Confluent Drawing Algorithms Using Rectangular Dualization[★]

Gianluca Quercini¹ and Massimo Ancona²

¹ Institute for Advanced Computer Studies,
University of Maryland College Park, MD, USA
`quercini@umiacs.umd.edu`

² Dipartimento di Informatica e Scienze dell'Informazione,
Università di Genova Genova, Italy
`ancona@disi.unige.it`

Abstract. The need of effective drawings for non-planar dense graphs is motivated by the wealth of applications in which they occur, including social network analysis, security visualization and web clustering engines, just to name a few. One common issue graph drawings are affected by is the visual clutter due to the high number of (possibly intersecting) edges to display. Confluent drawings address this problem by bundling groups of edges sharing the same path, resulting in a representation with less edges and no edge intersections. In this paper we describe how to create a confluent drawing of a graph from its rectangular dual and we show two important advantages of this approach.

Keywords: Confluent Drawing, Rectangular Dualization, Orthogonal Drawing, Clustered Graphs.

1 Introduction

Drawing graphs is a challenging problem, as witnessed by the wealth of approaches (recently surveyed in [6]) that have been proposed over the last two decades. Far from being mere and meaningless collections of nodes and edges, graphs are effective ways to describe relationships between objects (e.g. people, molecules, computers); consequently, visualizing a graph is an important step to clearly reveal all its information. In order to be readable, a drawing must comply with precise aesthetic criteria on the way nodes and edges are visualized; nodes (and edges) should not be too close to one another, symmetries should be highlighted, the total area should be minimized and so on. While optimizing all such constraints is clearly impossible, a good balance is usually the right way to go to create nice drawings. Large and dense graphs are inherently difficult to draw, as the amount of information that can be visualized is limited by the size of the medium (computer screen or paper) used to render the drawing. One

[★] This work was supported in part by the National Science Foundation under Grants IIS-10-18475, IIS-09-48548, IIS-08-12377, CCF-08-30618, and IIS-07-13501.

common issue is the visual clutter due to the high number of (possibly intersecting) edges, which prevents the human eye from easily following their paths. Recently a new drawing style called *confluent drawing* has been introduced to draw non-planar graphs with no edge crossings by bundling groups of intersecting edges into *tracks* [4]. Since not all graphs are *confluent* (e.g. can be drawn confluent) and no polynomial exact algorithm is known to decide whether a graph is confluent or not, most of the research in this area focuses on investigating new heuristics to recognize confluent graphs as well as studying new classes of confluent graphs [4, 7, 8, 14, 15]. However, it would be interesting to investigate how edges can be bundled not only to eliminate crossings but also to visualize as few segments/curves as possible, thus reducing the visual clutter in both non-planar and planar dense graphs.

Rather than improving the existing heuristics to recognize confluent graphs, our paper proposes an algorithm that is specifically designed for creating confluent drawings, bundling the edges so as to significantly reduce the amount of drawing elements to display. The algorithm creates a confluent drawing of a graph from its rectangular dual, a graph representation mostly used in VLSI floorplanning [13, 16–18]. We claim two major contributions of this approach:

- (a) We create orthogonal-like confluent drawings with large angular resolution ($\geq \pi/2$) of graphs with unbounded maximum degree, such that nodes are visualized as points in the plane and edges as sequences of vertical and horizontal segments connected through *curved bends* (Fig. 4);
- (b) We outline how to draw confluent a clustered graph with rectangular clusters, by extending the definition of rectangular dual to clustered graphs [19].

As far as (a) is concerned, in the case of planar graphs we may think of our drawings as a generalization of orthogonal drawings to graphs with maximum degree ≥ 4 , even though they are not perfectly orthogonal, due to the curved bends. We are fully aware that approaches exist that create perfectly orthogonal drawings for graphs with unbounded maximum degree. One of the best approaches is the Kandinsky model [11], where the nodes are represented as squares of equal size placed on a grid and edges as sequences of vertical and horizontal segments. However, if the size of the squares is too small the edges incident with a high-degree node may appear too close to one another which we avoid by keeping a large angular resolution. Tree-like confluent graphs [15] and delta-confluent graphs [7] can also be drawn with angular resolution $\geq \pi/2$, but they are limited to chordal bipartite graphs and distance hereditary graphs respectively, while our drawings can represent every confluent graph ¹.

The remainder of the paper is organized as follows. The notation and definitions used in the paper and relevant related work are introduced in Section 2. Theory of rectangular dualization is detailed in Section 3. In Section 4 we describe our drawing algorithms and we show preliminary results. In Section 5 we outline our method to create confluent drawings of clustered graphs. Section 6 concludes the presentation.

¹ Henceforth when we say “every confluent graph” we mean every graph that can be recognized as confluent by the existing heuristics.

2 Background and Related Work

In a *confluent drawing* edges are merged together into *tracks* which are the union of locally-monotone curves [4]. A *locally-monotone curve* is one that does not self intersect and contains no point with left and right tangents forming an angle ≤ 90 degrees. More precisely, in a confluent drawing D of an undirected graph $G = (V, E)$ each node $v \in V$ is drawn as a point v' in the plane, every edge $(v_i, v_j) \in E$ is represented as a locally-monotone curve connecting v'_i and v'_j and no two locally-monotone curves can cross.

A *clustered graph* C is a graph with a recursive partitioning of its node set $V(G)$ into groups called *clusters*. More formally $C = (G, T)$ consists of an *underlying graph* G and a tree T , describing the inclusion relation between clusters, such that every node ν of T is a cluster $V(\nu)$ of the nodes of G that are leaves of the subtree rooted at ν [9, 10]. *High-level clusters* are those included in no other clusters. Typically each cluster $V(\nu)$ is visualized as a region R in the plane enclosing the drawing of $G(\nu)$ (and the drawing of all subclusters). When the drawing of an edge e crosses the boundary of region R more than once and both endpoints of e are outside R we have a *edge-region crossing*. A drawing with no edge crossings nor edge-region crossings is said *c-planar* and C is *c-planar* if it admits such a drawing. Finally, C is *c-connected* if $G(\nu)$ is connected, for each $\nu \in T$ [9, 10]. Two important algorithms to draw clustered graphs using rectangular regions are described in [1, 5].

3 Rectangular Dualization

Let $G = (V, E)$ be a plane connected graph with n nodes and m edges. A *rectangular dual* $RD(G) = (F, f)$ of G (Fig. 1) is a *rectangular subdivision system* F – a partition of a rectangle into a set of non-overlapping rectangles no four of which meet at the same point – with a one-to-one correspondence $f : V \rightarrow F$ such that two nodes u and v are adjacent in G if and only if their corresponding rectangles $f(u)$ and $f(v)$ share a common boundary [16]. Not every planar graph admits a rectangular dual and, if so, it has many. Different groups of authors independently discovered and proved necessary and sufficient conditions for a graph to have a rectangular dual [16–18]. We prefer the formulation in [16] as we use the resulting algorithm to create rectangular duals.

Theorem 1. *A plane graph G has a rectangular dual with four rectangles on the boundary if and only if (a) every inner face of G is a 3-cycle and there are exactly four nodes on the outer cycle; (b) G has no separating triangles – 3-cycles containing at least one node in its interior.*

A graph which complies with Theorem 1 is called a *Proper Triangular Planar* (PTP) graph; we denote the four nodes on the outer cycle of G in counterclockwise order as v_N , v_W , v_S and v_E . By Theorem 1 only a limited subclass of planar graphs admits a rectangular dual; in order to support our statement that confluent drawings can be obtained from a rectangular dual of every confluent graph, we need to explain how we force a graph to comply with the admissibility conditions.

Fig. 1. A graph and its rectangular dual

3.1 Enforcing the Admissibility Conditions

First, Theorem 1 requires G to be planar; if it is not the case, we use the planarization procedure described in [4]. For every clique or biclique C of G , the edges of C are removed and the nodes of C are connected to a new node v , which we call a *clique crossover node* if C is a clique and *biclique crossover node* if C is a biclique; we say that the nodes of C are adjacent *through* the crossover node v . If the resulting graph is not planar, the algorithm concludes that G is not confluent and terminates. As pointed out in [4], this planarization procedure runs in $O(n)$ time assuming that G has bounded *arboricity*, which is the minimum number of forests into which the edges of G can be partitioned.

In order to turn G into a PTP graph G^* , we use a procedure described in [18] which goes through the following steps:

- (a) Edges are added to make G biconnected.
- (b) The four nodes v_N , v_W , v_S , v_E are added so that they form a new outer cycle of G . New edges are introduced to link the new nodes to those of the former outer cycle of G .
- (c) Separating triangles are searched [3] and *broken* (see below).
- (d) All inner faces are triangulated using the $O(n)$ algorithm described in [2].

As for step (c), the algorithm described in [3] finds all 3-cycles of G , including not only the separating triangles but also the 3-faces (e.g. faces bounded by a 3-cycle) of G . Since every 3-cycle can be described as a triple of integer numbers, each integer identifying a node (or an edge), the list L of 3-cycles and the list F of 3-faces can be lexicographically sorted using radix sort and compared against each other in $O(n)$ time to remove from L any item of F . To break a separating triangle $\Delta = (v, w, z)$, one of its edges (say (v, w)) is replaced by two new edges (v, c) and (c, w) , c being a new node called *ST crossover node*, and we say that v and w are adjacent *through* c . In order to break in $O(n)$ time all separating triangles of G by adding as few crossover nodes as possible we use the heuristics described in [19].

Using the algorithm described in [16] the rectangular dual $RD(G^*)$ is computed while removing the rectangles corresponding to v_N , v_W , v_S and v_E . It stands to reason that a rectangular dual of G^* is not a rectangular dual of G . Since new edges are added, indeed, two rectangles $f(u)$ and $f(v)$ may be adjacent in $RD(G^*)$ even if the two corresponding nodes u and v are not adjacent in

G ; similarly, due to the introduction of a crossover node c , $f(u)$ and $f(v)$ may not be adjacent in $RD(G^*)$ while $(u, v) \in E$. In the first case, since there is no relationship between u and v in G we can simply ignore that $f(u)$ is adjacent to $f(v)$; in the second case, we can think of $f(c)$ as a *gate*, through which one can go from $f(u)$ to $f(v)$, so as they can be considered as adjacent. Therefore, although it is not theoretically sound, we will refer to $RD(G^*)$ as the rectangular dual of G and we will denote it as $RD(G)$. Notice that exactly two nodes are adjacent through a *ST gate* (corresponding to a ST crossover node), while more than two are typically adjacent through *clique* and *biclique* gates.

4 From Rectangular Dual to Confluent Drawing

In this section we describe two methods to create a confluent drawing D of a graph $G = (V, E)$ from a rectangular dual $RD(G) = (\Gamma, f)$ of G . Both use an orthogonal-like drawing style, where nodes are represented as points in the plane and edges as sequences of horizontal and vertical segments, except for the edges connecting nodes adjacent through gates which are bundled into *traffic circles* (Fig. 2) as done in [4]. We remark that, while in an orthogonal drawing a *bend* is a point at which an horizontal and vertical segment meet, in our confluent drawings a bend is represented as a small diagonal segment (Fig. 2) connecting an horizontal to a vertical segment (or the other way round). The reason of this choice is that in a confluent drawing a segment may be shared by two or more edges that may follow different paths, whose direction is indicated by the bends.

Before describing the two approaches, we need some more notation. Every rectangle R in $RD(G)$ is bounded by four lines $x = x_{west}(R)$, $x = x_{east}(R)$, $y = y_{north}(R)$, $y = y_{south}(R)$, such that $x_{west}(R) < x_{east}(R)$ and $y_{south}(R) < y_{north}(R)$. Consequently R has four sides: the *north* and *south* sides lie on lines $y = y_{north}(R)$ and $y = y_{south}(R)$ respectively and the *west* and *east* sides lie on lines $x = x_{west}(R)$ and $x = x_{east}(R)$ respectively. Two rectangles R and Q are adjacent if they share one side or part of it. If R shares (part of) its north side with Q , R is *south of* Q (Q is *north of* R) and we write *south*(R, Q) (*north*(Q, R)); if R shares (part of) its west side with Q , R is *east of* Q (Q is *west of* R) and we write *east*(R, Q) (*west*(Q, R)).

In the following, when no ambiguity arises, we will use the same notation to denote a node or an edge and its pictorial representation (a point and a sequence of segments respectively).

4.1 Baseline Approach

The first method we will be describing creates D using no explicit technique to reduce the number of bends nor to bundle as many edges as possible so as to reduce the visual clutter. For this reason we refer to it as the *baseline approach*. To create the nodes, a point u is drawn at the center of every non-gate rectangle $f(u)$. As we will see, this choice heavily impacts on the number of resulting bends. In order to keep the bends at a readable length, if the x-coordinates $x(u)$

and $x(v)$ of two points u and v are such that $|x(u) - x(v)| < \epsilon$, for a small constant ϵ , the position of u and v is modified so that they appear aligned along the same vertical line. The same goes for $y(u)$ and $y(v)$. Next, we draw a traffic circle inside each gate. While traffic circles are the only possible option for clique and biclique gates, as more than two nodes are adjacent through them, they are not so necessary for ST gates, which connect exactly two nodes. However, we found cases where edges running through gates had too many unpleasant bends, which a traffic circle helped to hide. Finally for each point u we create its incident edges as follows (Fig. 3(a)). Without losing generality, we assume that at least one non-gate rectangle $f(v)$ is such that $north(f(v), f(u))$ and $(u, v) \in E$ (recall that since G may have been turned into a PTP graph $f(u)$ and $f(v)$ may be adjacent even if $(u, v) \notin E$). The same reasoning can be replicated if $f(v)$ has a different position with respect to $f(u)$. If u and v are aligned along the same vertical line (e.g. $x(u) = x(v)$), edge (u, v) is simply a segment between them (it has no bends). If u and v are not aligned, we create a new point (called *confluence point*) $c_{north}(u)$ with coordinates $(x(u), y_{north}(f(u)) - \epsilon)$ and we connect u and $c_{north}(u)$ by a segment s . Assuming s oriented from u to $c_{north}(u)$, v may be either left or right of segment s ; in the first case a *boundary point* $c_{north}^l(u)$ is created on the north side of $f(u)$ with coordinates $(x(u) - \epsilon, y_{north}(f(u)))$ and connected to $c_{north}(u)$ by a segment; otherwise a boundary point $c_{north}^r(u)$ is created with coordinates $(x(u) + \epsilon, y_{north}(f(u)))$ and connected to $c_{north}(u)$. A segment connecting a confluence point to a boundary point is a bend. Finally, if v has been already processed (meaning that $c_{south}^l(v)$ or $c_{south}^r(v)$ already exist), a segment is drawn between $c_{north}^l(u)$ and $c_{south}^l(v)$ (if v is at the left of segment s) or $c_{north}^r(u)$ and $c_{south}^r(v)$ (if v is at the right of segment s). The resulting edge (u, v) will have exactly two bends. The same procedure is used to create

Fig. 2. Drawing produced by the baseline approach

Fig. 3. How the baseline and the improved approaches work

an edge e between u and any traffic circle, with the only difference that e needs an extra bend to connect to the traffic circle, which is necessary to understand the direction e has to take inside the circle.

Theorem 2. *The baseline approach creates a confluent drawing D of graph G in $O(n)$ time.*

Proof. The number of gates is $O(n)$ and to create the edges the adjacency list of every node is crossed at most once, hence the linearity of the method. As for confluency, we notice that smooth lines are created only between points corresponding to nodes that are adjacent in G . In any edge e incident with a point u , the segment incident with u always run along a line passing through u ; the other edges run along the boundaries of the rectangles and cross a boundary only to reach an endpoint. Given that, it is impossible that two sequences of segments cross.

4.2 Improved Approach

As it is clear from Figure 2, the baseline approach creates many unwanted bends (look for example at edges (n, o) (o, p) and (p, q)) and it does not effectively reduce the visual clutter. Motivated by these observations, we propose a second approach, which we refer to as the *improved approach* that goes through the same steps as the baseline approach (creates the nodes, the traffic circles and, for each point, the edges incident with it) but optimizes through heuristics the position of the nodes as well as the layout of the edges to reduce the number of bends and to bundle as many edges as possible (Fig. 4).

In order to make sure that we do not create edge crossings, we stick with our principle that the drawing of node u must be inside the corresponding rectangle $f(u)$ and that the drawing of edge (u, v) must cross the boundary of no rectangle but $f(u)$ and $f(v)$. Instead of blindly drawing point u at the center of $f(u)$, we place u within $f(u)$ so as to minimize the number of bends of its incident edges. Although this does not necessarily imply that we minimize the number of bends across the whole drawing, we found that eliminates many unnecessary bends, such as the ones mentioned before. As for the edges, we remark that in the drawings generated by the baseline approach any edge between two nodes not adjacent through a gate either has no bends or two. Before creating a two-bend edge, the improved approach checks whether it is possible to lay it down with only one. Let us assume again, without loss of generality, that $north(v, u)$.

Fig. 4. Drawing produced by the improved approach

There are two ways to draw edge (v, u) with one bend (Fig. 3(b)): using the sequence of black thick segments denoted as s_1 or the sequence of gray thick segments denoted as s_2 . s_2 is not a good solution, as it may cross sequences of segments incident with v (such as s_4); on the other hand, s_1 cannot cross any segment incident with v because the segment of s_1 incident with v runs on a line passing through v , as in the baseline approach, where crossings are not possible. However, s_1 may still cross a previously created 1-bend edge s_3 incident with u . Since both s_1 and s_3 represent valid 1-bend edges incident with u , we create the one which maximizes the number of 1-bend edges incident with u ; in other words, we create s_1 if the edges following the same path as s_1 are more than the ones following the same path as s_3 . Finally, all edges that cannot be created with only one bend are drawn with two-bends as described in the baseline approach. It is immediate to verify that the improved approach creates confluent drawings in linear time.

4.3 Implementation and Results

Both approaches described in the previous section have been implemented in *C* and included in our software OcORD, which has been originally created to study the properties of rectangular dualization and now has been extended to the graph drawing domain [19]. While introducing this paper, we stated that the goal of our drawing algorithm is to reduce the visual clutter due to the high number of edges in large graphs. To support this statement, we need a measure to assess such a reduction. One way to go here is to compare the number of drawing elements (e.g. segments) used to create the confluent drawing against the number of segments that would have been necessary to create the same drawing without bundling the

Fig. 5. Experimental results

edges. Therefore we selected 100 graphs from the *AT&T* data set², whose order ranges from 10 to 100 nodes, and for each of them we created, besides a confluent drawing using both the baseline and the improved approach, a Kandinsky-like drawing using the same drawing conventions for the edges as in the confluent drawings described in the previous section (e.g. edges represented as sequences of vertical and horizontal segments with diagonal bends). For all drawings we then measured the total number of bends and the number of segments used to draw the edges. Figure 5 reports on the average of these values over all data set. As expected, the improved approach outperforms the baseline approach both in terms of reduction of the segments and in terms of total number of bends. On average the drawings produced with the improved approach have up to 30% less segments than the drawings produced without bundling the edges, while in the drawings created by the baseline approach this reduction drops to 20%. Moreover, the improved approach halves the number of bends created with the baseline; however, looking at Fig. 4, we are still able to find unwanted bends, partly due to the traffic circles and partly due to the constraint that edges cannot cross the boundaries of other rectangles than the ones corresponding to their endpoints. As bends are also responsible for visual clutter, we are planning on substituting heuristics with an optimal algorithm which minimizes the bends through a flow based method, like it is done in [12]. We conclude the discussion of the results with two important remarks. First, we found the quality of the drawings is sometimes questionable. Referring to Fig. 4, nodes tend to crowd in some areas, while other regions appear empty; this is due to the different size of the rectangles in the rectangular dual, which is also responsible for the excessive length of some edges (such as edge (f, n)). Finally, a graph has typically many rectangular duals, which obviously lead to different drawings. It would be interesting to investigate how different rectangular duals impact on the quality of the drawing (especially in terms of number of gates and thus bends). Both aspects will be object of future research.

5 Confluent Rectangular Drawing for Clustered Graphs

In [19] the *clustered rectangular dual* (*c-rectangular dual* for short) has been introduced as a generalization of the rectangular dual for c-planar clustered

² <http://www.graphdrawing.org/>

Fig. 6. A clustered graph with its c-rectangular dual and confluent drawing (in gray)

graphs. Intuitively, a c-rectangular dual of $C = (G, T)$ is a rectangular dual of G such that the rectangles corresponding to the nodes in $V(\nu)$ form a rectangular dual of $G(\nu)$ (Fig. 6). More formally:

Definition 1. A c-rectangular dual $RD(C) = (\Gamma, f)$ of a c-planar clustered graph $C = (G, T)$ is a rectangular dual of G such that, for each node ν of T : (A) The union of the rectangles in $\Gamma(V(\nu))$ is a rectangular dual of $G(\nu)$, where $\Gamma(V(\nu)) = \{R \in \Gamma \mid f^{-1}(R) \in V(\nu)\}$; (B) If $V(\mu) \subset V(\nu)$, $\Gamma(V(\mu)) \subset \Gamma(V(\nu))$.

The outline of the admissibility conditions require additional theoretical background which is out of the scope of this paper; the interested reader is referred to [19]. Here we limit ourselves to a short description of the algorithm that creates a c-rectangular dual of every c-planar graph $C = (G, T)$. Without losing generality we assume that the underlying graph G is a PTP graph, whose outer nodes belong to no cluster (they are mere construction nodes). The idea of the algorithm is extremely simple; first we create the rectangular dual $RD(Q)$ of the quotient graph Q of C , where each rectangle corresponds to a high-level cluster $V(\nu)$ of C , and next each rectangle is “filled” with the rectangular dual of $G(\nu)$ (if $G(\nu)$ is a plain graph) or the c-rectangular dual of $G(\nu)$ (if $G(\nu)$ is in turn a clustered graph). We recall that Q is the graph having a node v_ν for each high-level cluster $V(\nu)$ of C and an edge between two nodes v_ν, v_μ if and only if there is an edge in C linking a node in $V(\nu)$ to a node in $V(\mu)$. Figure 7 (a) and (b) show the quotient graph of the clustered graph in Fig. 6 and the rectangular dual respectively. It stands to reason that G being a PTP does not imply that Q is a PTP graph; separating triangles and even multiple edges may need to be broken using additional crossover nodes (which may be thought of as empty clusters). Similarly, the rectangular dual of $G(\nu)$, for each $V(\nu)$, must be such that the rectangles on the boundary fulfill the adjacency constraints with the rectangles on the boundary of adjacent clusters. Referring to Fig. 6, the rectangular dual of $G(\{a, b, c, l\})$ can be rotated so that rectangle l appears in the position where b, c are now (and vice versa), and still we would have a valid rectangular dual of $G(\{a, b, c, l\})$; but rectangle l would not be adjacent to rectangle k anymore and, similarly, c would not be adjacent to d . Again, for details on this point we refer the interested reader to [19]. Once a c-rectangular dual has been created, one of the algorithms described in Section 4 is used to create a confluent drawing of the underlying graph. Figure 6 shows that the confluent drawing of $G(\nu)$, for

Fig. 7. (a) Quotient graph Q of C . (b) $RD(Q)$. (c) Confluent R-drawing of C .

every cluster, is completely contained in the interior of the rectangular region delimited by the rectangular dual of $G(\nu)$. All we are left to do is to separate the sides of these rectangular regions so as to obtain the *confluent rectangular drawing* (or confluent R-drawing) shown in Figure 7, where each cluster is represented as an independent rectangular region. The advantage of a confluent R-drawing is that it reduces the visual clutter due to the edges and the visual clutter due to the representation of the clusters; using simple convex regions such as rectangles, in fact, improves the readability of the graph, as pointed out in [5]. We have no experimental results on this part so far, as the algorithm that creates the c-rectangular dual is not perfect yet. In particular, it seems to create too many gates, which, as seen before, have such a negative impact on the number of bends. This will be the object of our future work.

6 Conclusions

In this paper we discussed two algorithms that create a confluent drawing of a graph G from its rectangular dual. Preliminary results indicate that one of the approaches successfully decreases the amount of segments needed to visualize the edges, thus reducing visual clutter. Much work is still left to do to reduce the number of bends as well as applying the described techniques to clustered graphs.

References

1. Angelini, P., Frati, F., Kaufmann, M.: Straight-Line Rectangular Drawings of Clustered Graphs. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 25–36. Springer, Heidelberg (2009)
2. Biedl, T.C., Kant, G., Kaufmann, M.: On Triangulating Planar Graphs under the Four-Connectivity Constraint. In: Schmidt, E.M., Skyum, S. (eds.) SWAT 1994. LNCS, vol. 824, pp. 83–94. Springer, Heidelberg (1994)
3. Chiba, N., Nishizeki, T.: Arboricity and Subgraph Listing. SIAM Journal on Computing 14(1), 210–223 (1985)

4. Dickerson, M., Eppstein, D., Goodrich, M.T., Meng, J.Y.: Confluent Drawings: Visualizing Non-planar Diagrams in a Planar Way. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 1–12. Springer, Heidelberg (2004)
5. Eades, P., Feng, Q.-W., Nagamochi, H.: Drawing Clustered Graphs on an Orthogonal Grid. *Journal on Graph Algorithms and Applications* 3(4), 3–29 (1999)
6. Eades, P., Gutwenger, C., Hong, S.-H., Mutzel, P.: Graph Drawing Algorithms. In: *Algorithms and Theory of Computation Handbook*, 2nd edn., vol. 2, CRC Press, Boca Raton (2009)
7. Eppstein, D., Goodrich, M.T., Meng, J.Y.: Delta-Confluent Drawings. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 165–176. Springer, Heidelberg (2006)
8. Eppstein, D., Goodrich, M.T., Meng, J.Y.: Confluent Layered Drawings. *Algorithmica* 47(4), 439–452 (2007)
9. Feng, Q.-W., Cohen, R.F., Eades, P.: Planarity for Clustered Graphs. In: Spirakis, P.G. (ed.) ESA 1995. LNCS, vol. 979, pp. 213–226. Springer, Heidelberg (1995)
10. Feng, Q.: Algorithms for Drawing Clustered Graphs. Ph.D. thesis, University of Newcastle, Australia (1997)
11. Fössmeier, U., Kaufmann, M.: Drawing High Degree Graphs with Low Bend Numbers. In: Brandenburg, F.J. (ed.) GD 1995. LNCS, vol. 1027, pp. 254–266. Springer, Heidelberg (1996)
12. Garg, A., Tamassia, R.: A New Minimum Cost Flow Algorithm with Applications to Graph Drawing. In: North, S.C. (ed.) GD 1996. LNCS, vol. 1190, pp. 201–216. Springer, Heidelberg (1997)
13. He, X.: On Floorplans of Planar Graphs. In: *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pp. 426–435. ACM, New York (1997)
14. Hirsch, M., Meijer, H., Rappaport, D.: Biclique Edge Cover Graphs and Confluent Drawings. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 405–416. Springer, Heidelberg (2007)
15. Hui, P., Pelsmayer, M.J., Schaefer, M., Stefankovic, D.: Train Tracks and Confluent Drawings. *Algorithmica* 47(4), 465–479 (2007)
16. Kant, G., He, X.: Two Algorithms for Finding Rectangular Duals of Planar Graphs. In: van Leeuwen, J. (ed.) WG 1993. LNCS, vol. 790, pp. 396–410. Springer, Heidelberg (1994)
17. Koźmiński, K., Kinnen, E.: Rectangular Duals of Planar Graphs. *Networks* 15(2), 145–157 (1985)
18. Lai, Y.-T., Leinwand, S.M.: Algorithms for Floorplan Design via Rectangular Dualization. *IEEE Transactions on Computer-Aided Design* 7(12), 1278–1289 (1988)
19. Quercini, G.: Optimizing and Visualizing Planar Graphs via Rectangular Dualization. Ph.D. thesis, University of Genoa, Genoa, Italy (2009)