444.
22. R. Tamassia and I. Tollis, Planar Grid Embeddings in Linear Time, *IEEE Trans. on Circuits and Systems CAS-36* (1989), pp. 1230-1234.
23. Tom Sawyer Software Corp. GLT development group, *Graph Layout Toolkit User's Guide*, Berkeley, California, 1995.
24. Tom Sawyer Software Corp. GLT development group, *Graph Layout Toolkit Reference Manual* Berkeley, California, 1995.
25. L. Valiant, Universality Considerations in VLSI Circuits, *IEEE Trans. on Comp.*, vol. C-30, no 2, (1981), pp. 135-140.