Journal of Graph Algorithms and Applications
http://www.cs.brown.edu/publications/jgaa/
vol. 3, no. 2, pp. 1–23 (1999)

Experimental Comparison of Graph Drawing Algorithms for Cubic Graphs

Tiziana Calamoneri   *Simone Jannelli*   *Rossella Petreschi*
Dipartimento di Scienze dell'Informazione
Universitá di Roma "La Sapienza"
{calamo,petreschi}@dsi.uniroma1.it

Abstract

We report on the results of an experimental study in which we have compared the performances of three algorithms for drawing general cubic graphs on the bidimensional orthogonal grid. The comparison works on 18,000 randomly generated graphs with up to 300 vertices and analyzes the number of bends and crossings, the area, the edge length and the running time.

Communicated by R. Tamassia: submitted May 1997; revised May 1999.

The first author is supported by the Italian Research Council – CNR.

1 Introduction

An orthogonal grid drawing of a graph is a drawing such that the edges are polygonal chains consisting of horizontal and vertical segments and the vertices have integer coordinates. The graphs that admit such a drawing must have maximum degree 4. Among these graphs, cubic graphs (i.e. regular graphs of degree 3) and at most cubic graphs (i.e. graphs having bounded degree 3), constitute interesting and complex classes of graphs, despite their apparent simplicity. Indeed, cubic graphs are a natural model for a large number of real systems that interest many scientific fields such as Psychology, Probability, Economy, Physics, Geometry, and Computer Science. More in detail, some concrete examples are the study of the interaction of quantistic charged particles of high energy [4] and the study of flowgraphs to design hierarchical software metrics [21], see Figs. 1 and 2.

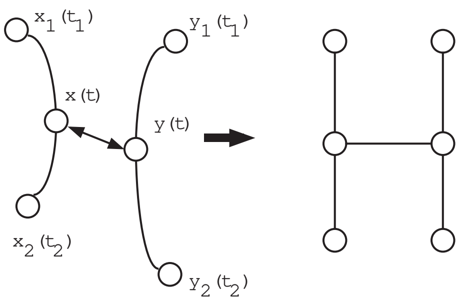

Figure 1: Interaction between two particles.

Furthermore, cubic graphs have been widely studied by many researchers, since they seem to be a threshold class of graphs, in the sense that they are the simplest graphs for which several fundamental problems are as difficult as in the general case [5, 14].

Several results on orthogonal drawing of cubic graphs have been presented [2, 6, 7, 11, 17, 18, 19, 20, 22]. However, none of the cited papers provides experimental results, although the interest in experimentally testing the performance of graph drawing algorithms has increased in the last years [8, 9, 10, 15, 16]. This paper tries to bridge this gap by presenting an experimental comparison of three orthogonal drawing algorithms for cubic graphs. The three selected algorithms are as follows: the first one, by Biedl and Kant [2], is a specialization of an algorithm for graphs with bounded degree 4; the second and third ones, by Calamoneri and Petreschi [6] and by Papakostas and Tollis [19], respectively, are specifically designed for cubic graphs.

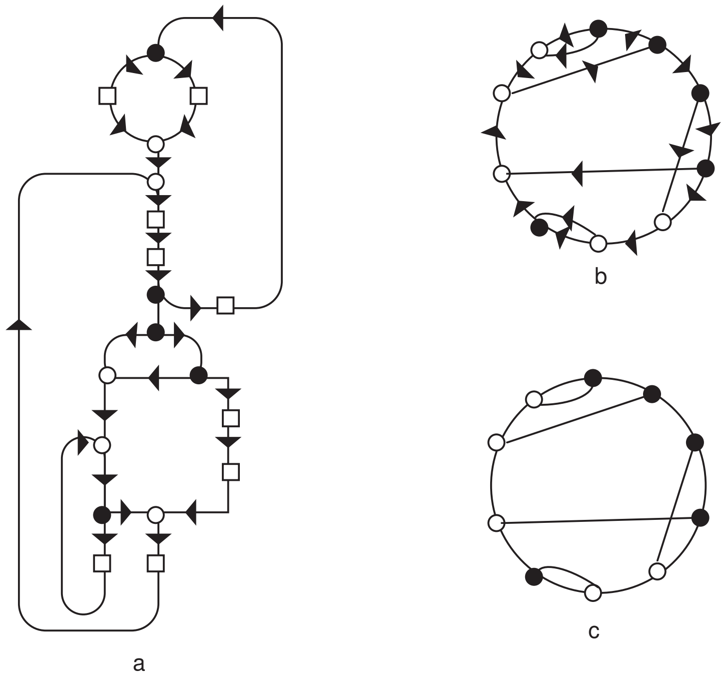

Figure 2: From a flowgraph to the underlying (undirected) cubic graph.

We have chosen these algorithms because they are homogeneous with respect to the input, the output and the leading idea. In particular, in our experimentation, no restriction on the input is required: neither planarity nor biconnectivity, nor a preliminary layout. This is the main reason why we have decided not to include in our experimentation algorithms such as those presented in [3, 17, 22], which deal with planar and/or triconnected graphs only. For what concerns the output, it is accepted to be non plane, even if the input graph is planar. The chosen algorithms add one vertex at a time according to an st-numbering [13]; therefore, the algorithm described in [20] has not been considered because it works on pairs of vertices. All the algorithms first work on biconnected components and then splice the drawings of the components to form a drawing of the entire graph. Thus, the results of our experiments are reported in two series of diagrams, one for biconnected graphs and the other for simply connected graphs.

Observe that the algorithm by Biedl and Kant is the only algorithm, among the chosen ones, not to be explicitly designed for cubic graphs. We have included this algorithm in the experimentation also to check if designing a drawing algorithm specifically for cubic graphs can improve its performance. The answer to this question is discussed in Section 5. Finally, the chosen algorithms achieve the best theoretical results known in the literature for a general at most cubic graph in minimizing the area, the total number of bends and the running time.

The rest of this paper is organized as follows: in Section 2, some basic definitions are given. In Section 3, the three analyzed drawing algorithms are sketched outlining the similarities and underlining the differences. Section 4 contains the results of the experiments (summarized in twelve diagrams) and a comparative analysis of the performance of the algorithms. Finally, in Section 5, we show drawings of the same graph generated by the three algorithms and offer concluding remarks.

2 Basic Definitions

In this section we introduce some preliminary concepts related to the topics dealt in this paper.

Definition 1 *Let $\mathcal{G} = (V, E)$ be a graph, $n = |V|$ and $m = |E|$. An* orthogonal drawing *of $\mathcal{G}$ is a drawing of $\mathcal{G}$ in the plane such that all edges are drawn as sequences of horizontal and vertical segments. The edges are not allowed to overlap for any distance (although a vertical segment may cross a horizontal one). In addition, the edges cannot cross vertices that are not their extremes. A point where the drawing of an edge changes its direction is called a* bend *of this edge. This drawing is said to be a* drawing in the (rectangular) grid *if all vertices and bends are at integer coordinates.*

In the following the four directions on the grid with respect to each vertex are distinguished and a direction is called *free* with respect to a vertex if no edge is present on it.

Definition 2 *Let k be a non negative integer. A k*-bend drawing *of a graph $\mathcal{G}$ is a grid drawing of $\mathcal{G}$ in which every edge contains at most k bends.*

Definition 3 *[1] Let $\mathcal{G}$ be a connected graph. A vertex a is said to be an* articulation vertex *of $\mathcal{G}$ if there exist vertices v and w such that v, w and a are distinct, and every path between v and w contains vertex a. A connected graph $\mathcal{G}$ is said to be* biconnected *if for every distinct triplet of vertices v, w, a, there exists a path between v and w and not containing a. Thus, a connected graph is biconnected if and only if it has no articulation vertex.*

Observe that a nontrivial biconnected graph cannot have any vertex of degree 1. When we deal with biconnected graphs, we will assume that the input graph is cubic. Indeed, if it is at most cubic, we can first eliminate each vertex of degree 2 by merging its incident edges. After drawing the resulting cubic

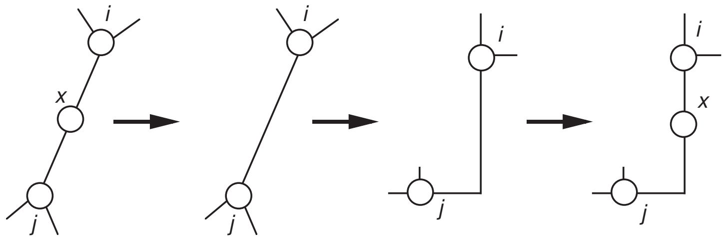

Figure 3: Transformation from biconnected at most cubic graph to cubic graph and vice-versa.

graph, we can reinsert a vertex of degree 2 adjacent to vertices i and j by simply placing a "dot" along the drawing of edge (i, j) (see Fig. 3).

Definition 4 *[13] Given any edge* $\{s, t\}$ *of a biconnected graph* $\mathcal{G} = (V, E)$*, a function* $g : V \to \{1, 2, \ldots, n\}$ *is called an* st-numbering *if the following conditions hold:*

- $g(u) \neq g(v)$ *for* $u \neq v$
- $g(s) = 1$
- $g(t) = |V| = n$
- *for every* $v \in V - \{s, t\}$ *there are vertices* u *and* w *such that* $g(u) < g(v) < g(w)$ *and* $\{u, v\}, \{w, v\} \in E$.

From now on, we shall refer to the vertices of a biconnected st-numbered cubic graph $\mathcal{G} = (V, E)$ by their st-numbers.

3 The Algorithms Under Evaluation

In order to make this paper self-contained, we shall briefly describe the three algorithms we compare in the following and we will refer to them by means of the first letters of their authors, i.e. **BK**[2], **CP**[6] and **PT**[19], respectively.

To make easier the reading of this section, we deal only with the key aspects of the algorithms and we skip technical details. Furthermore, we present the common characteristics and the differences between the algorithms.

- The input of these algorithms is a general simple loopless at most cubic graph $\mathcal{G}$ (neither planarity nor 2- or 3-connectivity are required). Since

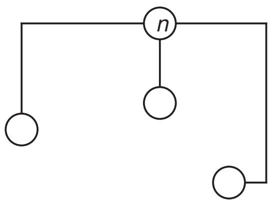

Figure 4: Insertion of vertex n in Algorithm **PT**.

Algorithm **PT** is presented in [19] only for biconnected graphs, in our comparisons we added to **PT** a second phase able to handle separated biconnected components and to splice them. Namely, we utilize the procedure detailed by Algorithm **CP** to assemble the biconnected components.

- In order to obtain better upper bounds for the optimization functions, the algorithms do not guarantee the output to be plane even if the input is planar. Actually, Algorithm **BK** has a behavior different from the other two since it gets a plane drawing if a plane layout of $\mathcal{G}$ is given as input. For the sake of uniformity, we have not considered this case.

- With respect to the number of bends per edge, all three algorithms reach a constant value, either 1 or 2. In particular, the output of algorithm **CP** is exactly a 1-bend drawing of $\mathcal{G}$; the output of **PT** is 1-bend except at most one edge incident to n (Fig. 4) while the output of **BK** is 2-bend. Notice that the introduction of 2 bends per edge arises from the fact that this algorithm has been designed for graphs with bounded degree 4.

- Both Algorithm **CP** and **PT** use the concept of *movement of bends*, already introduced in [18]. The movement of a bend consists of moving a vertex k along an edge in order to change its free directions (Fig. 5). This has also the effect of changing the position of a bend, while the total number of bends stays the same.

  In Algorithm **CP** it is possible to change the free directions of a vertex also through a *rotation*. This operation does not guarantee the invariance of the number of bends; in Fig. 6 a possible rotation is shown.

- All the algorithms divide the input graph into its biconnected components and run a pre-processing phase computing an st-numbering. Then, one by one, each biconnected component is drawn; finally, the whole drawing of the graph is obtained by connecting the drawings of the different components.

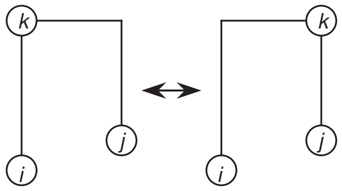

Figure 5: Movement of a bend.

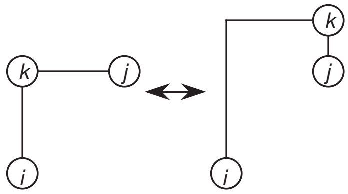

Figure 6: Rotation of a vertex.

- The basic idea of the first phase consists of adding the vertices to the drawing one at a time ordered according to an st-numbering. This construction is made for each biconnected component and it is based on the existence, in an st-numbered graph, of edges $\{1,2\}$, $\{n-1,n\}$ and, for each vertex j (where $j \neq 1, n$), of edges $\{j,i\}$ and $\{j,l\}$ such that $i < j < l$.

  Let $\mathcal{G}_k = (V_k, E_k)$ be the subgraph induced by the first k vertices in the st-numbering ($V_k = \{1,2,\ldots,k\}$) and D_k a drawing for $\mathcal{G}_k$. During the k-th step, vertex k and the edges $\{i,k\}$, with $i < k$, are added to D_{k-1}. Each author has his/her own rules to lay vertex k out, and they constitute the main reason of the difference between the three algorithms. In particular, Algorithm **BK**, at each step, associates with a free direction of k an edge incident to k that will be drawn in a successive step (Fig. 7).

  Algorithms **CP** and **PT** draw k in such a way that at least one and at most two edges do not introduce any bend.

- The final difference we want to highlight is the addition of vertex n to D_{n-1}. Algorithm **CP** places vertex n and its edges on the grid as a *comb graph* (Fig. 8) after having possibly operated a rotation or a movement of some bends.

  Beside the comb graph, Algorithm **PT** must accept also the configuration of Fig. 4 for n because this algorithm does not provide any rotation.

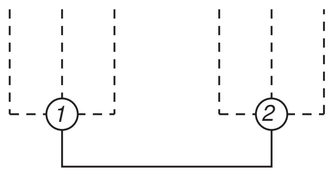

Figure 7: Insertion of vertices 1 and 2 in Algorithm **BK**.

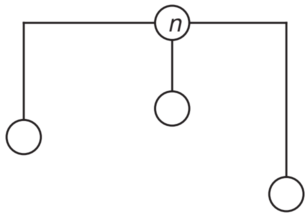

Figure 8: A comb graph.

Finally, in Algorithm **BK** the addition of vertex n induces the drawing of the last three edges dotted in the previous steps. Then, the insertion of n and its edges induces a comb graph-like structure in which the teeth are not connected to the n's adjacent vertices but to their dotted edges.

- The second phase of the algorithms deals with general at most cubic graphs and uses a decomposition into biconnected components.

  The common idea is to draw separately each component and then to join the drawings of the components through their connections. In this step, both Algorithms **BK** and **CP** work in a recursive way, but they use different methods. The main difference is that in **BK** the bridges (separating edges) are removed, while in **CP** the articulation vertices are deleted. This difference is reflected in the way the biconnected components are later spliced together. Indeed, in **BK** only edges whose endpoints are already drawn must be added, but this requires that at least one of these endpoints be on the boundary of the drawing of its biconnected component. In **CP** the articulation vertices and their incident edges are added and no constraint is required for their adjacent vertices. However, some directions towards the boundary of the current drawing are always left free (either directly, or after a movement of a bend, or after a rotation).

We conclude this section summarizing the worst-case theoretical results achieved by the three algorithms.

- Regarding the area of the drawing, for **BK** it is $(\frac{n}{2}+2) \times (n-2)$ if the input is biconnected and $\frac{3}{4}n \times \frac{3}{4}n$ otherwise. In **CP** and **PT** the area is $\frac{n}{2} \times \frac{n}{2}$ and $(\frac{n}{2}+1) \times \frac{n}{2}$, respectively, both for biconnected and simply connected graphs.
- Regarding the total number of bends, Algorithm **BK** achieves $n+1$ and n for biconnected and simply connected graphs, respectively. In Algorithm **CP** the bound is $\frac{n}{2}+1$ in the biconnected case and it is improved by 1 in the connected case. Finally, Algorithm **PT** draws a graph with at most $\frac{n}{2}+3$ bends.
- **CP** constructs a 1-bend drawing **PT** constructs a 2-bend drawing except for one edge with two bends, and **BK** constructs a 2-bend drawing.
- All three algorithms run in linear time.

Notice that for **BK** we have indicated better bounds than those reported in [2] since we consider only almost cubic graphs.

4 Analysis of the Experimental Results

4.1 Performance Indicators

The experiments provide a detailed quantitative evaluation of the performance of the three algorithms, both from an "aesthetic" point of view (area, number of crossings, number of bends, edge length) and from a computational one. Namely, the considered performance indicators for each algorithm are:

- **Area:** area of the smallest rectangle with horizontal and vertical sides covering the drawing;
- **Bends:** total number of bends;
- **Crossings:** total number of edge-crossings;
- **MaxEdgeLen:** maximum length of any edge;
- **TotalEdgeLen:** total edge length;
- **Time:** CPU time (measured in milliseconds) to compute the geometry of the drawing, including the st-numbering computation but excluding both the graph generation time and the time for drawing on screen. The resolution is 0.2 ms.

Observe that we do not consider **MaxEdgeBends** (maximum number of bends on any edge) because of the discussed theoretical results of the algorithms.

4.2 Experimental Setting

In order to compare the performance of the three algorithms outlined in Section 3, we have tested them on 18,000 randomly generated graphs with with number of vertices in the range $10 \ldots 300$. The results of the experimentation are presented in graphical form and reported in Figs. 10–21. We produced the statistics by grouping the graphs by number of vertices. We decided not to group the graphs by the number of edges since all at most cubic graphs are sparse and all our sample graphs have more or less the same number of edges for the same number of vertices. Therefore, data points in the diagrams are averages over 300 graphs with the same number of vertices. Moreover, we give two separate series of diagrams, one for biconnected graphs and one for simply connected graphs.

Biconnected graphs were generated by adding, one at a time, an edge between two randomly chosen vertices with degree less than three. The generation procedure ends when the graph becomes connected and the number m of inserted edges is such that $2.75n/2 \leq m \leq 3n/2$. At the end of the procedure, if the graph is biconnected it is added to the experimental suite, otherwise it is discarded. Observe that this *naive* method to impose biconnectivity is not computationally expensive because almost all at most cubic graphs with more than $2.75n/2$ edges are biconnected.

Simply connected graphs were generated by merging biconnected components: the number of biconnected components is randomly chosen in the range $2 \ldots n/20$ and, accordingly, their size is randomly decided. The biconnected components are generated using the previous procedure and then they are merged together avoiding the creation of cycles.

4.3 Implementation

The algorithms have been implemented in C and have been run on a computer with a Pentium MMX 166 MHz processor.

A technical problem faced during the implementation of all three algorithms is the maintenance of the vertex coordinates during the incremental construction of the drawing. Indeed, when adding a new row or column to the drawing to accommodate the placement of the next vertex, the coordinates of many vertices and bends may need to be updated. This implies that it is not possible to compute the final coordinates of each vertex as it is added to the drawing.

By representing the "virtual" coordinates by means of two sorted lists, this problem is reduced to the problem of maintaining order in a list, for which an optimal linear-time solution using sophisticated techniques is given by Dietz and Sleator [12]. In our implementation we have opted instead for a simple linear-time method that experimentally works very well but that is not guaranteed to always give correct results, although it never failed in our experimentations.

Our method uses temporary floating-point coordinates for the vertices during the execution of the algorithm. Namely, consider the insertion of a new vertex k on a new row (column) between the rows (columns) where its two adjacent vertices i and j lie. W.l.o.g., suppose that $y_j < y_i$ ($x_j < x_i$) and let y_b (x_b) be the first occupied row (column) in the range $[y_j, y_i]$ ($[x_j, x_i]$) after y_j (x_j). This is the value immediately following y_j (x_j) in the sorted list of y (x) coordinates. We assign to vertex k y- (x-)coordinate equal to the average of y_j and y_b (x_j and x_b) and update the sorted list by inserting $y_k(x_k)$ is after $y_j(x_j)$ and before y_b (x_b). At the end of the construction of the drawing we transform the temporary floating-point coordinates into final integer coordinates in linear time, with a simple prefix sum algorithm.

The above algorithm may yield incorrect results if there is a long succession of vertices $k_1, \ldots, k_s$ with the following property:

- k_1 lies between i and j;
- k_r ($r = 2, \ldots, s$) lies between k_{r-1} and i.

For typical processors, errors occur if $s \geq 36$, an event that never occurred in our experiments.

Another minor problem was that our implementation of Algorithm **PT** failed on some input graphs. This is due to the fact that the original description of the algorithm does not mention how to deal with certain special cases (see Fig. 9).

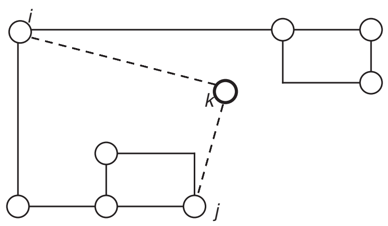

Figure 9: Example of ignored configuration in Algorithm **PT**.

4.4 Analysis

In this section, we discuss the results of the experiments and analyze in detail the various performance indicators.

- The observed practical behavior of the algorithms is consistent with their theoretical properties. Actually, the experimental results for some performance indicators, such as the number of bends for connected graphs, are slightly better than the theoretical worst-case bounds.

- There is no significant difference between the results for biconnected and simply connected graphs on the area, maximum edge length and running time. On the contrary, there is a wide gap between the results for connected and biconnected graphs on the number of bends, the number of crossings and the total edge length. Justifications of this behavior are given in the following.

- From a computational point of view, drawing cubic graphs is a well solved problem: even the slowest algorithm takes only 6.8 ms to draw the largest graphs (300 vertices).

We now discuss in detail each performance indicator.

- **Area:** (Figs. 10 and 11) Algorithm **PT** yields average area very close to the theoretical bound of $n^2/4 + o(n^2)$, both in the connected and in the biconnected case.

  The area of drawings computed by Algorithm **BK** is $n^2/2$ both in the connected and biconnected case. But, while this value matches the theoretical value of the area for biconnected graphs, it is slightly better than $9n^2/16$, which is the theoretical bound for connected graphs. This suggests that the configurations leading to the larger theoretical value for the area in the connected case are rare.

  Finally, since Algorithm **CP** tries to draw vertices without adding new bends and therefore without adding new rows and columns, it reaches a slightly better area bound (about $n^2/4.88$ instead of $n^2/4$) both in the connected and in the biconnected case.

- **Bends:** (Figs. 12 and 13) For what concerns the total number of bends, it is clear from the diagrams that all three algorithms generate fewer bends in the simply connected case than in the biconnected one.

  Namely, **BK** generates a total number of bends that is close to n for biconnected graphs and close to $n/1.25$ for connected graphs. Algorithm **PT** approaches $n/2$ and $n/2.5$ in the biconnected and connected case, respectively. Algorithm **CP** works better from this point of view, since it generates about $n/2.44$ and $n/3$ bends if the input graph is biconnected and connected, respectively. Also, the ratio between the averages of the numbers of bends in the biconnected and connected case is about 1.22 in all algorithms: the better results in the biconnected case can be justified by the lower number of edges on average.

- **Crossings:** (Figs. 14 and 15) The total number of crossings does not appear to be an increasing function of the number of vertices because the number of crossings is highly random. Since edges connecting the biconnected components are always drawn without crossings, their number

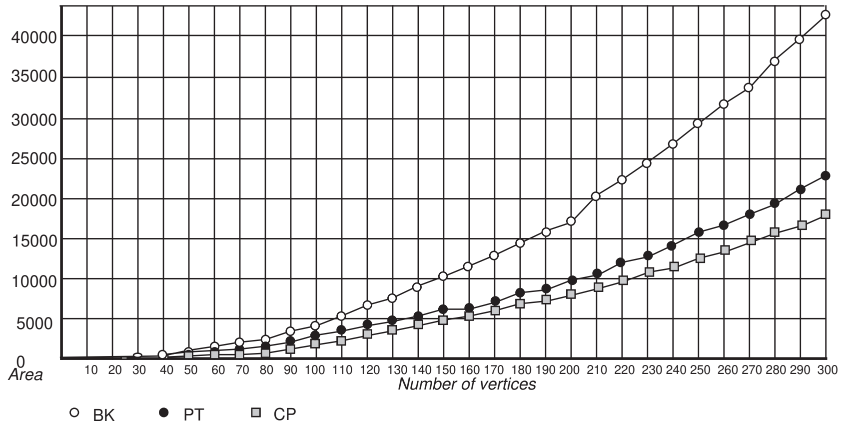

Figure 10: Area versus number of vertices for simply connected graphs.

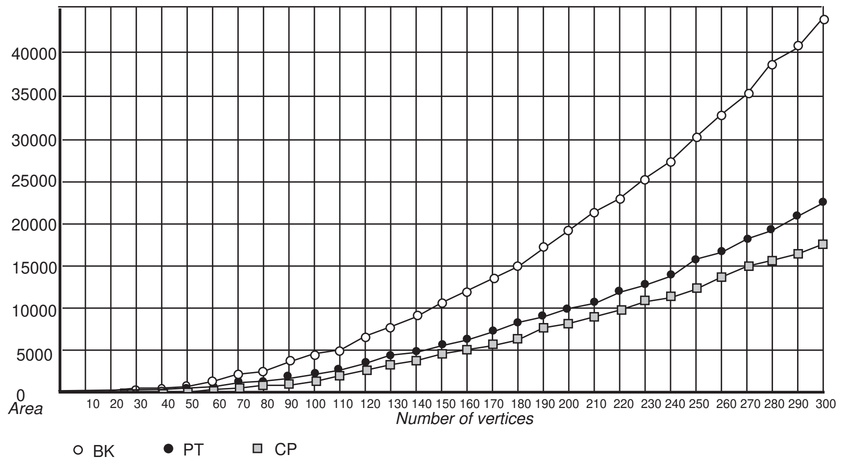

Figure 11: Area versus number of vertices for biconnected graphs.

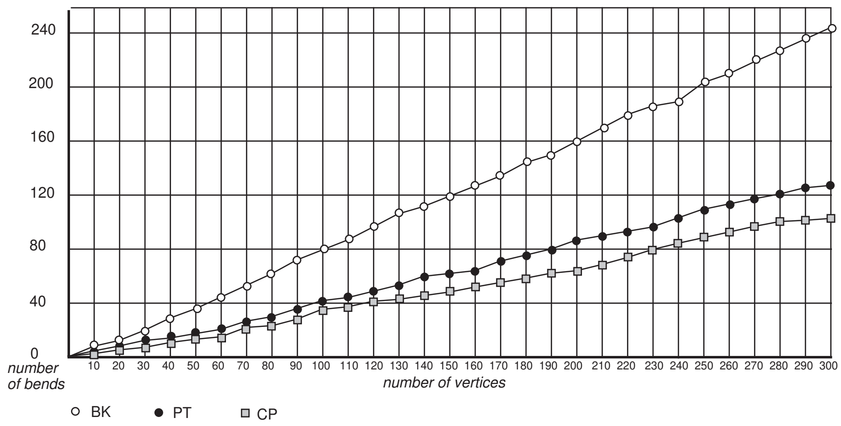

Figure 12: Total number of bends versus number of vertices for simply connected graphs.

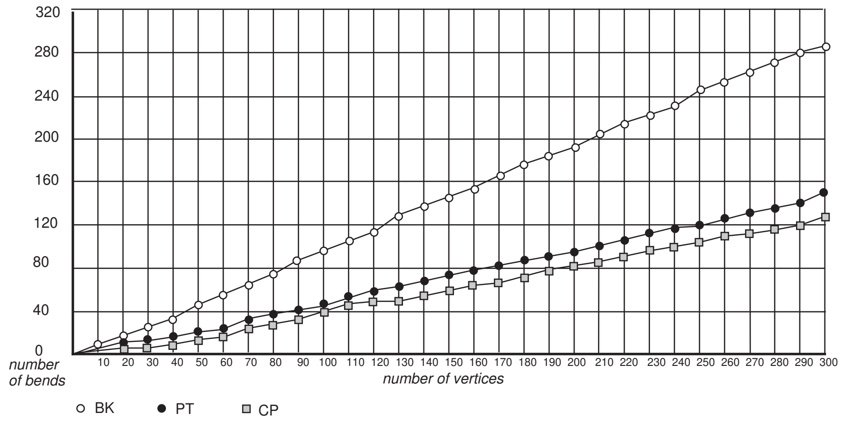

Figure 13: Total number of bends versus number of vertices for biconnected graphs.

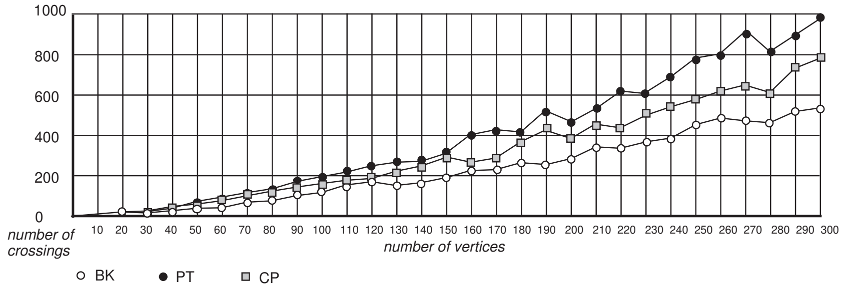

Figure 14: Crossings versus number of vertices for simply connected graphs.

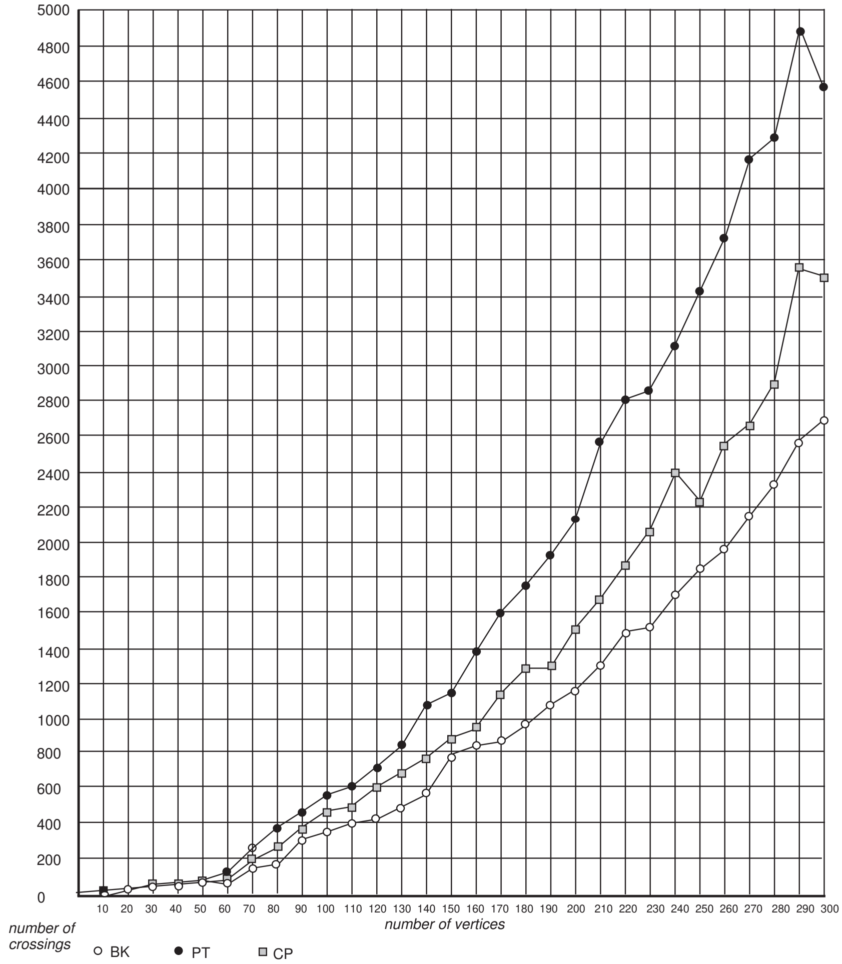

Figure 15: Crossings versus number of vertices for biconnected graphs.

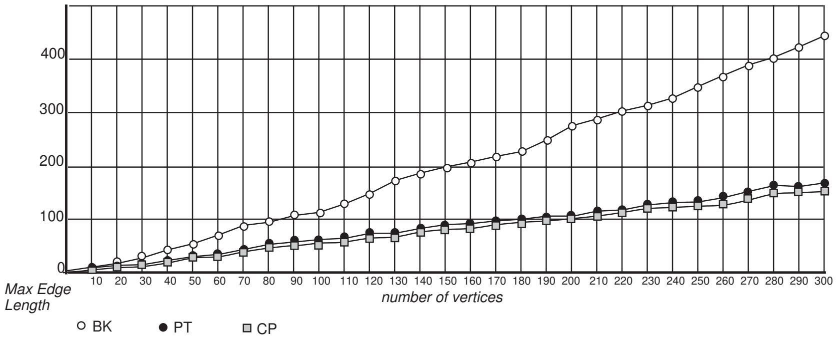

Figure 16: Maximum edge length versus number of vertices for simply connected graphs.

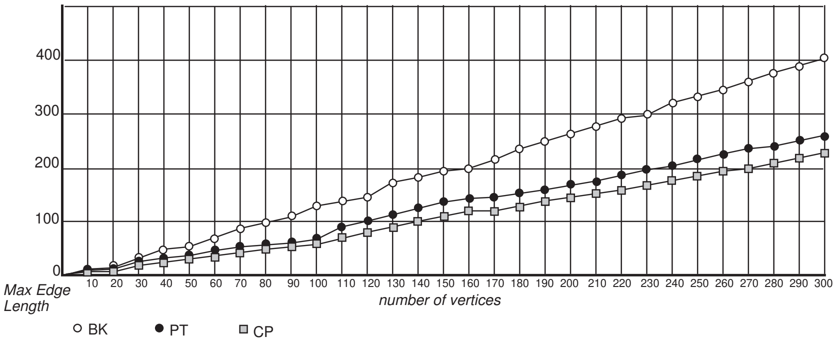

Figure 17: Maximum edge length versus number of vertices for biconnected graphs.

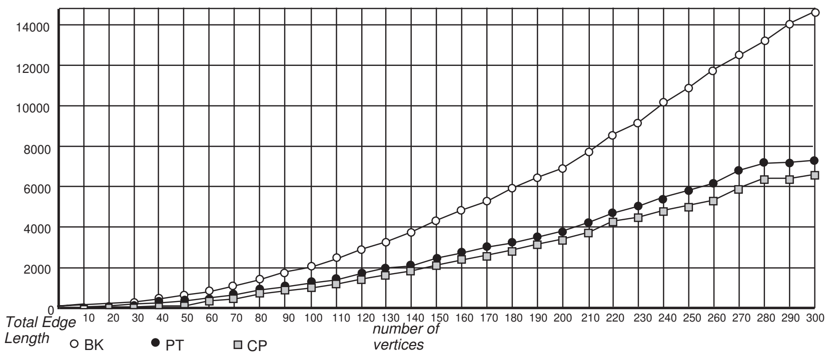

Figure 18: Total edge length versus number of vertices for simply connected graphs.

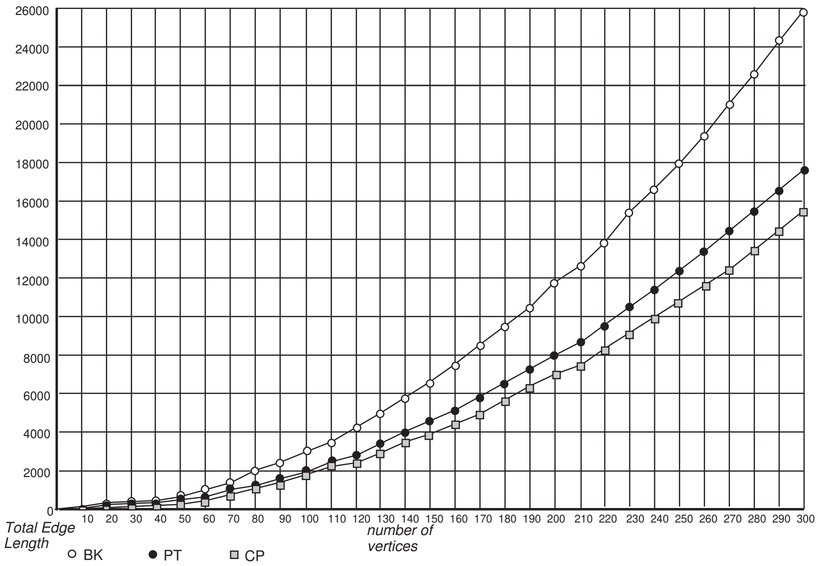

Figure 19: Total edge length versus number of vertices for biconnected graphs.

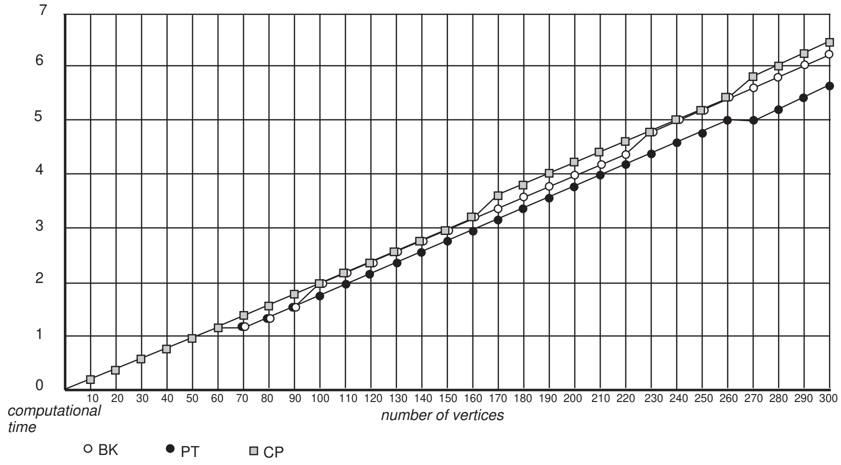

Figure 20: Computational time (in ms) versus number of vertices for simply connected graphs.

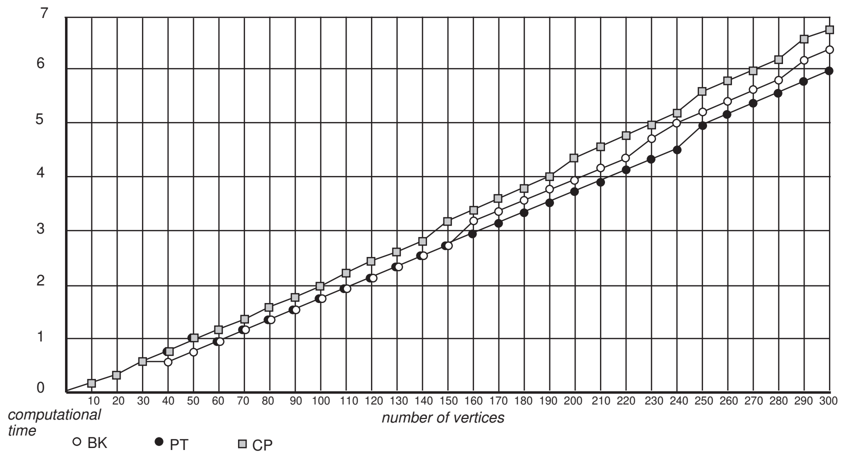

Figure 21: Computational time (in ms) versus number of vertices for biconnected graphs.

is consistently smaller when the graphs are simply connected rather than when they are biconnected. The average values in the biconnected case are $n^2/33, n^2/25.8$ and $n^2/17.6$ for Algorithms **BK**, **CP** and **PT**, respectively. The good result of **BK** can be explained as a consequence of the fact that it is designed to support the drawing of planar graphs without crossings, and therefore its number of crossings is lower, to the detriment of the area.

- **MaxEdgeLen:** (Figs. 16 and 17) and **TotEdgeLen:** (Figs. 18 and 19) The experimental values of the maximum edge length are $1.4n, 0.5n$ and $0.6n$ in the connected case and $1.36n, 0.77n$ and $0.88n$ in the biconnected case for Algorithms **BK**, **CP** and **PT**, respectively. The values of the total edge length are $1/4n^2, 1/13n^2$ and $1/12n^2$ in the connected case and $2/7n^2, 1/6n^2$ and $1/5n^2$ in the biconnected case for Algorithms **BK**, **CP** and **PT**, respectively.

  We have no theoretical bounds to compare these results to, but we observe that all the algorithms have considerably worse performance in the biconnected case. This is because when two biconnected components are merged, some edges are lengthened and influence these performance indicators.

- **Computational time:** (Figs. 20 and 21) All three algorithms are very fast: **PT** runs in $n/54$ and $n/50$ ms on connected and biconnected graphs, respectively; **BK** and **CP** need $n/48(n/47)$ and $n/47(n/44)$ ms to draw an n vertex connected and biconnected cubic graph, respectively.

5 Conclusions

Following the trend of comparing the theoretical properties of graph drawing algorithms with experimental results and of estimating performance indicators, in this paper we analyze three algorithms for constructing orthogonal grid drawings of cubic graphs. The main conclusion of our work is that this problem is well solved both from a theoretical and an experimental point of view.

Analyzing more in detail the results of the experimentation, we see that the best algorithm depends on the criteria used. If the most important objective is either the area or the maximum number of bends or the edge length, the best algorithm is **CP**. According to the computational time **PT** beats the other two. A partial justification of why **BK** performs best only regarding the number of crossings lies in its goals: to draw graphs with maximum degree 4 and to focus on planar graphs.

To conclude, we give an example of the layouts produced by the three algorithms by running them on the graph of Fig. 22. Figs. 23, 24 and 25 show the drawings generated by Algorithms **BK**, **CP** and **PT**, respectively.

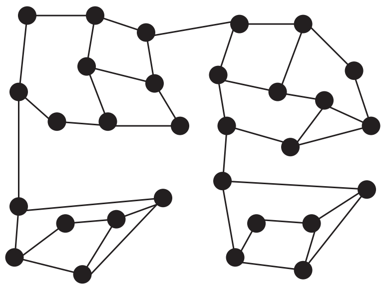

Figure 22: A graph with 30 vertices.

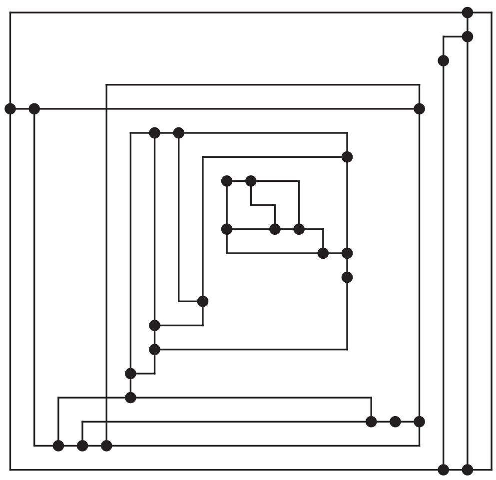

Figure 23: Graph of Fig. 22 drawn by Algorithm **BK**.

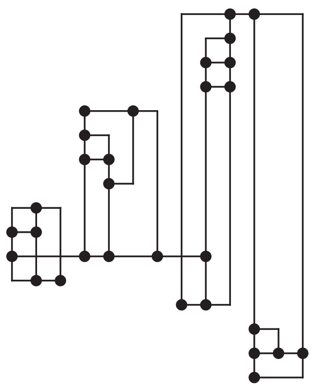

Figure 24: Graph of Fig 22 drawn by Algorithm **CP**.

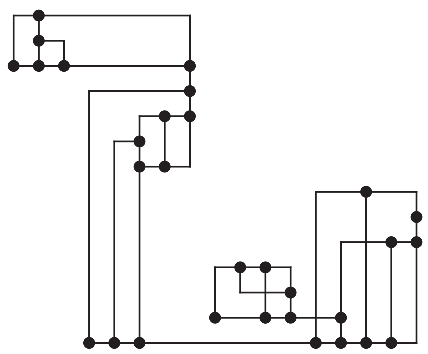

Figure 25: Graph of Fig 22 drawn by Algorithm **PT**.

References

[1] A. Aho, J. K. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms.* Addison Wesley, Reading, MA, 1973.

[2] T. Biedl, and G. Kant. A Better Heuristic for Orthogonal Graph Drawings. *Comput. Geom. Theory Appl.*, 9:159–180, 1998. Also in *Proc. European Symposium on Algorithms (ESA '94), Lectures Notes in Computer Science 855, Springer-Verlag*, pages 24–35, 1994.

[3] T. Biedl. Optimal plane orthogonal drawings for triconnected graphs. In *Proc. Scandinavian Workshop on Algorithm Theory (SWAT'96), Lectures Notes in Computer Science 1097, Springer-Verlag*, pages 333–344, 1996.

[4] J.D. Bjorken, and S.D. Drell. *Relativistic Quantum Fields.* Mc-Graw Hill, New York, 1965.

[5] T. Calamoneri. *Does Cubicity Help to Solve Problems?* Ph.D. Thesis, University of Rome "La Sapienza",XI-2-97, 1997.

[6] T. Calamoneri, and R. Petreschi. An Efficient Orthogonal Grid Drawing Algorithm for Cubic Graphs. In *Proc. First Annual International Conference on Computing and Combinatorics (COCOON '95), Lectures Notes in Computer Science 959, Springer-Verlag*, pages 31–40, 1995.

[7] T. Calamoneri, and R. Petreschi. Orthogonal Drawing Cubic Graphs in Parallel. *J. Parallel and Distr. Comput.*, 55:94–108, 1998.

[8] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An Experimental Comparison of Three Graph Drawing Algorithms. In *ACM Symposium on Computational Geometry, ACM*, pages 306–315, 1995.

[9] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An Experimental Comparison of Four Graph Drawing Algorithms. *Comp. Geom.*, 7(5-6):303–325, 1997.

[10] G. Di Battista, A. Garg, G. Liotta, A. Parise, R. Tamassia, and E. Tassinari. Drawing Directed Acyclic Graphs: An Experimental Study. In *Proc. Graph Drawing (GD'96), Lectures Notes in Computer Science 1190, Springer-Verlag*, pages 76–91, 1996.

[11] G. Di Battista, G. Liotta, and F. Vargiu. Spirality and Optimal Orthogonal Drawings. *SIAM J. Comput.*, 27(6):1764–1811, 1998.

[12] P. F. Dietz, and D. D. Sleator. Two algorithms for maintaining order in a list. In *Proc. 19th ACM Symp. on Theor. Comp. Sci. (STOC '87)*, 2, pages 436–441, 1976.

[13] S. Even, and R. E. Tarjan. Computing an st-numbering. *Theoret. Comp. Sci.*, 365–372, 1987.

[14] R. Greenlaw, and R. Petreschi. Cubic graphs. *ACM Computing Surveys*, 27 (4):471–495, 1995.

[15] M. Himsolt. Comparing and evaluating layout algorithms within *GraphEd. J. Visual Lang. Comput. (special issue on Graph Visualization)*, 6(3):255-273, 1995.

[16] S. Jones, P. Eades, A. Moran, N. Ward, G. Delott, and R. Tamassia. A note on planar graph drawing algorithms. Tech. Rep. 216, Department of Computer Science, University of Queensland, 1991.

[17] G. Kant. Drawing Planar Graphs Using the canonical ordering. In *Proc. 33th Ann. IEEE Symp. on Found. of Comp. Science (FOCS '92)*, pages 101–110, 1992. Also in *Algorithmica - Special Issue on Graph Drawing*, 16: 4–32, 1996.

[18] Y. Liu, P. Marchioro, and R. Petreschi. At most single bend embedding of cubic graphs. Tech. Rep. SI 92/01 Dept. of Comp. Science University of Rome "La Sapienza", 1992. Also in *Applied Mathematics (Chin. Journ.)*, 9/B/2,127–142, 1994.

[19] A. Papakostas, and I.G. Tollis. Improved Algorithms and Bounds for Orthogonal Drawings. In *Proc.Graph Drawing (GD'94), Lectures Notes in Computer Science 894, Springer-Verlag*, pages 40–51, 1994.

[20] A. Papakostas, and I.G. Tollis. Algorithms for Area-Efficient Orthogonal Drawings. *Comput. Geom. Theory Appl.*, 9:83–110, 1998.

[21] R.E. Prather. Design and Analysis of Hierarchical Software Metrics. *ACM Computing Surveys*, 27 (4):497–518, 1995.

[22] M. S. Rahman, S. Nakano, and T. Nishizeki. A Linear-Time Algorithm for Orthogonal Drawings of Triconnected Cubic Plane Graphs with the Minimum Number of Bends. In *Proc. Graph Drawing (GD'97), Lectures Notes in Computer Science 1353*, Springer-Verlag , pages 99–110, 1997.