Improved orthogonal drawings of 3-graphs

Therese C. Biedl

RUTCOR, Rutgers University,
P.O. Box 5062, New Brunswick, NJ 08903,
therese@rutcor.rutgers.edu, FAX (908) 445-54 72

Abstract

In this paper we deal with orthogonal drawings of graphs with maximum degree 3. We present a heuristic that works in linear time, and embeds every connected simple graph in a grid where the width and height each is at most $\lceil\frac{n+1}{2}\rceil$, and their sum is at most n. The number of bends is at most $\frac{n}{2}+2$. For the number of bends, this is very close to optimality.

1 Background

Orthogonal graph drawings are an important tool for graph layout, since the minimum angle of 90° makes the drawings easily readable. Specific uses include Data Flow Diagrams and Entity Relationships Diagrams. The precise definition is as follows:

An *orthogonal drawing* of a graph is an embedding in the plane such that vertices are drawn as points and edges are drawn as sequences of horizontal and vertical line segments. A point where the drawing of an edge changes its direction is called a *bend* of this edge. We assume that all vertices and bends are placed on points with integer coordinates. If the drawing can be enclosed by a rectangle of *width* n_1 and *height* n_2, we call it a drawing with *grid-size* $n_1 \times n_2$. The grid-size and the number of bends are two of the most important measurements of the quality of a drawing.

Orthogonal drawings exist only if every vertex in the graph has at most four incident edges. On the other hand, every such graph has an orthogonal drawing (for example with the algorithm in [1]). The problem of minimizing the number of bends is $\mathcal{NP}$-complete [3], and so is the question whether a graph can be embedded in a grid of prescribed size [4, 2].

Assume G is a simple connected graph with n vertices. The algorithm of Biedl gives an $n \times n$-grid and $2n+2$ bends [1]. The area was improved by Papakostas and Tollis to $0.76n^2$ [5]. In the same paper, the authors also deal with *3-graphs*, i.e. graphs with maximum degree 3. They provided an algorithm for biconnected graphs that produces a grid where the sum of the width and height is at most $n+2$, and there at most $\frac{n}{2}+3$ bends. For graphs that are not biconnected, the number of bends increases.

In this paper, we develop a different algorithm for 3-graphs, which improves on the one by Papakostas and Tollis in two ways. First, we show that no increase is necessary for graphs that are not biconnected: Every connected 3-graph can be embedded with at most $\frac{n}{2}+2$ bends. Secondly, we balance the relation of height and width: in our algorithm, the height and the width each is at most $\lceil\frac{n+1}{2}\rceil$. The difference between the height and the width is at most 2, while no such bounds are known for the algorithm by Papakostas and Tollis.

Our algorithm is very close to optimality in the number of bends. The K_4 needs $4 = \frac{n}{2}+2$ bends in any orthogonal drawing. Concerning larger n, there exist graphs which need $\frac{n}{2}+1$ bends [6]. Our algorithm can be improved to match this for graphs that are not biconnected. Thus the only small gap is for biconnected graph with $n \geq 5$ vertices. The best known lower bound here is $\frac{n}{2}$ bends [5].

2 Algorithm for 3-graphs that are not 3-regular

Let G be a simple connected 3-graph with $n = n(G)$ vertices and m edges. G is called *3-regular* if every vertex has degree 3. In this case, $m = \frac{3}{2}n$, and therefore n is even. In this section, we assume that G is not 3-regular, i.e. there exists at least one vertex with degree ≤ 3. For a 3-regular, we achieve this by subdividing one edge.

The basic idea to draw G is to build up the drawing of G successively in some ordering $\{v_1, \ldots, v_n\}$ of the vertices. As first vertex, we choose a vertex with degree ≤ 3. We use a *DFS-ordering*, which is the ordering produced while traversing the graph with depth first search.

We consider the edges to be directed from the lower-numbered to the higher-numbered vertex. Thus, we have for every vertex the neighbors divided into *predecessors* and *successors*, and the edges divided into *incoming* and *outgoing* edges. We denote the number of incoming (outgoing) edges of v with $indeg(v)$ ($outdeg(v)$). In a DFS-ordering, all vertices but v_1 have $indeg \geq 1$. Since we have a 3-graph and since $deg(v_1) \leq 2$, all vertices have $outdeg \leq 2$. When writing an edge e as (v_i, v_j), we assume that e is directed from v_i to v_j.

To route the edges, we color them with two different colors in a special way. Namely, let an rb-*coloring* be a coloring of the edges with the colors red and blue, such that for every vertex v (1) the incoming edges all have the same color, and (2) the outgoing edges have both colors, if $outdeg(v) \geq 2$. We will show the existence of such a coloring later.

The invariant of our algorithm is that when $\{v_1, \ldots, v_k\}$ are embedded, then for every edge $e = (v_i, v_l)$, $i \leq k < l$ that is colored red (blue), the ray leaving from v_i to the right (bottom) direction is empty. We embed $v_i, i = 1, \ldots, n$ as follows:

- If $indeg(v_i) = 0$: Since we used a DFS-ordering, and since the graph was connected, we must have $i = 1$, so this case happens only once. If $outdeg(v_1) = 2$, then one outgoing edge is red, and another outgoing edge is blue. v_1 can be embedded with one row and column.

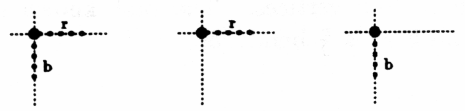

Figure 1: Embedding of v_1, for different degrees and colorings. Free rays are show as thick dotted lines.

- If $indeg(v_i) \geq 1$: By the invariant, either all incoming edges use rays to the right, or they all use rays to the bottom. Therefore, we can add a grid-line perpendicular to the rays of the incoming edges of v_i, and place v_i with $indeg(v)-1$ bends. If $indeg(v_i) = 2$, then we have two possible placements for v_i, and we choose the one as far to the right and down as possible.

  If $indeg(v_i) \leq 2$, then thus we have the ray to the right and the ray to the bottom free at v_i. Since we have a 3-graph, v_i has at most two outgoing edges, and if it has two, then they are colored in different colors. Therefore we can always assign the correct ray to the outgoing edges.

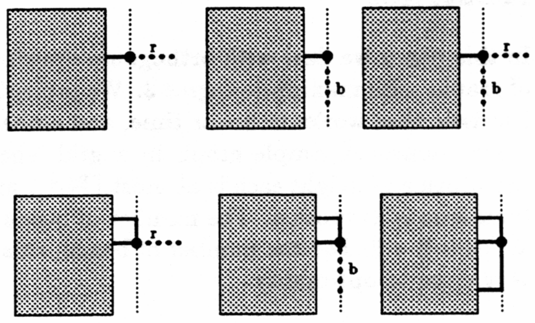

Figure 2: Embedding v_i, $i \geq 2$. We show only the case that all incoming edges are red.

Lemma 2.1 *There are at most* $m-n+1 \leq \frac{n+1}{2}$ *bends.*

Proof: For every vertex $v_i \neq v_1$, we add $indeg(v_i) - 1$ bends; for v_1, we use no bend. Since $indeg(v_1) = 0$, the total number of bends is $\sum_{v \in V}(indeg(v)-1)+1 = m-n+1$. Since we have a 3-graph and since $deg(v_1) \leq 2$, we have $2m = \sum_{v \in V} deg(v) \leq 3(n-1)+2$, therefore $m \leq \frac{3n-1}{2}$, and the number of bends is at most $m - n + 1 \leq \frac{n}{2} + \frac{1}{2}$. □

Since we add rows and columns at the bottom or at the right, we can assign them increasing numbers. Then comparing them takes $\mathcal{O}(1)$ time, and handling each vertex takes $\mathcal{O}(1)$ time as well. This gives a total time complexity of $\mathcal{O}(n)$, assuming that we can compute the *rb*-coloring in $\mathcal{O}(n)$ time.

3 Computing an *rb*-coloring

We will show the existence of an *rb*-coloring for any finite graph, not only for 3-graphs. Perform a depth first search for computing the DFS-ordering $\{v_1, \ldots, v_n\}$, and for directing the edges. This yields the DFS-tree T rooted at

v_1. We distinguish the edges into *tree-edges* and *non-tree-edges* (also called *NT-edges*). A DFS-tree has the property that if (v_i, v_j) is an edge, then there exists a directed path of tree-edges from v_i to v_j. Therefore, if v_i has any outgoing edges, then it also must have at least one outgoing tree-edge.

In the following, we show the algorithm to compute the *rb*-coloring. While doing this, we also define a partition of the edges into ***streaks***. These streaks are not necessary for computing just any *rb*-coloring, but they will be useful later when we want to compute a special *rb*-coloring.

rb-COLORING

```
for $i = n-1$ downto 1:
   if $outdeg(v_i) \geq 1$:
         Let $e_1, \ldots, e_s$ be the outgoing tree-edges
               of $v_i$. We have $s \geq 1$.
         Let $e'_1, \ldots, e'_r$ be the outgoing NT-edges
               of $v_i$, $r \geq 0$.
         We will show: $e_1, \ldots, e_s$ are uncolored,
               $e'_1, \ldots, e'_r$ are colored.

         if $r = 0$:
               Start a new streak,
                     consisting of $e_1, \ldots, e_s$.
               Color $e_1$ red, and
                     color $e_2, \ldots, e_s$ blue.
         else
               Assign $e_1, \ldots, e_s$
                     to the streak of $e'_1$.
               Color $e_1, \ldots, e_s$ in the
                     opposite color of $e'_1$.

         for $j = 1, \ldots, s$,
               Assume $e_j = (v_i, v_k)$.
               Let $\bar{e}_1, \ldots, \bar{e}_p$ be the
                     incoming edges of $v_k$, $\bar{e}_1 = e_j$.
               $\bar{e}_2, \ldots, \bar{e}_k$ are NT-edges.
               Assign $\bar{e}_2, \ldots, \bar{e}_p$ to the streak of $e_j$.
               Color them in the color of $e_j$.
```

Clearly, rb-COLORING works in $\mathcal{O}(m)$ time. By induction one can show directly that if (v_j, v_k) is a tree-edge, then it is colored in step $i = j$, while if it is an NT-edge, it is colored in step $i = k$. This proves the invariant of the algorithm. So we only need to show the correctness.

Lemma 3.1 *The resulting coloring is an rb-coloring.*

Proof: Assume v_k has $indeg(v_k) \geq 1$. Then v_k must have an incoming tree-edge $e = (v_j, v_k)$. We colored e with some color in step $i = j$. In the same step, we also colored all incoming edges of v_k with the same color as e. So incoming edges have one color.

Assume v_k has $outdeg(v_k) \geq 2$, and consider the step $i = k$. We distinguish whether $r = 0$ or $r > 0$. If $r = 0$, we colored e_1 red, and $e_2, \ldots, e_s$ blue. Since $s = outdeg(v_k) \geq 2$, we have both colors among the outgoing edges. If $r > 0$, then we colored e_1 in the opposite color of e'_1, so we have both colors among the outgoing edges. □

4 A balanced *rb*-coloring

We will in the following assume again that we have a 3-graph, and that $deg(v_1) \leq 2$. Let an *rb*-coloring be given, and define rv (bv) to be the number of red (blue) tree-edges. We have a width and height of 0 after embedding v_1. Every v_i, $i \geq 2$ must have an incoming tree-edge. When embedding v_i, we add one unit of width if the incoming tree-edge is red, and one unit of height if the incoming tree-edge is blue. Therefore, the obtained grid-size is $rv \times bv$. Using just any *rb*-coloring, we could get a very big width and small height. For example, for an n-cycle we color all edges red, except for the edge (v_1, v_2), so we get a $(n-2) \times 1$-grid. But width and height can be balanced with a special *rb*-coloring.

Assume that S is a streak computed during rb-COLORING. For every edge in S, we define the *generation* as follows. The tree-edges that started the streak are the generation 0. An NT-edge e' is added to the streak because of a tree-edge e incoming into the same vertex. We set the generation of e' to be one higher than the generation of e. A tree-edge e that is not in the first generation is added to the streak because of an NT-edge e' outgoing of the same vertex. We set the generation of e to be the same as the generation of e'. Set $t(i)$ and $nt(i)$ to be the number of tree-edges (NT-edges) of generation i.

Lemma 4.1 *Assume* G *is a 3-graph and* $\text{deg}(v_1) \leq 2$. *Let* $e = (v_j, v_k)$ *be a tree-edge of generation* g. *If* $g \geq 1$, *then* $\text{outdeg}(v_j) = 2$, *and* $\text{indeg}(v_k) \leq 2$.

Proof: We know $outdeg(v_j) \leq 2$ by the assumptions. Since the generation of e was $g \geq 1$, e was added to the streak due to some NT-edge $e' = (v_j, v_l)$. So $outdeg(v_j) \geq 2$.

Since e' was an NT-edge, there is a directed path P of tree-edges from v_j to v_l. The first edge on this path must be e, since we just showed that v_j has only one outgoing tree-edge. Therefore, the endpoint v_k of e also must have an outgoing tree-edge (the second edge on P), which implies $indeg(v_k) \leq 2$. □

With this, one can show that $t(i+1) \leq nt(i) = t(i)$ for $i \geq 1$. Also, since every vertex has $indeg \leq 3$, we have $t(1) = nt(1) \leq 2t(0)$. Since every vertex has $outdeg(v) \leq 2$, we have $t(0) \leq 2$. The streak thus has a structure as the one shown in Figure 3. Let $R(S)$ and $B(S)$ be the number of red-colored and blue-colored tree-edges in S, and define the *discrepancy* $\Delta(S) = |R(S) - B(S)|$.

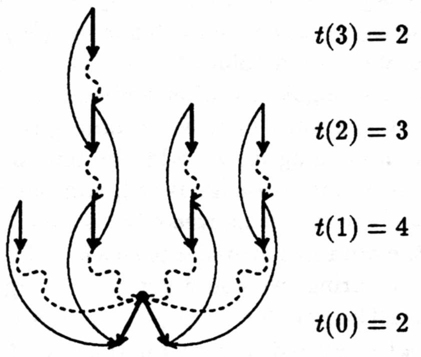

Figure 3: Examples of a streak, shown with the solid arrows (the thick lines are tree-edges). The dashed arrows mark that there must be a path of tree-edges (they need not belong to the streak).

Lemma 4.2 *Assume G is a 3-graph and* $\deg(v_1) \leq 2$. *Then for any streak S, $\Delta(S) \leq 2$.*

Proof: Assume first $t(0) = 1$. We colored the single tree-edge of generation 0 red. Therefore, the NT-edges of generation 1 are colored blue, and so are the tree-edge of generation 1. Iterating this, we find that the tree-edges of generation g are red if g is even, and blue if g is odd. Therefore, $R(S) = t(0) + t(2) + t(4) + \ldots \leq t(0) + t(1) + t(3) + \ldots = 1 + B(S)$, and $B(S) = t(1) + t(3) + t(5) + \ldots \leq 2t(0) + t(2) + t(4) + \ldots = 1 + R(S)$. So $\Delta(S) \leq 1$.

Assume $t(0) = 2$. The two outgoing edges e_1 and e_2 of v define two substreaks S_1 and S_2, and each of those has discrepancy $\Delta(S_i) \leq 1$. Combining the two, we get a discrepancy of at most 2. □

For example, in Figure 3, the discrepancy is 2.

Note that if we switch all colors to the opposite in one streak, the resulting coloring is still an *rb*-coloring. Therefore, we can switch the colorings for some of the streaks to get an *rb*-coloring where $|rv - bv| \leq 2$. We may switch all colors to get $rv \geq bv$. Computing the $\Delta(S_i)$ can be done while coloring the streaks, and we recolor every edge at most twice. Since $m \leq \frac{3}{2}n$, we have shown the following.

Lemma 4.3 *Let G be a 3-graph, oriented with a DFS-ordering $\{v_1, \ldots, v_n\}$ with* $\deg(v_1) \leq 2$. *Then there exists an* rb*-coloring such that $0 \leq rv - bv \leq 2$. It can be computed in $O(n)$ time.*

Theorem 1 *There exists a linear time algorithm to draw any connected 3-graph G orthogonally in a $\lceil \frac{n+1}{2} \rceil \times \lceil \frac{n+1}{2} \rceil$-grid. The sum of width and height is at most n, and the number of bends is at most $\frac{n}{2} + 2$.*

Proof: Assume G is not 3-regular. Choose v_1 with $deg(v_1) \leq 2$, and start the depth first search with v_1. Then, compute an *rb*-coloring as in Lemma 4.3, and apply the algorithm. We get an $rv \times bv$-grid, with $rv \geq bv \geq rv - 2$. Since $rv + bv$ counts the tree-edges, $rv + bv = n - 1$; therefore $rv = n - 1 - bv \leq n - 1 - rv + 2$, or $rv \leq \frac{n+1}{2}$. The number of bends is $\frac{n+1}{2}$, by Lemma 2.1.

If G is 3-regular, we subdivide one edge and get a graph G' with $n' = n + 1$ vertices. G' is not 3-regular, so we can embed it in a grid of width and height at most $\lceil \frac{n'+1}{2} \rceil = \lceil \frac{n+2}{2} \rceil = \lceil \frac{n+1}{2} \rceil$, since n is even. As shown in the first half of the proof, the sum of width and height is in fact $n' - 1 = n$, and there are at most $\frac{n'+1}{2}$ bends. In order to get a drawing of G, we remove the vertex of the subdivision, and this may create one more bend. So the drawing of G has $1 + \frac{n+2}{2} = \frac{n}{2} + 2$ bends. □

5 Remarks

In this paper, we presented an algorithm that produces an orthogonal drawing of any 3-graph, in a grid of width and height $\lceil \frac{n+1}{2} \rceil$, and with at most $\frac{n}{2} + 2$ bends. We have the following remarks:

- Storer [6] gave a class of connected simple 3-graphs that need $\frac{n}{2} + 1$ bends in any orthogonal drawing, so for the bends we are close to optimality. In fact, we are optimal for small n: The K_4 has 4 vertices and needs 4 bends in any orthogonal drawing (otherwise the graphs by Storer could be drawn with less bends).

- For graphs that are not biconnected, the lower bound by Storer can be matched with our algorithm, with only a small change. There is hardly any practical advantage of this change, but it is nice to know that the upper and lower bounds match.

  Namely, assume G is not biconnected. Then there exists a cutvertex, and it is incident to an edge whose removal splits G into two subgraphs, G_1 and G_2. Let the endpoints of this edge be v_1 and v_2, with $v_i \in G_i$, $i = 1, 2$. Embed G_i with v_i as the first vertex, this gives a drawing with $\frac{n(G_i)+1}{2}$ bends. v_i is in the left upper corner of the drawing of G_i, and by rotating the drawing of G_2, we can connect v_1 and v_2 without adding a bend. Therefore, the final drawing has $\frac{n(G)}{2} + 1$ bends.

- In our algorithm, we always added a new row/column when needed. For an implementation, one should try whether it is possible to reuse an old row or column. This will often save grid-size. Can any better worst case lower bounds be shown using this approach?

- Is there a simpler way to determine the *rb*-coloring with $|rv - bv| \leq 2$? Is there a straight-forward characterization of those tree-edges that should be red?

- Is it possible to use colorings for other orthogonal graph drawing algorithms as well? Notice that the algorithm in [1] is in some sense an algorithm where all edges have been colored with one color. It would be nice to see algorithms where edges are colored differently, so that then the drawing grows to up to four sides simultaneously. One would expect that this spreads out the vertices more evenly in the grid.

Acknowledgements

The authors wants to thank Endre Boros and Alexander Lawrenz for useful discussions.

References

[1] T. Biedl, Embedding Nonplanar Graphs into the Rectangular Grid, *Rutcor Research Report* 27-93. [1]
See also: T. Biedl, and G. Kant, A better heuristic for orthogonal graph drawings, *Proc. 2nd European Symp. on Algorithms (ESA'94), Lecture Notes in Comp. Science* 855, Springer-Verlag (1994), pp. 124–135.

[2] M. Formann, and F. Wagner, The VLSI layout problem in various embedding models, *Graph-Theoretic Concepts in Comp. Science (16th WG'90)*, Springer-Verlag, Berlin/Heidelberg, 1992, pp. 130–139.

[3] A. Garg, and R. Tamassia, On the computational complexity of upward and rectilinear planarity testing, *Proc. Graph Drawing '94, Lecture Notes in Comp. Science* 894, Springer Verlag (1994), pp. 286–297 .

[4] M.R. Kramer, and J. van Leeuwen, The complexity of wire routing and finding minimum area layouts for arbitrary VLSI circuits. *Advances in Computer Research, Vol. 2: VLSI Theory*, JAI Press, Reading, MA, 1992, pp. 129–146.

[5] A. Papakostas, and I.G. Tollis, Improved algorithms and bounds for orthogonal drawings, *Proc. Graph Drawing '94, Lecture Notes in Comp. Science* 894, Springer-Verlag (1994), pp. 40–51.
Revised and corrected version at *http://wwwpub.utdalllas.edu/~tollis*

[6] J. Storer, On minimal node-cost planar embeddings, *Networks* 14 (1984), 181–212.

[1] Rutcor Research Reports are available via anonymous FTP from *rutcor.rutgers.edu*, directory */pub/rrr*; or on the WWW at *http://rutcor.rutgers.edu/~rrr*.