Delta-Confluent Drawings*

David Eppstein, Michael T. Goodrich, and Jeremy Yu Meng

School of Information and Computer Science,
University of California, Irvine,
Irvine, CA 92697, USA
{eppstein, goodrich, ymeng}@ics.uci.edu

Abstract. We generalize the *tree-confluent* graphs to a broader class of graphs called *Δ-confluent* graphs. This class of graphs and distance-hereditary graphs, a well-known class of graphs, coincide. Some results about the visualization of Δ-confluent graphs are also given.

1 Introduction

Confluent Drawing is an approach to visualize non-planar graphs in a planar way [10]. The idea is simple: we allow groups of edges to be merged together and drawn as tracks (similar to train tracks). This method allows us to draw, in a crossing-free manner, graphs that would have many crossings in their normal drawings. Two examples are shown in Figure. 1. In a confluent drawing, two nodes are connected if and only if there is a smooth curve path from one to the other without making sharp turns or double backs, although multiple realizations of a graph edge in the drawing is allowed.

More formally, 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.

* Work by the first author is supported by NSF grant CCR-9912338. Work by the second and the third author is supported by NSF grants CCR-0098068, CCR-0225642, and DUE-0231467.

P. Healy and N.S. Nikolov (Eds.): GD 2005, LNCS 3843, pp. 165–176, 2005.
© Springer-Verlag Berlin Heidelberg 2005

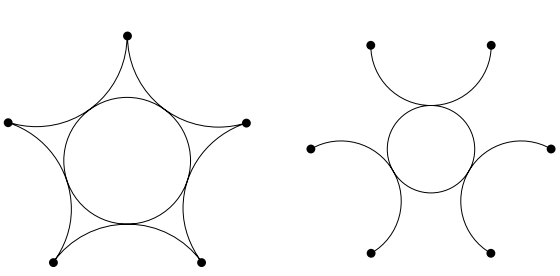

Fig. 1. Confluent drawings of K_5 and $K_{3,3}$

We assume readers have basic knowledge about graph theory and we will use conventional terms and notations of graph theory without defining them. All graphs considered in this paper are simple graphs, i.e., without loop or multi-edge. Confluent graphs are closely related to planar graphs. It is, however, very hard to check whether a given graph can be drawn confluently. The complexity of recognizing confluent graphs is still open and the problem is expected to be hard. Hui, Schaefer and Štefankovič [21] define the notion of *strong confluency* and show that strong confluency can be recognized in **NP**. It is then of interest to study classes of graphs that can or can not be drawn confluently. Several classes of confluent graphs, as well as several classes of non-confluent graphs, have been listed [10].

In this paper we continue in the positive direction of this route. We describe Δ-confluent graphs, a generalization of *tree-confluent* graphs [21]. We discuss problems of embedding trees with internal degree three, including embeddings on the hexagonal grid, which is related to Δ-confluent drawings with large angular resolution, and show that $O(n \log n)$ area is enough for a Δ-confluent drawing of a Δ-confluent graph with n vertices on the hexagonal grid.

Note that although the method of merging groups of edges is also used to reduce crossings in *confluent layered drawings* [14], edge crossings are allowed to exist in a confluent layered drawing.

2 Δ-Confluent Graphs

Hui, Schaefer and Štefankovič [21] introduce the idea of *tree-confluent* graphs. A graph is *tree-confluent* if and only if it is represented by a planar train track system which is topologically a tree. It is also shown in their paper that the class of tree-confluent graphs are equivalent to the class of chordal bipartite graphs.

The class of tree-confluent graphs can be extended into a wider class of graphs if we allow one more powerful type of junctions.

A *Δ-junction* is a structure where three paths are allowed to meet in a three-way complete junction. The connecting point is call a *port* of the junction. A *Λ-junction* is a broken Δ-junction where two of the three ports are disconnected from each other (exactly same as the *track* defined in the tree-confluent drawing [21]). The two disconnected paths are called *tails* of the Λ-junction and the remaining one is called *head*.

Fig. 2. Δ-junction and Λ-junction

A *Δ-confluent drawing* is a confluent drawing in which every junction in the drawing is either a Δ-junction, or a Λ-junction, and if we replce every junction in the drawing with a new vertex, we get a tree. A graph G is Δ-confluent if and only if it has a Δ-confluent drawing.

The class of cographs in [10] and the class of tree-confluent graphs in [21] are both included in the class of Δ-confluent graphs. We observe that the class of Δ-confluent graphs are equivalent to the class of distance-hereditary graphs.

2.1 Distance-Hereditary Graphs

A *distance-hereditary* graph is a connected graph in which every induced path is isometric. That is, the distance of any two vertices in an induced path equals their distance in the graph [2]. Other characterizations have been found for distance-hereditary graphs: forbidden subgraphs, properties of cycles, etc. Among them, the following one is most interesting to us:

Theorem 1. [2] *Let G be a finite graph with at least two vertices. Then G is distance-hereditary if and only if G is obtained from K_2 by a sequence of one-vertex extensions: attaching pendant vertices and splitting vertices.*

Here attaching a pendant vertex to x means adding a new vertex x' to G and making it adjacent to x so x' has degree one; and splitting x means adding a new vertex x' to G and making it adjacent to either x and all neighbors of x, or just all neighbors of x. Vertices x and x' forming a split pair are called *true twins* (or *strong siblings*) if they are adjacent, or *false twins* (or *weak siblings*) otherwise.

By reversing the above extension procedure, every finite distance-hereditary graph G can be reduced to K_2 in a sequence of one-vertex operations: either delete a pendant vertex or identify a pair of twins x' and x. Such a sequence is called an *elimination sequence* (or a *pruning sequence*).

In the example distance-hereditary graph G of Figure. 3, the vertices are labelled reversely according to an elimination sequence of G:

17 merged into 16, 16 merged into 15, 15 cut from 3, 14 cut from 2, 13 merged into 5, 12 merged into 6, 10 merged into 8, 11 merged into 7, 9 cut from 8, 8 merged into 7, 7 cut from 6, 6 merged into 0, 5 cut from 0, 4 merged into 1, 3 cut from 1, 2 merged into 1.

The following theorem states that the class of distance hereditary graphs and the class of Δ-confluent graphs are equivalent.

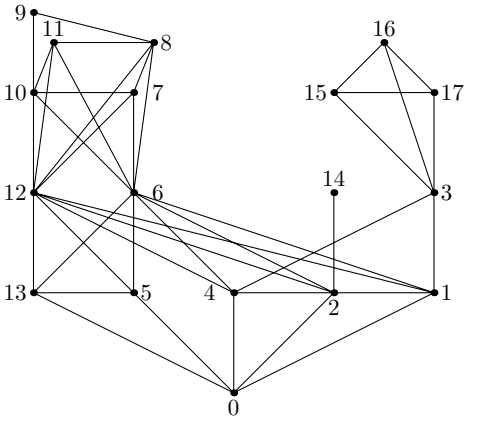

Fig. 3. A distance-hereditary graph G

Theorem 2. *A graph G is distance hereditary if and only if it is Δ-confluent.*

Proof sketch. Assume G is distance hereditary. We can compute the elimination sequence of G, then apply an algorithm, which will be described in Section 2.2, to get a Δ-confluent drawing of G. Thus G is Δ-confluent.

On the other hand, given a Δ-confluent graph G in form of its Δ-confluent drawing A, we can apply the following operations on the drawing A:

1. *contraction.* If two vertices y and y' in A are connected to two ports of a Δ-junction, or y and y' are connected to the two tails of a Λ-junction respectively, then contract y and y' into a new single vertex, and replace the junction with this new vertex.
2. *deletion.* If two vertices y and y' in A are connected by a Λ-junction, y is connected to the head and y' to one tail, remove y' and replace the junction with y.

It is easy to observe that contraction in the drawing A corresponds to identifying a pair of twins in G; and deletion corresponds to removing a pendant vertex in G.

It is always possible to apply an operation on two vertices connected by a junction because the underlying graph is a tree. During each operation one junction is replaced. Since the drawing is finite, the number of junctions is finite. Therefore, we will reach a point at which the last junction is replaced. After that the drawing reduces to a pair of vertices connected by an edge, and the corresponding G reduces to a K_2. Therefore G is a distance-hereditary graph.

This completes the proof of the equivalence between Δ-confluent graphs and distance-hereditary graphs. □

2.2 Elimination Sequence to Δ-Confluent Tree

The recognition problem of distance-hereditary graphs is solvable in linear time (see [2, 20]). The elimination sequence (ordering) can also be computed in linear

time. Using the method of, for example, Damiand et al. [9] we can obtain an elimination sequence L for G of Figure. 3:

By using the elimination sequence reversely, we construct a tree structure of the Δ-confluent drawing of G. This tree structure has n leaves and $n-1$ internal nodes. Every internal node has degree of three. The internal nodes represent our Δ- and Λ-junctions. The construction is as follows.

- While L is non-empty do:
  - Get the last *item* from L
  - If *item* is "b merged into a"
    - If edge $(a, b) \in E(G)$, then replace a with a Δ conjunction using any of its three connectors, connect a and b to the other two connectors of the Δ conjunction; otherwise replace a with a Λ conjunction using its head and connect a and b to its two tails.
  - Otherwise *item* is "b cut from a", replace a with a Λ conjunction using one of its tails, connect a to the head and b to the other tail left.

Clearly the structure we obtain is indeed a tree. Once the tree structure is constructed, the Δ-confluent drawing can be computed by visualizing this tree structure with its internal nodes replaced by Δ- and Λ-junctions.

3 Visualizing the Δ-Confluent Graphs

There are many methods to visualize the underlying topological tree of a Δ-confluent drawing. Algorithms for drawing trees have been studied extensively (see [4, 8, 12, 13, 18, 19, 22, 26, 27, 28, 29, 31, 32] for examples). Theoretically all the tree visualization methods can be used to lay out the underlying tree of a Δ-confluent drawing, although free tree drawing techniques might be more suitable. We choose the following two tree drawing approaches that both yield large angular resolution ($\geq \pi/2$), because in drawings with large angular resolution, each junction lies in a center-like position among the nodes connected to it, so junctions are easy to perceive and paths are easy to follow.

3.1 Orthogonal Straight-Line Δ-Confluent Drawings

The first tree drawing method is the orthogonal straight-line tree drawing method. In the drawings by this method, every edge is drawn as a straight-line segment and every node is drawn at a grid position.

Pick an arbitrary leaf node l of the underlying tree as root and make this free tree a rooted tree T (alternatively one can adopt the elimination hierarchy tree of a distance-hereditary graph for use here.) It is easy to see that T is a binary tree because every internal node of the underlying tree has degree three. We can then apply any known orthogonal straight-line drawing algorithm for trees ([e.g.[4, 6, 7, 24, 25, 30]]) on T to obtain a layout. After that, replace drawings of internal nodesz with their corresponding junction drawings.

3.2 Hexagonal Δ-Confluent Drawings

Since all the internal nodes of underlying trees of Δ-confluent graphs have degree three, if uniform-length edges and large angular resolution are desirable, it is then natural to consider the problem of embedding these trees on the hexagonal grid where each grid point has three neighboring grid points and every cell of the grid is a regular hexagon.

Some researchers have studied the problem of hexagonal grid drawing of graphs. Kant [23] presents a linear-time algorithm to draw tri-connected planar graphs of degree three planar on a $n/2 \times n/2$ hexagonal grid. Aziza and Biedl [1] focus on keeping the number of bends small. They give algorithms that achieve $3.5n + 3.5$ bends for all simple graphs, prove optimal lower bounds on number of bends for K_7, and provide asymptotic lower bounds for graph classes of various connectivity. We are not aware of any other result on hexagonal graph drawing, where the grid consists of regular hexagon cells.

In the Δ-confluent drawings on the hexagonal grid, any segment of an edge must lie on one side of a hexagon sub-cell. Thus the slope of any segment is $1/2$, ∞, or $-1/2$. An example drawing for the graph from Figure. 3 is shown in Figure. 4.

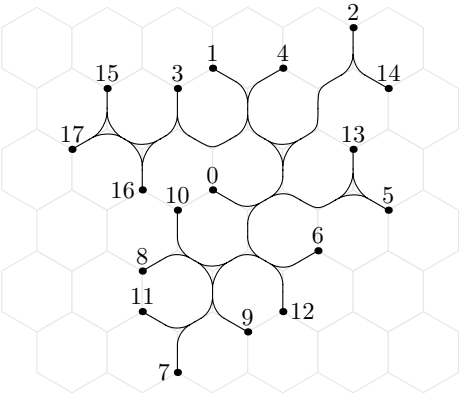

Fig. 4. A hexagonal grid Δ-confluent drawing example

Readers might notice that there are edge bends in the drawing of Figure. 4. Some trees may require a non-constant number of bends per edge to be embedded on a hexagonal grid. Thus it is impossible to embed the tree without edge crossing or edge overlapping, when the bends are limited per edge. However, if unlimited bends are allowed, we show next that Δ-confluent graphs can be embedded in the hexagonal grid of $O(n \log n)$ area in linear time.

The method is to transform an orthogonal straight-line tree embedding into an embedding on the hexagonal grid. We use the results of Chan et al. [6] to obtain an orthogonal straight-line tree drawing. In their paper, a simple "recursive winding" approach is presented for drawing arbitrary binary trees in small area with good aspect ratio. They consider both upward and non-upward cases of

orthogonal straight-line drawings. We show that an upward orthogonal straight-line drawing of any binary tree can be easily transformed into a drawing of the same tree on the hexagonal grid.

Figure. 5 (a) exhibits an upward orthogonal straight-line drawing for the underlying tree of G in Figure. 3, with node 15 being removed temporarily in order to get a binary tree.

We cover the segments of the hex cell sides with two set of curves: u-curves and v-curves (Figure. 5 (b)). The u-curves (solid) are waving horizontally and the v-curves (dashed) along one of the other two slopes. These two sets of curves are not direct mapping of the lines parallel to x-axis or y-axis in an orthogonal straight-line drawing settings, because the intersection between a u-curve and a v-curve is not a grid point, but a side of the grid cell and it contains two grid points. However this does not matter very much. We choose the lower one of the two grid points in the intersection (overlapping) as our primary point and the

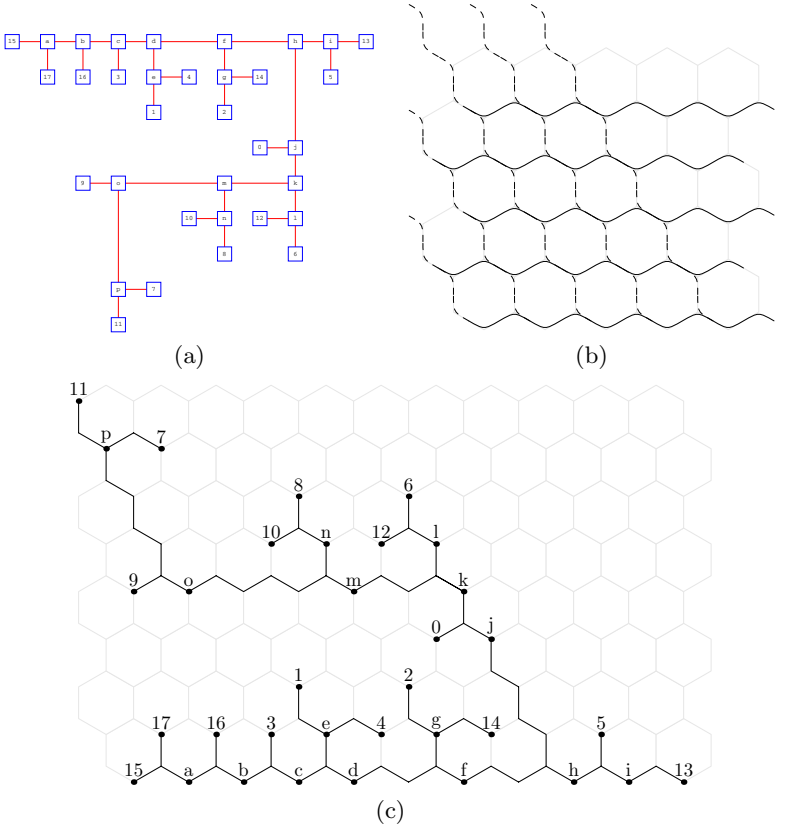

Fig. 5. From upward straight-line orthogonal drawing to hexagonal grid drawing. Internal nodes are labelled with letters and leaves with numbers. (a) orthogonal drawing, generated by Graph Drawing Server (GDS) [5]. (b) u-curves and v-curves. (c) unadjusted result of transformation (mirrored upside-down for a change).

other one as our backup point. So the primary point is at the bottom of a grid cell and its backup is above it to the left. As we can see later, the backup points allow us to do a final adjustment of the node positions.

When doing the transformation from an orthogonal straight-line drawing to a hexagonal grid drawing, we are using only the primary points. So there is a one-to-one mapping between node positions in the orthogonal drawing and the hexagonal grid drawing. However, there are edges overlapping each other in the resultant hexagonal grid drawing of such a direct transformation (e.g. edge (a, b) and edge $(a, 16)$ in Figure. 5 (c)). Now the backup points are used to remove those overlapping portion of edges. Just move a node from a primary point to the point's backup when overlapping happens.

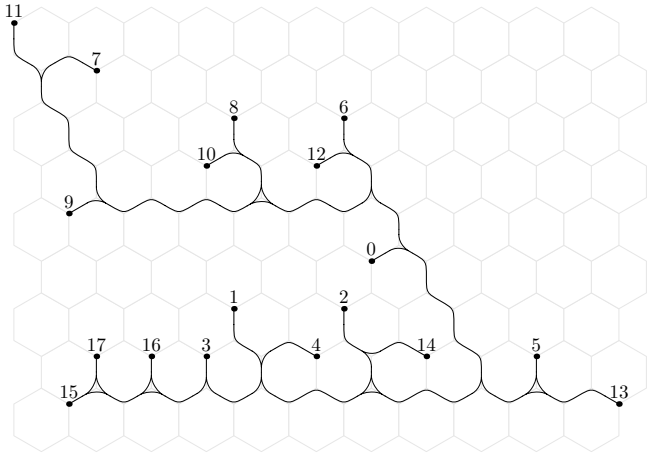

Fig. 6. Final drawing after adjustment

Figure. 6 shows the Δ-confluent drawing of G after overlapping is removed. The drawing does not look compact because the orthogonal drawing from which it is obtained is not tidy in order to have the subtree separation property.

It is not hard to see that backup points are enough for removing all the overlapping portions while the tree structure is still maintained. If wanted, the backup points can be also used to reduce the bends along the edges connecting the tree leaves (e.g. edge connecting node 1). Some bends can be removed as well after junctions are moved (e.g. the subtree of node 8 and 10).

Theorem 3. *Any Δ-confluent graph can be embedded on a grid of size $O(n \log n)$. The representation of its Δ-confluent drawing can be computed in linear time and can be stored using linear space.*

Proof sketch. First the underlying tree of a Δ-confluent graph can be computed in linear time. The transformation runs in linear time as well. It then remains to show that the orthogonal tree drawing can be obtained in linear time. Chan et al.

[6] can realize a upward orthogonal straight-line grid drawing of an arbitrary n-node binary tree T with $O(n \log n)$ area and $O(1)$ aspect ratio. The drawing achieves subtree separation and can be produced in $O(n)$ time.

By using the transformation, we can build a description of the drawing in linear time, which includes the placement of each vertex and representation of each edge. It is straightforward that the drawing has an area of $O(n \log n)$ size. Since the edges are either along u-curves, or along v-curves, we just need to store the two end points for each edge. Note that although some edge might contain $O(\sqrt{n \log n})$ bends (from the "recursive winding" method), constant amount of space is enough to describe each edge. Thus the total space complexity of the representation is $O(n)$. □

In the hexagonal grid drawings for trees, the subtree separation property is retained if the subtree separation in hexagonal grid drawings is defined using u, v area. If different methods of visualizing binary trees on the orthogonal grid are used, various time complexities, area requirements, and other drawing properties for the hexagonal grid Δ-confluent drawing can be derived as well.

4 More About Δ-Confluent Graphs

In this section we discuss a Δ-confluent subgraph problem, and list some topics of possible future work about Δ-confluent graphs.

One way to visualize a non-planar graph is to find a maximum planar subgraph of the original graph, compute a planar drawing of the subgraph, and add the rest of the original graph back on the drawing. An analogous method to visualize a non-Δ-confluent graph would be to find a maximum Δ-confluent subgraph, compute a Δ-confluent drawing, and add the rest back. However, just like the maximum planar subgraph problem, the maximum Δ-confluent subgraph problem is difficult. The problem is defined below, and its complexity is given in Theorem 4.

Maximum Δ-confluent Subgraph Problem:
Instance: A graph $G = (V, E)$, an integer $K \leq |V|$.
Question: Is there a $V' \subset V$ with $|V'| \geq K$ such that the subgraph of G induced by V' is a Δ-confluent?

Theorem 4. *Maximum Δ-confluent subgraph problem is NP-complete.*

Proof. The proof can be derived easily from Garey and Johnson [17, GT21].

[GT21] Induced Subgraph with Property Π:
Instance: A graph $G = (V, E)$, an integer $K \leq |V|$.
Question: Is there a $V' \subset V$ with $|V'| \geq K$ such that the subgraph of G induced by V' has property Π?

It is NP-hard for any property Π that holds for arbitrarily large graphs, does not hold for all graphs, and is hereditary (holds for all induced subgraphs of G whenever it holds for G). If it can be determined in polynomial time whether

Π holds for a graph, then the problem is NP-complete. Examples include "G is a clique", "G is an independent set", "G is planar", "G is bipartite", "G is chordal."

Δ-confluency is a property that holds for arbitrarily large graphs, does not holds for all graphs, and is hereditary (every induced subgraph of a Δ-confluent graph is Δ-confluent.) It can be determined in linear time whether a graph is Δ-confluent. Thus the maximum Δ-confluent subgraph problem is NP-complete. □

Instead of drawing the maximum subgraph Δ-confluently and adding the rest back, We could compute a Δ-confluent subgraph cover of the input graph, visualize each subgraph as a Δ-confluent drawing, and overlay them together. This leads to the Δ-CONFLUENT SUBGRAPH COVERING PROBLEM. Like the maximum Δ-confluent subgraph problem, we expect this problem to be hard as well.

This alternative way is related to the concept of *simultaneous embedding* (see [3, 11, 15, 16]). To visualize an overlay of Δ-confluent subgraph drawings is to draw trees simultaneously. However *simultaneously embedding* draws only two graphs that share the same vertex set V, while a Δ-confluent subgraph cover could have a cardinality larger than two. Furthermore, the problem of simultaneously embedding (two) trees hasn't been solved.

Other interesting problems include:

- How to compute the drawing with optimum area (or number of bends, etc.) for a Δ-confluent graph?
  Generally hexagonal grid drawings by transforming orthogonal drawings are not area (number of bends, etc.) optimal. If subtree separation is not required, hexagonal grid drawings with more compact area or smaller number of bends can be achieved. Maybe a simple incremental algorithm would work.
- The underlying track system here is topologically a tree. What classes of graphs can we get if other structures are allowed?

References

[1] S. Aziza and T. Biedl. Hexagonal grid drawings: Algorithms and lower bounds. In J. Pach, editor, *Graph Drawing (Proc. GD '04)*, volume 3383 of *Lecture Notes Comput. Sci.*, pages 18–24. Springer-Verlag, 2005.

[2] H. Bandelt and H. M. Mulder. Distance-hereditrary graphs. *J. Combin. Theory Ser. B*, 41:182–208, 1986.

[3] P. Brass, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. B. Mitchell. On simultaneous graph embedding. In *8th Workshop on Algorithms and Data Structures*, pages 243–255, 2003.

[4] R. P. Brent and H. T. Kung. On the area of binary tree layouts. *Inform. Process. Lett.*, 11:521–534, 1980.

[5] S. Bridgeman, A. Garg, and R. Tamassia. A graph drawing and translation service on the WWW. In S. C. North, editor, *Graph Drawing (Proc. GD '96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 45–52. Springer-Verlag, 1997.

[6] T. M. Chan, M. T. Goodrich, S. R. Kosaraju, and R. Tamassia. Optimizing area and aspect ratio in straight-line orthogonal tree drawings. In S. North, editor, *Graph Drawing (Proc. GD '96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 63–75. Springer-Verlag, 1997.

[7] S. Y. Choi. Orthogonal straight line drawing of trees. M.Sc. thesis, Dept. Comput. Sci., Univ. Brown, Providence, RI, 1999.

[8] P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Comput. Geom. Theory Appl.*, 2:187–200, 1992.

[9] G. Damiand, M. Habib, and C. Paul. A simple paradigm for graph recognition: Application to cographs and distance hereditary graphs. *Theor. Comput. Sci.*, 263(1–2):99–111, 2001.

[10] M. Dickerson, D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent drawing: Visualizing nonplanar diagrams in a planar way. In G. Liotta, editor, *Graph Drawing (Proc. GD '03)*, volume 2912 of *Lecture Notes Comput. Sci.*, pages 1–12. Springer-Verlag, 2004.

[11] C. A. Duncan, D. Eppstein, and S. G. Kobourov. The geometric thickness of low degree graphs. In *20th Annual ACM-SIAM Symposium on Computational Geometry (SCG '04)*, pages 340–346, 2004.

[12] P. Eades, T. Lin, and X. Lin. Two tree drawing conventions. *Internat. J. Comput. Geom. Appl.*, 3:133–153, 1993.

[13] P. D. Eades. Drawing free trees. *Bulletin of the Institute for Combinatorics and its Applications*, 5:10–36, 1992.

[14] D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent layered drawings. In J. Pach, editor, *Graph Drawing (Proc. GD '04)*, volume 3383 of *Lecture Notes Comput. Sci.*, pages 184–194. Springer-Verlag, 2005.

[15] C. Erten and S. G. Kobourov. Simultaneous embedding of a planar graph and its dual on the grid. In M. Goodrich and S. Kobourov, editors, *Graph Drawing (Proc. GD '02)*, volume 2518 of *Lecture Notes Comput. Sci.*, pages 575–587. Springer-Verlag, 2003.

[16] C. Erten and S. G. Kobourov. Simultaneous embedding of planar graphs with few bends. In J. Pach, editor, *Graph Drawing (Proc. GD '04)*, volume 3383 of *Lecture Notes Comput. Sci.*, pages 195–205. Springer-Verlag, 2005.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[18] A. Garg, M. T. Goodrich, and R. Tamassia. Area-efficient upward tree drawings. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 359–368, 1993.

[19] A. Gregori. Unit length embedding of binary trees on a square grid. *Inform. Process. Lett.*, 31:167–172, 1989.

[20] P. Hammer and F. Maffray. Completely separable graphs. *Discrete Appl. Math.*, 27:85–99, 1990.

[21] P. Hui, M. Schaefer, and D. Štefankovič. Train tracks and confluent drawings. In J. Pach, editor, *Graph Drawing (Proc. GD '04)*, volume 3383 of *Lecture Notes Comput. Sci.*, pages 318–328. Springer-Verlag, 2005.

[22] P. J. Idicula. Drawing trees in grids. Master's thesis, Department of Computer Science, University of Auckland, 1990.

[23] G. Kant. Hexagonal grid drawings. In *Proc. 18th Internat. Workshop Graph-Theoret. Concepts Comput. Sci.*, 1992.

[24] C. E. Leiserson. Area-efficient graph layouts (for VLSI). In *Proc. 21st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 270–281, 1980.

[25] C. E. Leiserson. *Area-efficient graph layouts (for VLSI)*. ACM Doctoral Dissertation Award Series. MIT Press, Cambridge, MA, 1983.
[26] P. T. Metaxas, G. E. Pantziou, and A. Symvonis. Parallel h-v drawings of binary trees. In *Proc. 5th Annu. Internat. Sympos. Algorithms Comput.*, volume 834 of *Lecture Notes Comput. Sci.*, pages 487–495. Springer-Verlag, 1994.
[27] E. Reingold and J. Tilford. Tidier drawing of trees. *IEEE Trans. Softw. Eng.*, SE-7(2):223–228, 1981.
[28] K. J. Supowit and E. M. Reingold. The complexity of drawing trees nicely. *Acta Inform.*, 18:377–392, 1983.
[29] J. S. Tilford. Tree drawing algorithms. Technical Report UIUCDCS-R-81-1055, Department of Computer Science, University of Illinois at Urbana-Champaign, 1981.
[30] L. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Comput.*, C-30(2):135–140, 1981.
[31] J. Q. Walker II. A node-positioning algorithm for general trees. *Softw. – Pract. Exp.*, 20(7):685–705, 1990.
[32] C. Wetherell and A. Shannon. Tidy drawing of trees. *IEEE Trans. Softw. Eng.*, SE-5(5):514–520, 1979.