A Pairing Technique for Area-Efficient Orthogonal Drawings* (Extended Abstract)

Achilleas Papakostas and Ioannis G. Tollis

Dept. of Computer Science
The University of Texas at Dallas
Richardson, TX 75083-0688
email: papakost@utdallas.edu, tollis@utdallas.edu

Abstract. An orthogonal drawing of a graph is a drawing such that vertices are placed on grid points and edges are drawn as sequences of vertical and horizontal segments. In this paper we present linear time algorithms that produce orthogonal drawings of graphs with n nodes. If the maximum degree is four, then the drawing produced by our first algorithm needs area no greater than $0.76n^2$, and introduces no more than $2n+2$ bends. Also, every edge of such a drawing has at most two bends. Our algorithm is based on forming and placing pairs of vertices of the graph. If the maximum degree is three, then the drawing produced by our second algorithm needs at most $\frac{1}{4}n^2$ area, and at most $\lfloor \frac{n}{2} + 2l + 1 \rfloor$ bends ($\lfloor \frac{n}{2} \rfloor + 3$ bends, if the graph is biconnected), where l is the number of biconnected components that are leaves in the block tree. For biconnected graphs, this algorithm produces optimal drawings with respect to the number of bends (within a constant of two), since there is a lower bound of $\frac{n}{2}+1$ in the number of bends for orthogonal drawings of degree 3 graphs.

1 Introduction and Preliminaries

Research on algorithms for drawing graphs has received increasing attention recently. For a survey of graph drawing algorithms and other related results see the annotated bibliography of Di Battista, Eades, Tamassia and Tollis [4]. In this paper we focus on the problem of *orthogonal* drawings of graphs, that is drawings in which each edge of the graph is a polygonal chain consisting of horizontal and vertical segments. Orthogonal drawings are grid drawings, such that vertices and bends correspond to grid points (i.e., they have integer coordinates), and edges correspond to grid paths. A graph admits an orthogonal drawing if it has maximum degree 4. There are various aesthetic criteria for orthogonal drawings that look "nice"; minimizing the area, bends and crossings are some of these criteria. Our goal, here, is to minimize the area and to obtain drawings with a small number of bends.

* Research supported in part by NIST, Advanced Technology Program grant number 70NANB5H1162.

Several results have appeared in the literature regarding planar orthogonal drawings of graphs. In [17] and [19] it is shown that every biconnected planar graph of maximum degree 4 can be embedded in an $n \times n$ grid with $2n+4$ bends. If the graph is not biconnected then the total number of bends rises to $2.4n+2$. In all cases, no more than 4 bends per edge are required. The algorithms of [19] take linear time and produce drawings, such that at most one edge may have 4 bends. Kant [10] shows that if the graph is triconnected of maximum degree 4, then it can be drawn on an $n \times n$ grid with at most 3 bends per edge. The total number of bends is no more than $\lceil \frac{3}{2}n \rceil + 3$. Even and Granot [7] present an algorithm for obtaining an orthogonal drawing of a 4-planar graph with at most 3 bends per edge. If the embedding of a planar graph is fixed, then an orthogonal drawing with the minimum number of bends can be computed in $O(n^2 \log n)$ time [18]. If the planar embedding is not given, the problem is polynomially solvable for 3-planar graphs [6], and NP-hard for 4-planar graphs [9].

It is interesting to note that there is a lower bound of $2n - 2$ bends for biconnected planar graphs [20]. There are also examples of biconnected graphs [20], for which every bend-optimal planar drawing introduces a single edge with length $\Omega(n^2)$ having $\Omega(n)$ bends along it. Although these drawings achieve optimality in terms of the total number of bends, they are not aesthetically pleasing. This suggests that research in this area should concentrate on deriving orthogonal drawings of graphs with $O(1)$ bends per edge (usually 2 or 3) and $O(n)$ maximum edge length.

Upper and lower bounds have been proved in the case when the orthogonal drawing of a graph is not necessarily planar. Leighton [11] presented an infinite family of planar graphs which require area $\Omega(n \log n)$. Independently, Leiserson [12] and Valiant [21] showed that every planar graph of degree 3 or 4 has an orthogonal drawing with area $O(n \log^2 n)$. Valiant [21] showed that the orthogonal drawing of a general (nonplanar) graph of degree 3 or 4 requires area no more than $9n^2$, and described families of graphs that require area $\Omega(n^2)$. Valiant was not concerned about minimizing the total number of bends. In fact, an analysis of his construction shows that each edge can have up to 4 bends. Recently, Biedl [2] has presented a number of lower bounds on the area and number of bends for orthogonal drawings of different families of graphs of maximum degree 4.

Some algorithms for drawing general graphs of degree 4 in an orthogonal but not necessarily planar fashion have appeared in the literature. Schäffter [16] presented such an algorithm which constructs orthogonal drawings of graphs with at most two bends per edge. The area required was $2n \times 2n$. A better algorithm was presented in [1, 3], which draws the graph within an $n \times n$ grid with no more than 2 bends per edge. This algorithm introduced at most $2n + 2$ bends. In [14] we presented an algorithm that produces orthogonal drawings of graphs of maximum degree 4 in less than $n \times n$ area. The bounds depend on the number of balanced vertices (i.e., vertices that have 2 incoming and 2 outgoing edges) produced by the st-numbering. It turns out that for many graphs the number of balanced vertices is very low. So, although an upper bound of $0.81n^2$ can be shown for graphs with high enough number of balanced vertices, this is

not a general upper bound. Also, the bounds cannot be extended easily to one-connected graphs. An extensive experimental work appeared in [5] where four general purpose orthogonal graph drawing algorithms where implemented and compared.

In this paper we present an algorithm that produces an orthogonal drawing of an n-vertex graph of maximum degree 4 that needs at most $0.76n^2$ area and at most $2n + 2$ bends. The number of bends that appear on each edge is no more than two. Our algorithm is based on pairing the vertices of the graph, and placing the pairs in the plane. If the maximum degree is 3, then we present another algorithm which produces an orthogonal drawing which needs area at most $\frac{1}{4}n^2$ and $\lfloor \frac{n}{2} + 2l + 1 \rfloor$ bends ($\lfloor \frac{n}{2} \rfloor + 3$ bends, if the graph is biconnected). In this drawing, no more than one bend appears on each edge except for one edge, which may have at most two bends. Note that l is the number of the biconnected components of G that are leaves in G's block tree. A preliminary version of this algorithm appeared in [14]. Due to space limitations, we omit some details and proofs, but they can be found in [13].

We use n (resp. m) to denote the number of vertices (resp. edges) of a graph. An st-ordering is an ordering $v_1, v_2, \cdots, v_n$ of the vertices such that every v_j $(2 \leq j \leq n-1)$ has at least one predecessor and at least one successor, that is neighbors v_i, v_k with $i < j < k$. It is known that:

Theorem 1. *[8] Let G be biconnected and s, t be two vertices of G. Then there exists an st-ordering such that s is the first and t is the last vertex, and it can be computed in $O(m)$ time.*

An incremental linear time algorithm for producing orthogonal drawings of biconnected graphs of maximum degree 4 is presented in [1, 3]. The main result is given by the following theorem:

Theorem 2. *[1, 3] Let G be a biconnected graph of maximum degree 4. Then there exists a linear time algorithm which embeds G on an $n \times n$ grid with at most $2n + 2$ bends. Each edge is bent at most twice.*

2 Using Pairing to Improve the Area Bounds

In this section we present an algorithm for obtaining orthogonal drawings of general (nonplanar) biconnected graphs of maximum degree 4. Our algorithm achieves better results in terms of area than any previously known algorithm for orthogonal drawings. Let G be a general (nonplanar) biconnected graph of maximum degree 4. We obtain an st-numbering for G, with s as the source and t as the sink. We call a vertex with a incoming edges and b outgoing edges an a-b vertex $(1 \leq a, b \leq 4)$. For example, a vertex with one incoming edge and two outgoing, is a 1-2 vertex.

After the st-numbering is complete, we scan the graph G looking for those 1-1 vertices whose outgoing edge enters a 1-2 or a 1-3 vertex, if there are any. In order to simplify the description of our algorithm, we "absorb" these vertices into

a single edge until no 1-2 or 1-3 vertex has a 1-1 vertex as its unique immediate predecessor. Notice that no double edge is introduced when these 1-1 vertices are (temporarily) removed from G. Also notice that if a vertex was an a-b vertex in G and it was not removed as a result of the above procedure, it will still be an a-b vertex in the *reduced* graph. Let us use the notation G' for the reduced graph, and n' for its number of vertices. We then modify the st-numbering of G' so that there are no gaps in the st-number sequence assigned to the vertices of G' as a result of the removal of some 1-1 vertices from G.

The main idea of the algorithm is to create pairs of vertices of G' so that every 1-2, 1-3 and 2-2 vertex is a member of exactly one such pair. We distinguish between two different kinds of pairs:

- *row pairs.* The two vertices of such a pair share the same row in the final drawing of G. In other words, we save a row.
- *column pairs.* The two vertices of such a pair are placed in such a way so that a column is *reused* in the final drawing of G. A column is reused when at least two different edges use it. In other words, we save a column.

Algorithm Form_pairs considers the vertices of G' in reverse order of the st-numbering starting with the vertex which is right before the sink t. If a vertex already belongs to a pair, the vertex is called *assigned*, otherwise it is called *unassigned*. The next unassigned vertex we consider is always either a 1-2, 1-3, or 2-2 vertex and we pair it with some other lower numbered vertex in G'. The assignment of the 1-2, 1-3, and 2-2 vertices of G' to pairs is called *pairing* of G'. The vertex of a pair with the lowest st-number of the two is called the *first* vertex of the pair, and the other is called the *second* vertex of the pair.

In the rest of this paper, when we talk about a predecessor of a vertex in G or G' with respect to the st-numbering, we mean the immediate predecessor of this vertex.

Algorithm: Form_pairs
Input: A reduced graph G' of maximum degree 4 along with an st-numbering
Output: A pairing of G'

1. $i := n' - 1$;
2. While $i > 2$ do
   (a) Consider the next vertex v_i according to a decreasing order of the st-numbering.
   (b) If v_i is a 1-1, 2-1, or 3-1 vertex then
      - $j := 0$;
      - goto (e);
   (c) If v_i is a 1-2 or 1-3 vertex then
      - form a pair containing vertex v_{i-1} and v_i
      - $j := 1$;
      - goto (e);
   (d) If v_i is a 2-2 vertex then
      i. $j := 1$;

ii. While v_{i-j} is a 1-1, 2-1 or 3-1 vertex and v_{i-j} is not a predecessor of v_i do
   - $j := j + 1$;

iii. End_While

iv. form a pair containing vertex v_{i-j} and v_i

(e) $i := i - j - 1$;

3. End_While

This algorithm assigns every 2-2, 1-2 and 1-3 vertex v_i where $3 \leq i \leq n' - 1$ to one pair. Vertex v_2 (which is a 1-2 or 1-3 vertex) might or might not be paired with another vertex and this depends on the graph and the st-numbering. Every 1-2 or 1-3 vertex v_i is always paired with vertex v_{i-1}, when v_i is the next vertex that Algorithm Form_pairs considers in line (a). If the next vertex v_i to be considered is a 2-2 vertex, then Algorithm Form_pairs looks for the highest $j < i$ so that vertex v_j is one of the following types, and pairs v_i with v_j. Vertex v_j may be: a 2-2 vertex, or a 1-2 vertex, or a 1-3 vertex, or a 1-1 vertex which is also a predecessor of v_i, or a 2-1 vertex which is also a predecessor of v_i, or a 3-1 vertex which is also a predecessor of v_i.

Since v_2 and v_3 are not 3-1 vertices, the algorithm will pair all 2-2, 1-2 and 1-3 vertices except possibly for v_2. Let us assume that vertex v_i is paired with vertex v_j $(j < i)$ as a result of Algorithm Form_pairs. v_j might be a predecessor of v_i, or the two vertices might not have a predecessor-successor relationship. If the latter is the case, they are called *independent*. For different types of pairs, we draw the vertices of the pair in a different fashion. The pair $< v_i, v_j >$ can be one of the following types:

1. If v_i is a 2-2 vertex, we distinguish three cases for v_j:
   (a) v_j is a 2-2 vertex; we have a column pair. If v_j is v_i's predecessor, then a column which v_j's placement closes can be reused as shown in Fig. 1a. If v_i and v_j are independent, then we can always reuse one column (see Fig. 1b). Notice that in order for this column reuse to be possible, sometimes we might have to place v_i in a row that has lower y-coordinate than v_j's row. This placement is possible since the two vertices are independent, and can be depicted by Fig. 1b if we just swap the names of the two vertices shown.
   (b) v_j is a 1-1, 2-1 or 3-1 vertex and v_j is a predecessor of v_i. If v_j is 2-1 or 3-1 a column can be reused (i.e., we have a column pair), as shown in Fig. 1c. If v_j is 1-1, then the two vertices can share the same row (i.e., we have a row pair) as shown in Fig. 1d.
   (c) v_j is a 1-2 or 1-3 vertex. If v_j is a predecessor of v_i, we have a row pair, since v_i can be placed in the same row as v_j, at the intersection point between the edge coming from v_i's other predecessor and v_j's row, as shown in Fig 2a. Notice that v_i's other predecessor, say v_k, has to be such that $k < j$. If v_i and v_j are independent, we have a column pair since v_j can be placed in the row right above v_i's row and thus reuse one column (see Fig. 2b).

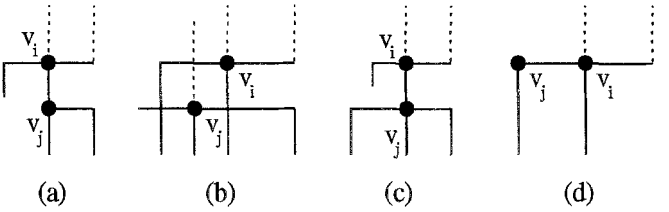

Fig. 1. v_i is a 2-2 vertex; a column is reused when: (a) v_j is 2-2 and v_i's predecessor, (b) v_j is independent from v_i, and (c) v_j is 2-1 or 3-1 and v_i's predecessor, (d) a row is shared when v_j is 1-1.

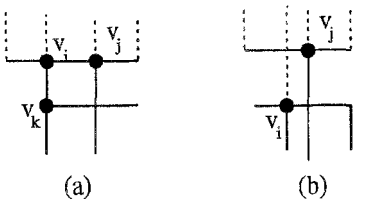

Fig. 2. (a) v_i and v_j share the same row, (b) v_j is placed in a row above v_i and a column is saved.

2. If v_i is a 1-2 or 1-3 vertex, it always pairs with vertex v_{i-1}. We distinguish four cases for v_{i-1}:
   (a) v_{i-1} is a 2-2, 2-1 or 3-1 vertex; we have a column pair. v_i is placed in a row above v_{i-1}'s row and a column is reused as described in Cases 1(a) and 1(b).
   (b) v_{i-1} is a 1-2 or 1-3 vertex and vertices v_i and v_{i-1} are independent. We have a row pair, and vertices v_i and v_{i-1} are placed in the same row as shown in Fig. 3a.
   (c) v_{i-1} is a 1-2 or 1-3 vertex and v_i's predecessor. If both of the following conditions hold (if not, see 2(d) below):
      - v_i is connected later to another vertex, say v_j, which is 1-1, 1-2 or 1-3. Or v_i is connected later to a 2-2 vertex v_j which is the second vertex of the pair of type 1(c) shown in Fig. 2a.
      - edge (v_{i-1}, v_i) has not absorbed any 1-1 vertices from the original graph G

   then we have a row pair and v_i and v_{i-1} are placed in the same row, as shown in Fig. 3b. In this case, we have to ensure that edge e (see Fig. 3b) will connect to v_j later in the drawing. This way, every edge is bent at most twice. Also notice that the total number of bends for both v_i and v_{i-1} is the same as if these two vertices were placed in two different rows.

(d) v_{i-1} is a 1-2 or 1-3 vertex and v_i's predecessor. If at least one of the two conditions described in pair type 2(c) does not hold, then v_i and v_{i-1} are placed in two different rows as shown in Fig. 3c. In this case, we make sure that the vertex (vertices) which is (are) supposed to be placed in the next row after v_i is (are) placed in the same row as v_i. Notice that this is possible since v_i together with the columns that it opens are placed entirely outside the boundaries of the current drawing. If new columns have to be opened in v_i's row, we open them in the middle of the current drawing, and to the immediate right or left of the vertex (vertices) that open(s) the new columns.

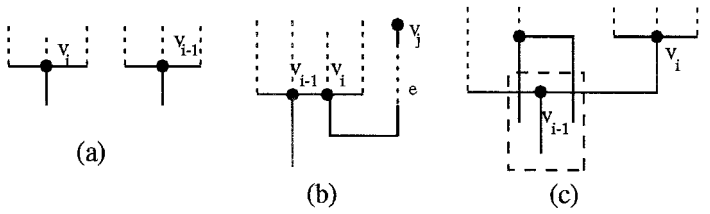

Fig. 3. (a) v_i and v_{i-1} are independent and share the same row, (b) v_i and v_{i-1} share the same row and edge e will connect to an appropriate vertex inserted later, (c) v_i shares the same row with the vertex that is next to be placed.

Let us assume that we have a pair of type 2(d) described in Fig. 3c, and the next pair to be placed is a column pair, say $< v_{i+1}, v_{i+2} >$. Let us also assume that there is an edge between v_i and v_{i+1}. Then, because v_i and v_{i+1} are placed in the same row, the savings for column pair $< v_{i+1}, v_{i+2} >$ really comes from reusing one of the columns that might have been used for the edge (v_i, v_{i+1}) had it not been drawn horizontally.

If it should happen that v_i is paired with a vertex v_j for which $j < i-1$, then this means that all vertices v_k where $j < k \leq i-1$ are 1-1, 2-1 or 3-1 vertices and they are not predecessors of v_i. Since this is the case, v_i can be placed in a row which is below the rows of vertices v_k, without affecting the pairing or the drawing algorithm.

Recall that in order to simplify our description, we absorbed some degree 2 vertices. After the drawing of G' is complete, we need to restore the degree 2 vertices of the original graph G which were absorbed at the beginning of the procedure. These vertices are placed primarily on bends, or on grid points (i.e., points of integer coordinates that do not have a crossing). In the extreme case where this is not possible, we introduce new rows as needed. Notice that all the other vertices of the drawing maintain their positions, that is the rows and columns in which they are placed.

We are now ready to present our algorithm formally.

Algorithm: 4_*ORTHOGONAL*
Input: A biconnected graph G of maximum degree 4.
Output: An orthogonal drawing of G.

1. Compute an st-numbering of G.
2. Produce a reduced graph G' and modify the st-numbering so that there are no gaps in the st-sequence.
3. Run Form_pairs on the reduced graph G'.
4. Place vertices v_1 and v_2 in the same row, if v_2 does not belong to a pair in which it shares a row with another vertex (see Fig. 4a). If v_1 and/or v_2 have degree less than 4, then the placement of v_1 and v_2 might require one or two rows. Figure 4b shows the case where v_1 had degree 3 and v_2 has degree 4. Notice that in this case there is only one bend along edge (v_1, v_2). If v_2 is assigned to a pair, we place v_1 as shown in Fig. 4c (if v_1 has degree 4). Vertex v_2 will be placed when its pair is considered.
5. REPEAT
   (a) Consider the next vertex v_i according to the st-numbering of G'.
   (b) If v_i has already been placed, then go to Step 6.
   (c) If vertex v_i is unassigned, then place v_i in a new row. Connect v_i with each vertex v_j $(j < i)$ such that (v_j, v_i) is a directed edge of G'. Add as many uncompleted edges as required, depending on v_i's outdegree.
   (d) If vertex v_i is assigned to a pair, then place v_i together with the other vertex in the same pair following the placement rules described above for the specific type of pair.
6. UNTIL the only remaining vertex is $v_{n'}$.
7. Insert $v_{n'}$ in a new row. If $v_{n'}$ is of degree 4, then there is an incoming edge that enters $v_{n'}$ from the top and bends twice. This edge is chosen to be the one that connects to $v_{n'-1}$.
8. Restore the degree 2 vertices of G that were absorbed in Step 2, as described above.
9. End.

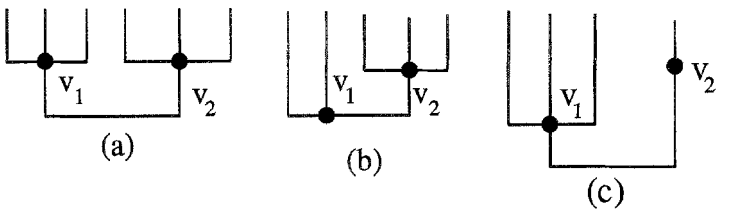

Fig. 4. v_1 and v_2 placed by Algorithm 4_ORTHOGONAL: (a) v_1 and v_2 can be placed in the same row, (b) the placement of v_1 and v_2 when v_1 has degree 3, and (c) the placement of v_1 when v_2 is assigned to a pair.

Notice that the difference between the given graph G and the reduced graph G' is that some degree 2 vertices of G have been (temporarily) removed, as discussed earlier in this section. In Step 3 of Algorithm 4_ORTHOGONAL we apply the pairing process to the vertices of G'. The pairing of the vertices of G' "transfers" to G since we can always assert that if two vertices participate in some pair in G', the same two vertices participate in the same pair in G.

Lemma 3. *Let us assume that there is a total of p_1 column pairs, p_2 1-1 vertices, p_3 2-1 vertices in G, and that $k_1 = p_1 + p_2 + \frac{p_3}{2}$. Then the width of the drawing of G will be at most $n + 1 - k_1$.*

Lemma 4. *Let us assume that there are k_2 row pairs of vertices in G. Then the height of the drawing will be $n + 1 - k_2$, when $k_2 \geq 1$, or n when $k_2 = 0$.*

Theorem 5. *Algorithm 4_ORTHOGONAL constructs an orthogonal drawing of an n-vertex degree 4 biconnected graph with area no more than $0.76n^2$. The total number of bends of the drawing is at most $2n + 4$, and no edge has more than two bends. The algorithm runs in $O(n)$ time.*

An important feature of the algorithm is that when we place vertices in the same row we save the potential crossings that would have been introduced if the vertices were placed in separate rows. Reusing columns also contributes to saving crossings significantly. Notice that in practice, the area which is typically required by the orthogonal drawing of a graph produced by 4_ORTHOGONAL is better than what the above theorem claims, for two reasons: the first is that, the total number of 1-1, 1-2, 1-3, 2-2 and 2-1 type vertices is usually much larger than $\frac{n}{2}$. The second reason is that when we form pairs, we typically expect some vertices of type 3-1 to participate as the second vertex in some column pairs of type 1(b) or 2(a). If this is the case, the total number of pairs increases.

In fact, experimentation of the performance of Algorithm 4_ORTHOGONAL was conducted on about 15 dense graphs of maximum degree 4. The size of these graphs varied from as small as 13 nodes to as big as 150 nodes. Each graph had a very small number (no more than six) of degree 3 vertices, whereas all the rest were vertices of degree 4. The set included both biconnected and non-biconnected graphs.

The first observation that we made is that the shape and area of the produced drawings depended heavily on the specific st-numbering that was employed. St-numberings that resembled DFS produced drawings in which the height was larger than the width, but the area was no more than $0.65n^2$. St-numberings that resembled BFS produced more squarish drawings, with shorter edges, but the column reuse was not as good as in the first case. As a result the area was a bit larger, but never more than $0.72n^2$. Finally, in all the cases, the number of bends was no larger than $2n$. We also noticed that for the larger graphs, the number of bends was significantly lower than $2n$.

Figure 5a shows a regular degree 4 graph with 13 vertices and Fig. 5b shows the orthogonal drawing which our algorithm produces for it. Notice that vertices 1 and 2, 3 and 4, and 6 and 7 are placed in the same row. Also, the pairs $< 10, 8 >$, $< 7, 6 >$, $< 5, 4 >$ and $< 3, 2 >$ save one column each. A total of 4 bends are

saved in the rows where vertices 3 and 4, and 6 and 7 are placed. Our drawing has height 11 and width 10.

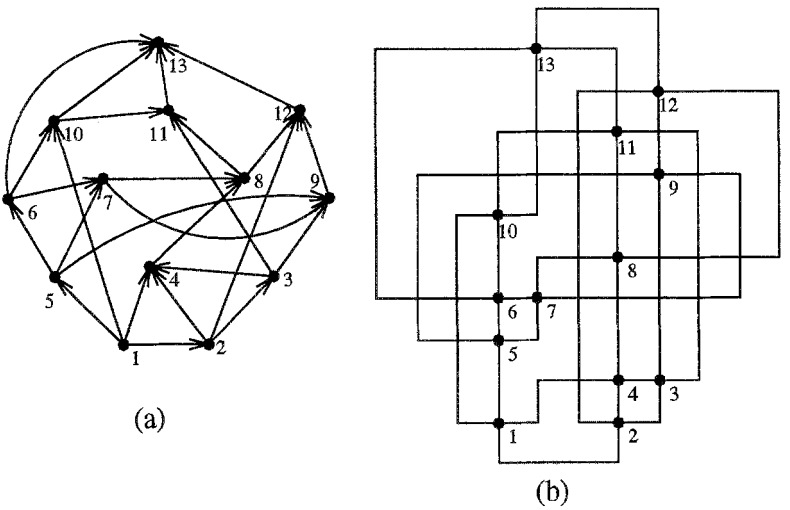

Fig. 5. (a) st-numbering of an example graph, (b) orthogonal drawing of the graph in Fig. 5a produced by Algorithm 4_ORTHOGONAL.

3 The One-Connected Case

In this section we extend our results to the case where the given graph is one-connected. The technique is based on breaking the graph into its biconnected components, which is also suggested in [3, 19]. In [3] it is shown that each component has to have no more than $2n-1$ bends and $(n-1)\times(n-1)$ area in order for the final drawing (after merging the components together) to have at most $2n+2$ bends and $n \times n$ area. Our technique is similar to the one in [3]. In order to apply this technique for one-connected graphs, we have to guarantee that, when Algorithm 4_ORTHOGONAL draws any biconnected graph of maximum degree 4, neither the height nor the width of the drawing can be larger than n. Moreover, the number of bends has to be at most $2n+2$. These bounds guarantee that the height or width of any biconnected component is at most $n-1$, and the number of bends is at most $2n-1$.

Notice that it is possible for Algorithm 4_ORTHOGONAL to produce a drawing with $2n+4$ bends or width $n+1$. Here we describe a technique which forces a row pair in G's pairing which additionally saves one column and two bends. We scan the vertices starting from v_2 and following the st-numbering, until we find the first vertex v that is not a 1-3 vertex. We distinguish two cases for v:

1. v is a 2-2 vertex. Let u be v's highest numbered predecessor. Vertices v and u do not belong to the same pair, since in that case we would have a row pair of the kind we are trying to form. We break the pair that v is in, and pair v with u. This new pair is a pair of type 1(c) (the case shown in Fig. 2a). Notice that in order to form the new pair, we have to break the pair that u was previously assigned to. However, doing so will not increase the width or the total number of bends.
2. v is a 3-1 vertex. We break any pair that v might have been assigned to with another higher numbered vertex. We check v's highest numbered 1-3 predecessor, say u (notice that, since v has three incoming edges, u cannot be vertex v_2). Vertex u has to be assigned to a pair with another 1-3 vertex, and u is the second vertex in that pair, as a result of the running of Algorithm Form_pairs. This pair becomes a pair of type 2(d) (we disregard whatever kind this pair was before). In this case, v will be placed in u's row.

From the above it follows that, if we have a regular degree 4 biconnected graph G, we can always place one 2-2 or 3-1 vertex (vertex v in the above description) in the same row as its highest numbered predecessor (vertex u in the above description). We accomplish this by forming a new row pair of type either 1(c) (u is the first vertex and v is the second vertex of the pair), or 2(d) (u is the second vertex of the pair, v becomes an unassigned vertex and is placed in u's row). In either case, the resulting drawing has at most $2n+2$ bends; also, the width of the drawing is at most n. Notice that although we might have to break two existing pairs, the new row pair that we form saves one row, one column and two bends. Therefore we have:

Theorem 6. *Algorithm 4_ORTHOGONAL constructs an orthogonal drawing of an n-vertex maximum degree 4 biconnected graph with area no more than $0.76n^2$. The total number of bends of the drawing is at most $2n + 2$, and no edge has more than 2 bends. The algorithm runs in linear time.*

We saw that the height of a drawing under Algorithm 4_ORTHOGONAL is $n + 1 - k_2$ (see Lemma 4), if vertices v_2 and v_3 form a row pair (in this case, $k_2 \geq 1$). For the rest of this section, we assume the same pairing process as described in the previous section, except that we will not count the row pair formed by v_2 and v_3 in k_2. In this way, we have a new k_2' for a biconnected graph, which is defined in the same way as in Lemma 4 and has the adjustment we just described. It follows that the height of the drawing of a biconnected graph is no more than $n - k_2'$ ($k_2' \geq 0$). From the discussion above, it holds that:

Lemma 7. *If we are given a biconnected graph G of maximum degree 4, then Algorithm 4_ORTHOGONAL can produce an orthogonal drawing of G whose size is as follows: the width of the drawing is at most $min(n, n + 1 - k_1)$ (see also Lemma 3), and the height of the drawing is at most $n - k_2'$ ($k_2' \geq 0$).*

Algorithm 4_ORTHOGONAL can be extended to the case of one-connected graphs of maximum degree 4. Let us assume that we have such a graph G.

We split G into its biconnected components, produce G's block tree and apply Algorithm 4_ORTHOGONAL (see Theorem 6) on each biconnected component separately. Then we put the components together to form the final drawing. We use an inductive approach for producing the drawing of G, which is similar to [3]. The base case is always a biconnected graph. In the induction step we consider a subtree of G's block tree and we split the subtree into a biconnected component G_0 (i.e., the root of the subtree) and (not necessarily biconnected) subgraphs $G_1, G_2, \cdots G_{q+s}$. Each one of the G_i's is already drawn according to the induction hypothesis. The drawing of the subtree of G's block tree then reduces to drawing G_0 and merging the G_i's at their appropriate places.

We now give a description of our technique. There are q subgraphs of the G_i's that are connected to G_0 through a bridge, while the rest s of the G_i's are connected to G_0 via a cutvertex which is shared by both G_0 and a G_i. Clearly, G_0 has a total of $q+s$ cutvertices. For those subgraphs G_i which are connected to G_0 through a bridge, the edge representing the bridge is not part of the drawing of the subgraph. This edge will be added when the drawing of the corresponding subgraph is merged with the drawing of G_0. First, we insert subgraph G_1. The cutvertex shared between G_0 and G_1, or the one of the two cutvertices of the bridge connecting G_0 and G_1 is in the top row of the drawing of G_1. We continue from there with the drawing of G_0.

G_0 is drawn as a biconnected graph, making sure that none of its $q + s$ cutvertices is the "final" vertex (i.e., vertex drawn in the top row, locally, in the subgraph considered). G_0 has some other vertex (say w) as the final vertex of the drawing. Vertex w will be used to attach the produced drawing of G_0 and the G_i's (i.e., the drawing of the subtree of G's block tree) to the drawing of another biconnected component considered at a later induction level. For this reason, vertex w must be a cutvertex for graph G. The important thing to note here is that we regard G_0 and G_1 as "one" subgraph. This means that the vertices of G_0 are placed (forming row or column pairs) as if we were continuing the drawing of G_1. In other words, G_1's "final" vertex as well as the three first vertices of G_0 may form pairs which count towards reducing the number of rows and columns of the subgraph consisting of the union of G_0 and G_1 (see Fig. 6a).

When the time comes to place a cutvertex v that is also shared by some G_i, we do the following: We rotate the drawing of G_i (G_i was already drawn with v as the final vertex) and place it in such a way so that a total of at least three rows and/or columns of the current drawing of previous components are reused. Note that the rows and/or columns that are reused as a result of G_i's insertion are different from the row and the column that vertex v is placed in. If additional rows and/or columns need to open up to accommodate G_i's drawing, we do so now. In order for this row/column reuse to be possible, we make sure that G_1 is selected so that it has at least four rows or at least four columns. If no such subgraph exists, we select G_1 to be the next largest graph. In Fig. 6b we can see how four consecutive subgraphs can be inserted so that three columns are reused at each time. The drawing of each one of these subgraphs has four columns, while the shared cutvertices are in the corners of the drawings of subgraphs G_j and

G_{j+1}, and in one of the middle columns of the drawings of subgraphs G_{j+2} and G_{j+3} (see Fig. 6b). Notice that when subgraph G_{j+3} is inserted, edge e might acquire an extra bend which is saved later, since edge e' does not have a bend.

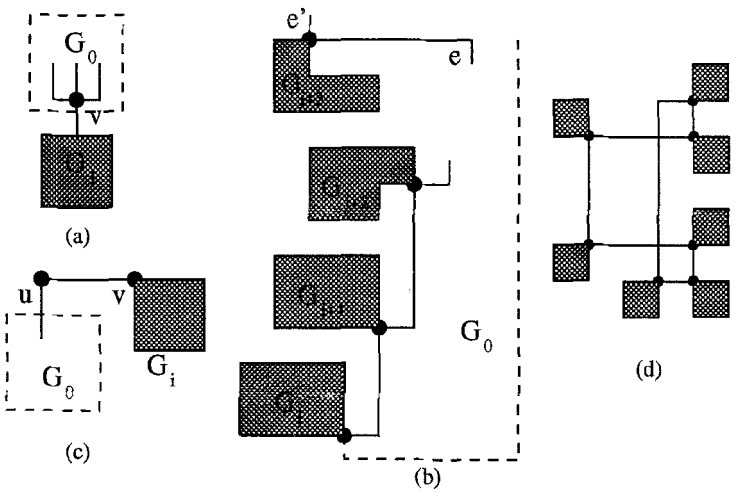

Fig. 6. G_0 continues the drawing of G_1, (b) examples of subgraph rotation, placement and column reuse, (c) reusing row(s) when G_i is connected to G_0 through a bridge, (d) drawing of a graph whose G_i's are small size graphs (each G_i here is a triangle).

We follow a similar procedure if subgraph G_i is connected to G_0 through a bridge. However, when such a subgraph G_i is inserted to the drawing, we tend to reuse mostly rows. For example, let b be the bridge that connects G_i with G_0, and let u be the cutvertex of G_0 that was incident to bridge b. We place b along u's row (right or left), and then we place the rest of the drawing of G_i (rotated appropriately). Note that G_i was already drawn with the vertex incident to bridge b as the final vertex. Both this vertex and u are placed in the same row (thus we have a row pair), so we only need to reuse two more rows. This example is illustrated in Fig. 6c, where G_i (whose drawing has three rows) is reusing three rows.

From the discussion above (especially Lemma 7 and Theorem 6) it follows that each biconnected component of G can fit within a grid of size $(n-1)\times(n-1)$ with $2n-1$ bends, when the component is drawn using our algorithm and with some vertex of maximum degree three as final vertex. Using a simple induction (similar to the one in [3]) we can show that the drawing of one-connected graph G requires area at most $n \times n$ and no more than $2n+2$ bends. We will not provide further arguments on the number of bends. However, since each component of G has area at most $0.76n^2$ (because of pairing within the component), our target

is to show that the same upper bound holds for one-connected graph G. It turns out that in order for the induction to yield this upper bound, we have to reuse at least three rows and/or columns, as discussed above. This is formalized in the following invariant, which we maintain during the drawing process:

Invariant: Let G be a graph of maximum degree 4. We draw G using our approach and with some vertex of maximum degree 3 as the final vertex. Let K_1 and K_2 be all the savings we have along the x and the y axis respectively (i.e., columns and rows saved). It holds that: $K_1 + K_2 \geq \lceil \frac{n-4}{4} \rceil - 2$, $K_1 \geq 0$ and $K_2 \geq 0$. Additionally, the drawing of G requires at most $(n-1-K_1) \times (n-1-K_2)$ area.

Notice that when the G_i's are attached to G_0's appropriate places, they are typically rotated by $\frac{\pi}{2}$, π, or $\frac{3\pi}{2}$. The rotation of the G_i's does not affect the sum of $K_1 + K_2$, since a row saving will become a column saving and vice versa as a result of the rotation. We have described a technique that reuses at least three rows and/or columns, and that was sufficient in order to prove our bound for $K_1 + K_2$. However, a clever implementation of our algorithm will reuse as many rows and columns as possible, when subgraphs G_i merge with component G_0, yielding better bounds for $K_1 + K_2$ and thus drawings with even smaller area.

Theorem 8. *There is a linear time algorithm which constructs an orthogonal drawing of an n-vertex graph of maximum degree 4 with area no more than $0.76n^2$. The total number of bends is at most $2n+2$, and no edge has more than 2 bends.*

Consider the example of Fig. 6d: each subgraph G_i is a simple triangle (we have a total of seven triangles), G_0 is a simple cycle, and the total number of vertices is $n = 21$. Drawing this graph using our technique, returns the drawing of Fig. 6d whose area is just $0.08n^2$; this result is mostly due to the very small size of these subgraphs, and not to the pairing process.

4 Drawing Maximum Degree 3 Graphs

In [14] we presented an algorithm for obtaining an orthogonal drawing of any biconnected graph (not necessarily planar) of maximum degree 3. This algorithm matched the upper bounds on the area and number of bends for planar graphs. In this paper, we enrich this algorithm with some more features so that it can draw maximum degree 3 graphs which are only one-connected. The new algorithm is called 3_ORTHOGONAL (explained in detail in [13]), and achieves bounds that are identical to the bounds in [14] in terms of area. Also, for biconnected graphs of maximum degree 3, the bounds of Algorithm 3_ORTHOGONAL in [13] are identical to the bounds in [14] in terms of the number of bends. More precisely, we have that:

Theorem 9. *[14] There exists a linear time algorithm for constructing an orthogonal drawing of a biconnected graph of maximum degree 3 with at most $\lfloor \frac{n}{2} \rfloor + 3$ bends. There is at most one edge that bends twice, while every other*

edge in the drawing has at most one bend. Moreover, the drawing requires at most $\frac{1}{4}n^2$ *area.*

For graphs of maximum degree 3, Storer [17] has shown a family of graphs that require at least $\frac{n}{2}+1$ bends under any orthogonal drawing. From this it follows that Algorithm 3_ORTHOGONAL for biconnected graphs [14, 13] is optimal in terms of the number of bends, within an additive constant of two.

Let us assume that we have a general n-vertex graph G of maximum degree 3, which is connected but not biconnected. In this case, we break G into its biconnected components and derive its block tree. Notice that for a graph with maximum degree 3, this can be done in linear time. Then, we obtain a global numbering for the vertices of G from 1 to n. This numbering implies an orientation of the edges of G in the same way as an st-numbering implies an orientation of the edges of a biconnected graph. In [13], we give a complete and detailed description of how Algorithm 3_ORTHOGONAL works. Due to space limitations, here we give only the main result.

Theorem 10. *There exists a linear time algorithm for constructing an orthogonal drawing of a general graph of maximum degree 3 with at most* $\lfloor \frac{n}{2}+2l+1 \rfloor$ *bends, where* l *is the number of the biconnected components of* G *that are leaves in* G*'s block tree. Also, there is at most one edge that bends twice, while every other edge in the drawing has at most one bend. Moreover, the drawing requires at most* $\frac{1}{4}n^2$ *area.*

5 Open Problems

- Is it possible to obtain better area upper bounds than $0.76n^2$, for graphs of maximum degree 4?
- Develop algorithms to produce orthogonal drawings of graphs of arbitrary degree, and establish bounds on the area and number of bends. An issue that has to be studied here is the way that the vertices are represented.
- Develop interactive graph drawing algorithms. Some work for graphs of maximum degree four appears in [15].

References

1. Therese Biedl, "Embedding Nonplanar Graphs in the Rectangular Grid", *Rutcor Research Report* 27-93, 1993.
2. Therese Biedl, "New Lower Bounds for Orthogonal Graph Drawings", *Proc. of GD '95, Lecture Notes in Comp. Sci.*, 1027, Springer-Verlag, 1995, pp. 28-39.
3. T. Biedl and G. Kant, "A Better Heuristic for Orthogonal Graph Drawings", *Technical Report, Utrecht Univ., Dept. of Comp. Sci.*, UU-CS-1995-04. Prelim. version appeared in *Proc. 2nd Ann. European Symposium on Algorithms (ESA '94), Lecture Notes in Computer Science*, vol. 855, pp. 24-35, Springer-Verlag, 1994.

4. G. DiBattista, P. Eades, R. Tamassia and I. Tollis, "Algorithms for Drawing Graphs: An Annotated Bibliography", *Computational Geometry: Theory and Applications*, vol. 4, no 5, 1994, pp. 235-282. Also available via anonymous ftp from `ftp.cs.brown.edu`, `gdbiblio.tex.Z` and `gdbiblio.ps.Z` in `/pub/papers/compgeo`.
5. G. DiBattista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari and F. Vargiu, "An Experimental Comparison of Three Graph Drawing Algorithms", *Proc. of ACM Symp. on Computational Geometry*, pp. 306-315, 1995. The version of the paper with the four algorithms can be obtained from `http://www.cs.brown/people/rt`.
6. G. DiBattista, G. Liotta and F. Vargiu, "Spirality of orthogonal representations and optimal drawings of series-parallel graphs and 3-planar graphs," *Proc. Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science 709*, Springer-Verlag, 1993, pp. 151-162.
7. S. Even and G. Granot, "Rectilinear Planar Drawings with Few Bends in Each Edge", *Tech. Report 797, Comp. Science Dept.*, Technion, Israel Inst. of Tech., 1994.
8. S. Even and R.E. Tarjan, "Computing an st-numbering", *Theor. Comp. Sci.* 2 (1976), pp. 339-344.
9. A. Garg and R. Tamassia, "On the Computational Complexity of Upward and Rectilinear Planarity Testing", *Proc. DIMACS Workshop GD '94, Lecture Notes in Comp. Sci. 894*, Springer-Verlag, 1994, pp. 286-297.
10. Goos Kant, "Drawing planar graphs using the lmc-ordering", *Proc. 33th Ann. IEEE Symp. on Found. of Comp. Science*, 1992, pp. 101-110.
11. F. T. Leighton, "New lower bound techniques for VLSI", *Proc. 22nd Ann. IEEE Symp. on Found. of Comp. Science*, 1981, pp. 1-12.
12. Charles E. Leiserson, "Area-Efficient Graph Layouts (for VLSI)", *Proc. 21st Ann. IEEE Symp. on Found. of Comp. Science*, 1980, pp. 270-281.
13. A. Papakostas and I. G. Tollis, "Algorithms for Area-Efficient Orthogonal Drawings", *Tech. Report UTDCS-06-95*, The University of Texas at Dallas, 1995. Also available on the WWW at http://wwwpub.utdallas.edu/~tollis.
14. A. Papakostas and I. G. Tollis, "Improved Algorithms and Bounds for Orthogonal Drawings", *Proc. DIMACS Workshop GD '94, Lecture Notes in Comp. Sci. 894*, Springer-Verlag, 1994, pp. 40-51.
15. A. Papakostas and I. G. Tollis, "Issues in Interactive Orthogonal Graph Drawing", *Proc. of GD '95, Lecture Notes in Comp. Sci. 1027*, Springer-Verlag, 1995, pp. 419-430.
16. Markus Schäffter, "Drawing Graphs on Rectangular Grids", *Discr. Appl. Math.* 63 (1995), pp. 75-89.
17. J. Storer, "On minimal node-cost planar embeddings", *Networks* 14 (1984), pp. 181-212.
18. R. Tamassia, "On embedding a graph in the grid with the minimum number of bends", *SIAM J. Comput.* 16 (1987), pp. 421-444.
19. R. Tamassia and I. Tollis, "Planar Grid Embeddings in Linear Time", *IEEE Trans. on Circuits and Systems* CAS-36 (1989), pp. 1230-1234.
20. R. Tamassia, I. Tollis and J. Vitter, "Lower Bounds for Planar Orthogonal Drawings of Graphs", *Information Processing Letters* 39 (1991), pp. 35-40.
21. L. Valiant, "Universality Considerations in VLSI Circuits", *IEEE Trans. on Comp.*, vol. C-30, no 2, (1981), pp. 135-140.