Confluent Drawings: Visualizing Non-planar Diagrams in a Planar Way*

Matthew Dickerson[1], David Eppstein[2], Michael T. Goodrich[2], and Jeremy Yu Meng[2]

[1] Middlebury College, Middlebury, VT 05753, USA dickerso@middlebury.edu
[2] University of California, Irvine, Irvine, CA 92697, USA
{eppstein, goodrich, ymeng}@ics.uci.edu

Abstract. We introduce a new approach for drawing diagrams. Our approach is to use a technique we call *confluent drawing* for visualizing non-planar graphs in a planar way. This approach allows us to draw, in a crossing-free manner, graphs—such as software interaction diagrams—that would normally have many crossings. The main idea of this approach is quite simple: we allow groups of edges to be merged together and drawn as "tracks" (similar to train tracks). Producing such confluent diagrams automatically from a graph with many crossings is quite challenging, however, so we offer two heuristic algorithms to test if a non-planar graph can be drawn efficiently in a confluent way. In addition, we identify several large classes of graphs that can be completely categorized as being either confluently drawable or confluently non-drawable.

1 Introduction

In most graph visualization applications, graphs are often drawn in a standard way: the vertices of a graph are drawn as simple shapes, such as circles or boxes, and the edges are drawn as individual curves connecting pairs of these shapes (e.g., see [12,13,22]).

Related Prior Work. There are several aesthetic criteria that have been explored algorithmically in the area of graph drawing (e.g., see [12,13,22]). Examples of aesthetic goals designed to facilitate readability include minimizing edge crossings, minimizing a drawing's area, minimizing bends, and achieving good separation of vertices, edges, and angles. Of all of these criteria, however, the arguably most important is to minimize edge crossings, since crossing edges tend to confuse the eye when one is viewing adjacency relationships. Indeed, an experimental analysis by Purchase [31] suggests that edge-crossing minimization [19, 20,25] is the most important aesthetic criteria for visualizing graphs. Ideally, we would like drawings that have no edge crossings at all.

* This is an extended abstract. The full version of this paper can be found at http://arxiv.org/abs/cs.CG/0212046. Work by the second author is supported by NSF grant CCR-9912338. Work by the third and the forth author is supported by NSF Grants CCR-0098068, CCR-0225642, and DUE-0231467.

G. Liotta (Ed.): GD 2003, LNCS 2912, pp. 1–12, 2004.

Graphs that can be drawn in the standard way in the plane without edge crossings are called *planar graphs* [28], and there are a number of existing efficient algorithms for producing crossing-free drawings of planar graphs (e.g., see [8,9,11,34,6,21,36]). Unfortunately, most graphs are not planar; hence, most graphs cannot be drawn in the standard way without edge crossings, and such non-planar graphs seem to be common in many applications. There are some heuristic algorithms for minimizing edge crossings of non-planar graphs (e.g., see [19,26,20,25]), but the general problem of drawing a non-planar graph in a standard way that minimizes edge-crossings is NP-hard [16]. Thus, we cannot expect an efficient algorithm for drawing non-planar graphs so as to minimize edge crossings.

The technique of replacing complete bipartite subgraphs (bicliques) with star-like structures is used as *Edge Concentration* in [27] and *Factoring* in [5], both to reduced the number of edges in the original graphs. This technique has the desired side effect of reducing the number of crossings, however, its primary goal is to minimize the total number of edges, not to minimize the number of crossings. Furthermore, the time complexity of approximation algorithm given in [27] is not desirable. Recently Lin [24] proves that the optimization problem of edge concentration is NP-hard. A similar idea is used in [15] for weighted graph compressions, where cliques and bicliques are replaced with stars. It is shown that the general unit weight problem is essentially as hard to approximate as graph coloring and maximum clique. Again, [15] doesn't directly address the minimization of the number of crossings.

Our Results. Given the difficulty of edge-crossing minimization and the ubiquity of non-planar graphs, we explore in this paper a diagram visualization approach, called *confluent drawing*, that attempts to achieve the best of both worlds—it draws non-planar graphs in a planar way. Moreover, we provide two heuristic algorithms for producing confluent drawings for directed and undirected graphs, respectively, focusing on graphs with bounded arboricity.

The main idea of the confluent drawing approach for visualizing non-planar graphs in a planar way is quite simple—we merge edges into "tracks" so as to turn edge crossings into overlapping paths. (See Fig. 1.)

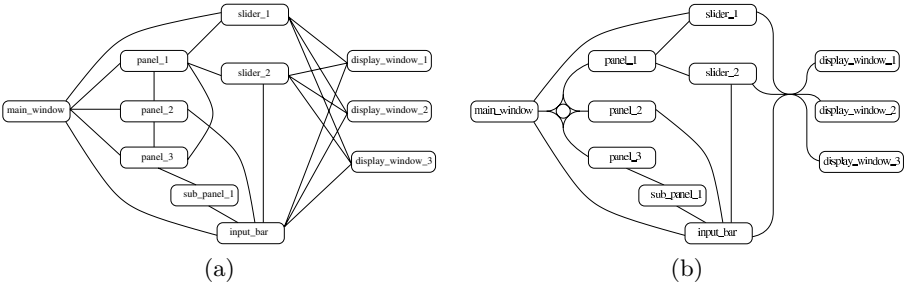

Fig. 1. An example of confluent drawing of an object-interaction diagram. We show a standard drawing in (a) and a confluent drawing in (b).

The resulting graphs are easy to read and comprehend, while also encapsulating a high degree of connectivity information. Although we are not familiar with any prior work on the automatic display of graphs using this confluent diagram approach, we have observed that some airlines use hand-crafted confluent diagrams to display their route maps. Diagrams similar to our confluent drawings have also been used by Penner and Harer [29] to study the topology of surfaces.

In addition to providing heuristic algorithms for recognizing and drawing confluent diagrams, we also show that there are large classes of non-planar graphs that can be drawn in a planar way using our confluent diagram approach.

2 Confluent Drawings

It is well-known that every non-planar graph contains a subgraph homeomorphic to the complete graph on five vertices, K_5, or the complete bipartite graph between two sets of three vertices, $K_{3,3}$ (e.g., see [3]). On the other hand, confluent drawings, with their ability to merge crossing edges into single tracks, can easily draw any $K_{n,m}$ or K_n in a planar way. Fig. 2 shows confluent drawings of $K_{3,3}$ and K_5.

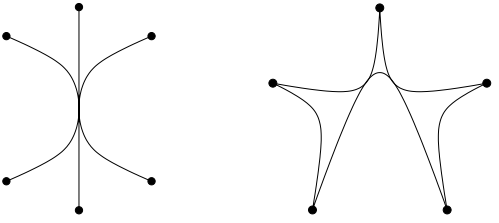

Fig. 2. Confluent drawings of $K_{3,3}$ and K_5.

A curve is *locally-monotone* if it contains no self intersections and no sharp turns, that is, it contains no point with left and right tangents that form an angle less than or equal to 90 degrees. Intuitively, a locally-monotone curve is like a single train track, which can make no sharp turns. Confluent drawings are a way to draw graphs in a planar manner by merging edges together into *tracks*, which are the unions of locally-monotone curves.

An undirected graph G is *confluent* if and only if there exists a drawing A such that:

- There is a one-to-one mapping between the vertices in G and A, so that, for each vertex $v \in V(G)$, there is a corresponding vertex $v' \in A$, which has a unique point placement in the plane.
- There is an edge (v_i, v_j) in $E(G)$ if and only if there is a locally-monotone curve e' connecting v'_i and v'_j in A.
- A is planar. That is, while locally-monotone curves in A can share overlapping portions, no two can cross.

Our definition does not allow for confluent graphs to contain self loops or parallel edges, although we do allow for tracks to contain cycles and even multiple ways of realizing the same edge. Moreover, our definition implies that tracks in a confluent drawing have a "diode" property that does not allow one to double-back or make sharp turns after one has started going along a track in a certain direction.

Directed confluent drawings are defined similarly, except that in such drawings the locally-monotone curves are directed and the tracks formed by unions of curves must be oriented consistently.

3 Heuristic Algorithms

Though the planarity of a graph can be tested in linear time, it appears difficult to quickly determine whether or not a graph can be drawn confluently. If a graph G contains a non-planar subgraph, then G itself is non-planar too. But similar closure properties are not true for confluent graphs. Adding vertices and edges to a non-confluent graph increases the chances of edges crossing each other, but it also increases the chances of edges merging. Currently, the best method we know of for determining conclusively in the worst case whether a graph is confluent or not is a brute force one of exhaustively listing all possible ways of edge merging and checking the merged graphs for planarity. Therefore, it is of interest to develop heuristics that can find confluent drawings in many cases.

Fig. 3 shows confluent drawings using a "traffic circle" structure for complete subgraphs (cliques) and complete bipartite subgraphs (bicliques). At a high level, our heuristic drawing algorithm iteratively finds clique subgraphs and biclique subgraphs and replaces them with traffic-circle subdrawings.

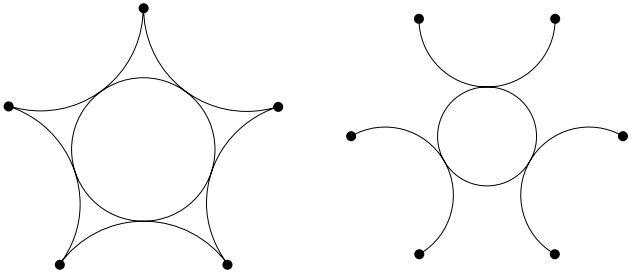

Fig. 3. Confluent drawings of K_5 and $K_{3,3}$ using "traffic circle" structures.

Chiba and Nishizeki [7] discuss the problem of listing complete subgraphs for graphs of bounded arboricity. The *arboricity* $a(G)$ is the minimum number of forests into which the edges of G can be partitioned. The listing algorithm is applicable for such graphs. Chiba and Nishizeki show that there can be at most $O(n)$ cliques of a given size in such graphs and give a linear time algorithm for listing these clique subgraphs. Eppstein [14] gives a linear time algorithm for listing maximal complete bipartite subgraphs in graphs of bounded arboricity.

The total complexity of all such graphs is $O(n)$, and again they can be listed in linear time.

In our heuristic algorithm for undirected graphs, we will use the clique subgraphs listing and the biclique subgraphs listing algorithms as our subroutines.

HeuristicDrawUndirected(G)
Input. A undirected sparse graph G.
Output. Confluent drawing of G if succeed, fail otherwise.
1. If G is planar
2. draw G
3. else if G contains a large clique or biclique subgraph C
4. create a new vertex v
5. obtain a new graph G' by removing edges of C and connecting each vertex of C to v
6. HeuristicDrawUndirected(G')
7. replace v by a small "traffic circle" to get a confluent drawing of G
8. else fail

Fig. 4.

In step 3 of the algorithm in Fig. 4, the cliques are given higher priority over bicliques, otherwise a clique would be partially covered by a biclique. Cliques of three or fewer vertices, and bicliques with one side consisting of only one vertex, are not replaced because the replacement cannot change the planarity of the graph. We now discuss the time performance of this heuristic.

Theorem 1. *In graphs of bounded arboricity, algorithm* HeuristicDrawUndirected *can be made to run in time $O(n)$, assuming hash tables with constant time per operation.*

Proof. We store a bit per edge of the original graph so we can quickly look up whether it is still part of our replacement. We begin the heuristic by looking for cliques, since we want to give them priority over bicliques. List all the complete subgraphs in the graph with four or more vertices, and sort them by size (the size of the complete subgraph is bounded too in graphs with bounded arboricity). Then, for each complete subgraph X in sorted order, we check whether X is still a clique of the modified graph, and if so perform a replacement of X. It is not hard to see that the new vertex v of the replacement cannot belong to any clique, so this algorithm correctly finds a maximal sequence of cliques to replace.

Next, we need to similarly dynamize the search for bicliques. This is more difficult, because a biclique may have nonconstant size and because the replacement vertex v may belong to additional bicliques. We perform this step by dynamizing the algorithm of Eppstein [14] for listing all bicliques. This algorithm uses the idea of a *d-bounded acyclic orientation*: that is, an orientation of the edges of the graph, such that the oriented graph is acyclic and the vertices have maximum outdegree d. For graphs of arboricity a, a $(2a-1)$-bounded acyclic orientation may easily be found in linear time. For such an orientation, define a *tuple* to be

a subset of the outgoing neighbors of any vertex, and let v be a *tuple creator* of tuple T if all vertices of T are outgoing neighbors of v. For graphs of bounded arboricity, there are at most linearly many distinct tuples. For each maximal biclique, one of the two sides of the bipartition must be a tuple, T [14]. The other side consists of two types of vertices: tuple creators of T, and outgoing neighbors of vertices of T.

Our algorithm stores a hash table indexed by the set of all tuples in the modified graph. The hash table entry for tuple T stores the number of tuple creators of T, and a list of outgoing neighbors of vertices of T that are adjacent to all tuple members. For each edge (u, v) in the graph, oriented from u to v, we store a list of the tuples T containing v for which u is listed as an outgoing neighbor. We also store a priority queue of the maximal bicliques generated by each tuple, prioritized by size; it will suffice for our purposes if the time to find the largest biclique is proportional to the biclique size, and it is easy to implement a priority queue with such a time bound. With these structures, we may easily look up each successive biclique replacement to perform in algorithm HeuristicDrawUndirected. Each replacement takes time proportional to the number of edges removed from the graph, so the total time for performing replacements is linear.

It remains to show how to update these data structures when we perform a biclique replacement. To update the acyclic orientation, orient each edge from C to v, except for those edges from vertices of C that have no outgoing edges in C. It can be seen that this orientation preserves d-boundedness and acyclicity. When a new vertex v is created by a replacement, create the appropriate hash table entries for tuples containing v; the number of tuples created by a replacement is proportional to the number of edges removed in the same replacement, so the total number of tuples created over the course of the algorithm is linear. Whenever a replacement causes edges from a vertex x to change, update the hash entries for all tuples for which x is a creator; this step takes $O(1)$ time per change. Also, update the hash entries for all tuples to which x belongs, to remove vertices that are no longer outgoing neighbors of x; this step takes time $O(1)$ per changed tuple, and each tuple changes $O(1)$ times over the course of the algorithm. Whenever a change removes incoming edges of x, we must remove the other endpoints of those edges from the lists of outgoing neighbors of tuples to which x belongs; using the lists associated with each incoming edge, this takes constant time per removal. Therefore, all steps can be performed in linear total time. □

An example of the input for algorithm HeuristicDrawUndirected and the output drawing produced by this heuristic is shown in Fig. 5.

4 Some Confluent Graphs

The heuristic algorithms presented in the previous section are most applicable to sparse graphs, because sparseness is needed for the linear time bound of the

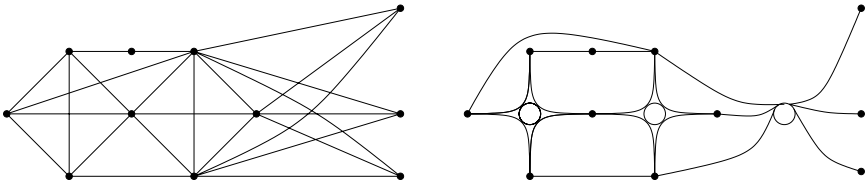

Fig. 5. An example of running the undirected heuristic algorithm. The input graph is shown in (a) and the output drawing is shown in (b).

maximal bipartite subgraph listing subroutine. However, there are also several denser classes of graphs that we can show to be confluent.

Interval graphs. An *interval graph* is formed by a set of closed intervals $S = \{[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]\}$. The interval graph is defined to have the intervals in S as its vertices and two vertices $[a_i, b_i]$ and $[a_j, b_j]$ are connected by an edge if and only if these two intervals have a non-empty intersection. Such graphs are typically non-planar, but we can draw them in a planar way using a confluent drawing[1].

Theorem 2. *Every interval graph is confluent.*

Proof. The proof is by construction. We number the interval endpoints by rank, $X = \{0, 1, \ldots, n-1\}$, and place these endpoints along the x-axis. We then build a two-dimensional lattice on top of these points in a fashion similar to Pascal's triangle, using a connector similar to an upside-down "V". These connectors stack on top of one another so that the apex of each is associated with a unique interval on X. We place each point from our set S of intervals just under its corresponding apex and connect it into the (single) track so that it can reach everything directly dominated by this apex in the lattice. At the bottom level, we connect the updside-down V's with rounded connectors. By this construction, we create a single track that allows each pair of vertices connected in the interval graph to have a locally-monotone path connecting them. (See Fig. 6.) □

Complements of trees. The complements of trees (graphs formed by connecting all pairs of vertices that are not connected in some tree) are also called *cotrees*. In general, cotrees are highly non-planar and dense, since a cotree with n vertices has $n(n-1)/2 - n + 1$ edges. Nevertheless, we have the following interesting fact.

Theorem 3. *The complement of a tree is confluent.*

[1] A similar construction works for circular-arc graphs and is left as an exercise for the interested reader.

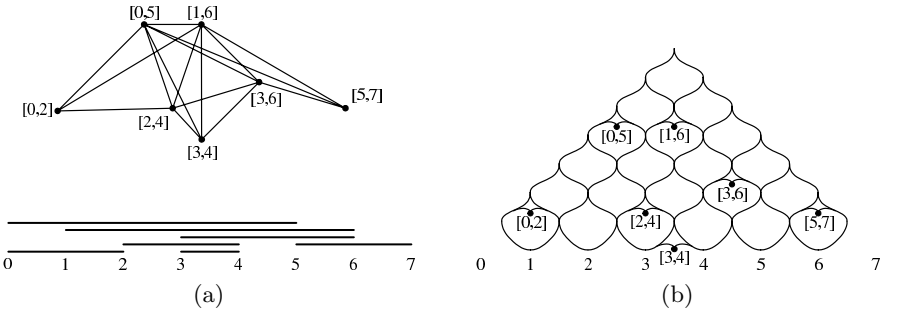

Fig. 6. Illustrating a confluent way to draw a non-planar interval graph: (a) an interval graph and its defining intervals; (b) its corresponding confluent drawing.

Proof. We prove the claim by recursive construction, using a single track for the entire graph. Assign a bounding rectangle for the tree and a bounding rectangle for every subtree in that tree. Place the complement of the tree into the bounding rectangles such that nodes of every subtree is within its bounding rectangle and the bounding rectangles of subtrees are contained in their parent's bounding rectangle. In addition, place a connector at the Northeastern corner of every bounding box. This connector is an imaginary point at which the single track for the entire graph will connect into this portion of the cotree. (See Fig. 7.) Connect the root node in each subtree to every connector of its children. Connect every node to the connector of its parent. Also connect every node to its siblings and the connectors of its siblings, as shown in the figure. The obtained drawing is the confluent drawing of the complement of the given tree. □

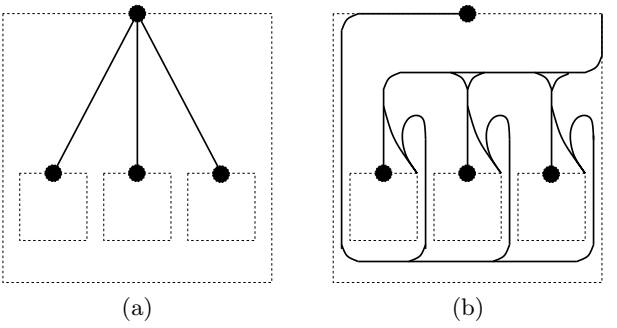

Fig. 7. Illustrating a confluent way to draw the complement of a tree: (a) a node and its children in the tree; (b) the corresponding portion of a track in the confluent drawing of the complement.

Paths are very special cases of trees. Every vertex in a path has a degree of 2 except its two endpoints, each of which has a degree of 1. The complement of a

path can be drawn using the cotree method in the above proof. We show a nice confluent drawing of the complement of a path in Fig. 8.

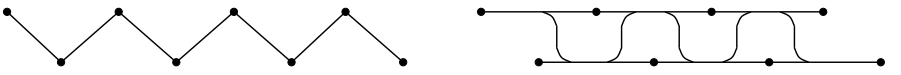

Fig. 8. P_8 and the confluent drawing of $\overline{P_8}$.

Complements of n-cycles. An n-cycle is a cycle with n vertices.

Theorem 4. *The complement of an n-cycle is confluent.*

Proof. First remove one vertex from the n-cycle and draw the confluent graph for the complement of the obtained path. Then add the vertex back and connect it with all vertices in the path except for its two neighbors. The obtained drawing is a confluent drawing (See Fig. 9 for an example.) □

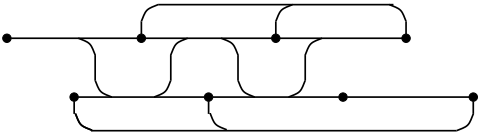

Fig. 9. A confluent drawing of $\overline{C_8}$.

Cographs. A *complement reducible graph* (also called a *cograph*) is defined recursively as follows [10]:

- A graph on a single vertex is a cograph.
- If G_1, G_2, $\cdots$, G_k are cographs, then so is their union $G_1 \cup G_2 \cup \cdots \cup G_k$.
- If G is a cograph, then so is its complement $\overline{G}$.

Cographs can be obtained from single node graphs by performing a finite number of unions and complementations.

Theorem 5. *Cographs are confluent.*

Proof. If cographs A and B are confluent, we can show $A \cup B$ and $\overline{A \cup B}$ are confluent too. First we draw A confluently inside a disk and attach a "tail" to the boundary of the disk. Connect the attachment point to each vertex in the disk. B is drawn in the same way. Then $A \cup B$ is formed by joining the two "tails" together so that they don't connect to each other. $\overline{A \cup B}$ is formed by joining the two "tails" of $\overline{A}$ and $\overline{B}$ together so that they connect to each other. (See Fig. 10.) By the definition of cographs and induction we know cographs are confluent. □

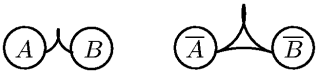

Fig. 10. Confluent $A \cup B$ and $\overline{A \cup B}$.

5 Some Non-confluent Graphs

In this section, we show that some graphs cannot be drawn confluently. These graphs include the Petersen graph P, the graph $P - v$ formed by removing one vertex from the Petersen graph, and the 4-dimensional hypercube.

The Petersen graph. By removing one vertex and its incident edges from the Petersen graph we obtain a graph homeomorphic to $K_{3,3}$. It contains no $K_{2,2}$ as a subgraph. Moreover, note that $K_{2,2}$ is the most basic structure that allows for edge merging into tracks. Thus the resulting graph is non-confluent. This graphis the smallest non-confluent graph we know of.

The Petersen graph itself is also non-confluent, as adding the vertex and edges back to its non-confluent subgraph doesn't create any four-cycles that could be used for confluent tracks.

4-dimensional hypercube. The 4-dimensional hypercube in Fig. 11 is non-confluent.

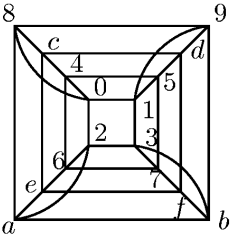

Fig. 11. The 4-dimensional hypercube.

The hypercube contains many subgraphs isomorphic to 3-dimensional cubes. Cubes are planar graphs, but in order to show non-confluence for the hypercube, we analyze more carefully the possible drawings of the cubes. Observe that, because there are no $K_{2,3}$ subgraphs in cubes or hypercubes, the only possible confluent tracks are $K_{2,2}$'s formed from the vertices of a single cube face.

Proof. (We omit the proof in this extended abstract.)

Acknowledgments. We would like to thank Jie Ren and André van der Hoek for supplying us with several examples of graphs used in software visualization.

References

1. T. Ball and S. G. Eick. Software visualization in the large. *IEEE Computer*, 29(4):33–43, 1996.
2. C. Batini, E. Nardelli, and R. Tamassia. A layout algorithm for data flow diagrams. *IEEE Trans. Softw. Eng.*, SE-12(4):538–546, 1986.
3. J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Macmillan, London, 1976.
4. R. Breu, U. Hinkel, C. Hofmann, C. Klein, B. Paech, B. Rumpe, and V. Thurner. Towards a formalization of the unified modeling language. In *Proc. 11th European Conf. Object-Oriented Programming (ECOOP'97)*, volume 1241 of *Lecture Notes in Computer Science*, pages 344–366, 1997.
5. C. Chambers, J. Dean, and D. Grove. A framework for selective recompilation in the presence of complex intermodule dependencies. In *Proceedings: 17th International Conference on Software Engineering*, pages 221–230, 1995.
6. C. C. Cheng, C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Drawing planar graphs with circular arcs. In *Proc. Graph Drawing*, volume 1731 of *Lecture Notes in Computer Science*, pages 117–126, 1999.
7. N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14:210–223, 1985.
8. N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *J. Comput. Syst. Sci.*, 30(1):54–76, 1985.
9. M. Chrobak and T. H. Payne. A linear time algorithm for drawing a planar graph on a grid. Technical Report UCR-CS-90-2, Dept. of Math. and Comput. Sci., Univ. California Riverside, 1990.
10. D. G. Corneil, H. Lerchs, and L. S. Burlingham. Complement reducible graphs. *Discrete Appl. Math.*, 3:163–174, 1981.
11. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.
12. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.
13. P. Eades and P. Mutzel. Graph drawing algorithms. In M. Atallah, editor, *CRC Handbook of Algorithms and Theory of Computation*, chapter 9. CRC Press, 1999.
14. D. Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information Processing Letters*, 51(4):207–211, August 1994.
15. T. Feder, A. Meyerson, R. Motwani, L. O'Callaghan, and R. Panigrahy. Representing graph metrics with fewest edges. In *Proceedings: 20th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2607 of *Lecture Notes Comput. Sci.*, pages 355–366, 2003.
16. M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM J. Algebraic Discrete Methods*, 4(3):312–316, 1983.
17. D. Grove, G. DeFouw, J. Dean, and C. Chambers. Call graph construction in object-oriented languages. In *Proc. of ACM Symp. on Object-Oriented Prog. Sys., Lang. and Applications (OOPSLA)*, pages 108–124, 1997.
18. P. Haynes, T. J. Menzies, and R. F. Cohen. Visualisations of large object-oriented systems. In P. D. Eades and K. Zhang, editors, *Software Visualisation*, volume 7, pages 205–218. World Scientific, Singapore, 1996.
19. M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing minimization problem. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, number 1353 in Lecture Notes Comput. Sci., pages 13–24. Springer-Verlag, 1997.

20. M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Algorithms Appl.*, 1(1):1–25, 1997.
21. G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996. (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia).
22. M. Kaufmann and D. Wagner. *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.
23. D. E. Knuth. Computer drawn flowcharts. *Commun. ACM*, 6, 1963.
24. X. Lin. On the computational complexity of edge concentration. *DAMATH: Discrete Applied Mathematics and Combinatorial Operational Research*, 101, 2000.
25. P. Mutzel. An alternative method to crossing minimization on hierarchical graphs. In S. North, editor, *Graph Drawing (Proc. GD '96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 318–333. Springer-Verlag, 1997.
26. P. Mutzel and T. Zeigler. The constrained crossing minimization problem. In J. Kratochvil, editor, *Graph Drawing Conference*, volume 1731 of *Lecture Notes in Computer Science*, pages 175–185. Springer-Verlag, 1999.
27. F. J. Newbery. Edge concentration: A method for clustering directed graphs. In *Proc. 2nd Internat. Workshop on Software Configuration Management*, pages 76–85, 1989.
28. T. Nishizeki and N. Chiba. *Planar Graphs: Theory and Algorithms*, volume 32 of *Ann. Discrete Math.* North-Holland, Amsterdam, The Netherlands, 1988.
29. R. C. Penner and J. L. Harer. *Combinatorics of Train Tracks*, volume 125 of *Annals of Mathematics Studies*. Princeton Univ. Press, Princeton, NJ, 1992.
30. B. Price, R. Baecker, and I. Small. A principled taxonomy of software visualization. *Journal of Visual Languages and Computing*, 4(3):211–266, 1993.
31. H. Purchase. Which aesthetic has the greatest effect on human understanding? In G. Di Battista, editor, *Proc. 5th Int. Symp. Graph Drawing, GD*, number 1353 in Lecture Notes in Computer Science, LNCS, pages 248–261. Springer-Verlag, 18–20 Sept. 1997.
32. G.-C. Roman and K. C. Cox. A taxonomy of program visualization systems. *IEEE Computer*, 26(12):11–24, 1993.
33. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.
34. W. Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Sympos. Discrete Algorithms*, pages 138–148, 1990.
35. J. T. Stasko and E. Kraemer. A methodology for building application-specific visualizations of parallel programs. *Journal of Parallel and Distributed Computing*, 18(2):258–264, 1993.
36. R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. Circuits Syst.*, CAS-36(9):1230–1234, 1989.
37. F. Tip and J. Palsberg. Scalable propagation-based call graph construction algorithms. *ACM SIGPLAN Notices*, 35(10):281–293, 2000.