A Fast Heuristic for Hierarchical Manhattan Layout

G. Sander (sander@cs.uni-sb.de)

Universität des Saarlandes, FB 14 Informatik, 66041 Saarbrücken

Abstract. A fast heuristic for the layout of directed graphs according to Manhattan convention is presented. Nodes are placed into layers. Edges consist of sequences of vertical and horizontal segments. Sharing of segments is allowed in certain situations. The algorithm is an extension of the hierarchical layout method [11, 15] that includes crossing reduction and emphasis on a uniform edge orientation. Compared to the original algorithm, the time overhead is $O(n+ek)$ where n, e and k are the number of nodes, of edges, and the maximal number of line rows between two layers of nodes. It produces drawings where each edge has at most four bends.

1 Introduction

We address the problem of constructing drawings of graphs from the viewpoint of compiler construction where these drawings are used for demonstration, debugging and documentation of algorithms and data structures. Typically, these data structures are large and dense, while the construction of the drawing should be fast enough to support online debugging and algorithm animation.

In Manhattan drawings all edges consist of sequences of line segments which have a strict horizontal or vertical orientation. Manhattan layout (also called orthogonal layout) is widely used in VLSI design [1, 16]. However, the focus of VLSI design is different to our applications: First, in VLSI design, normally the nodes (terminals) have a fixed place, while only the edges (wires) need to be routed. Secondly, the cost, correctness and usability of the VLSI circuit (e.g., the minimizing of wire length [3]) has much more importance than the speed of the layout process. Typical VLSI routing times for the wires range up to 1h, while in interactive data structure visualization, the user should never wait more than a minute for the positioning of both nodes and edges.

There are good solutions for subclasses of graphs, mostly for grid embeddings. Vijayan and Wigderson [14] discuss the straight line embedding of graphs with n nodes in a square grid without edge crossings in time complexity $O(n^2)$. They also give a classification, which planar graphs can be handled. Tamassia's algorithm [12] allows edge bendings, thus it is suitable for all planar graphs with nodes of maximal degree 4. It runs in $O(n^2 log\, n)$ and produces a layout with minimal number of bends. A linear algorithm for 4-ary planar graphs is presented by Tamassia and Tollis [13], where the number of bends is not minimal but restricted by the upper bound $2.4n+2$. Biedl and Kant [4] describe a linear time method to draw an undirected graph without loops or multiple edges, but with maximal node degree 4, in an $n \times n$ area with at most $2n + 2$ bends. This method is based on st-numbering. Nodes are drawn as points, i.e. have size zero. Even and Granot [5] present a similar approach for the case that nodes are structures of nonzero size. Orthogonal layouts for general graphs can be obtained by the layout method of Batini et al. [2]. Here, the graph is embedded in an orthogonal grid according to several aesthetics like minimization of edge crossings, of edge bends, of edge lengths, of the area, and placement of specified nodes on the external boundary. Since some of the aesthetics are contradicting, and some of the subproblems in this algorithm are NP-hard (e.g., the planarization problem to minimize edge crossings [8]), the layout method is a heuristic. This approach is perfectly suitable for undirected graphs. It is also applied to directed graphs, however it does not emphasize a uniform edge orientation. Protsko et al. [9] present a similar approach. They further sketch the idea how the uniform edge orientation can be specified by additional constraints.

Our preconditions are very special: We have large directed graphs (not necessarily planar or 4-ary). Nodes have nonzero size and different shapes. The layout should emphasize a uniform edge orientation and allow the grouping of nodes into the levels of a hierarchy. This implies a strong restriction of the y-position of the nodes to discrete coordinates, but there are only weak constraints to the x-position of the nodes. The layout algorithm should be very fast. Reducing the number of edge crossings, the number of bends in edges or the area of the drawing is useful, but has not the first priority.

Why do we prefer orthogonal edge segments: Manhattan layout gives a very uniform aesthetics which is in particular of interest for annotated control flow graphs (Fig. 1). Most of hand drawn control flow diagrams in the literature use Manhattan convention, thus programmers are familiar with this aesthetics. Edges tend to be bundled like busses such that it is easy to follow chains of long edges. The visual resolution is very high even if the graphs are very dense.

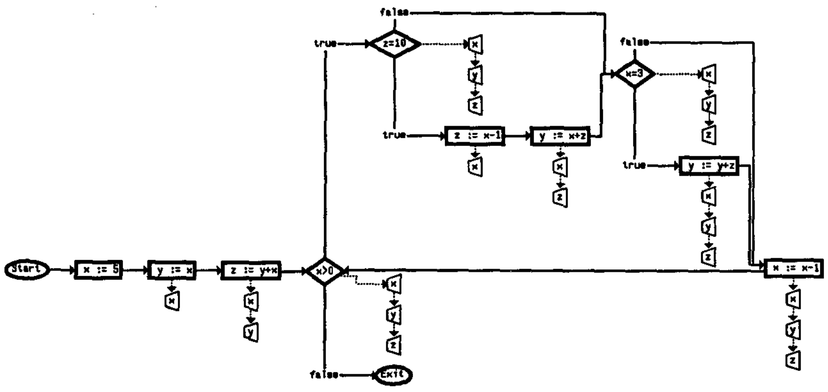

Fig. 1. Annotated CFG by VCG Manhattan Layout

Our algorithm is implemented in the VCG tool [10]. It is an extension of our variant of the hierarchical layout methods of Warfield [15] and Sugiyama et al. [11] which are known to be suitable for large directed graphs. It runs fast: we show that the overhead of Manhattan layout compared to normal hierarchical layout is $O(n + ek)$ where n is the number of nodes, e is the number of edges in the proper hierarchy, and k is the maximal number of line rows between two layers of nodes. Note that k is very small for most layouts. It produces drawings where each edge has maximally four bends.

In the next section, we explain the drawing rules for our variant of the Manhattan layout. Section 3 shortly sketches well known results about edge crossing reduction. Section 4 presents the segment ordering graph needed for the calculation of node positions. Section 5 deals with the adjustment of positions to a grid. Then, we describe the algorithm for the routing of edges, and finally, we show some statistics and experiences.

2 Manhattan Drawing Conventions

We define the drawing conventions of our applications:

1) Nodes are placed in a hierarchy of layers and never overlap.
2) Edges are drawn as a sequence of horizontal or vertical line segments.
3) An edge segment never crosses a node.
4) A horizontal segment of one edge and a vertical segment of another edge may share a point (i.e., we have an edge crossing, Fig. 3, Sit. b).
5a) Two horizontal segments of different edges never share a point.
6) Vertical segments of different edges may share subsegments, if and only if the segments are adjacent to a node (i.e., they must be the first or the last segment of the edge, Fig. 2, Sit. b).
7) Horizontal segments should have a minimal vertical distance d_h. Vertical segments should have a minimal horizontal distance d_v.

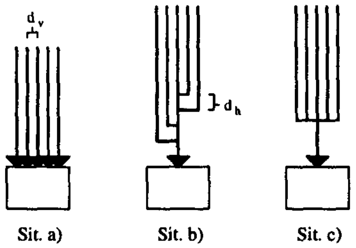

Fig. 2. Diff. Segment Sharing Conv.

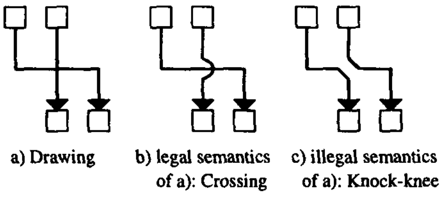
a) Drawing
b) legal semantics of a): Crossing
c) illegal semantics of a): Knock-knee

Fig. 3. Crossing and Knock-knee

In a sequence of edge segments belonging to the same edge two adjacent segments always share an endpoint. Convention 4 allows edge crossings, thus the convention is applicable for nonplanar graphs, but the aesthetics requires that the number of edge crossings should be minimized. Convention 5a and 6 contribute to the visual readability: The route of two different edges is clearly distinguishable if they never share common equally-oriented segments (Fig. 2, Sit. a). However, we relax this constraint and allow the sharing on the first and the last segment of edges. The reason: If the last segments of edges are shared, they also share the arrowhead, which increases the resolution (Fig. 2, Sit. b). Of course, an edge segment with arrowhead never shares an edge segment without arrowhead. Otherwise, it would not be recognizable which segments have arrowheads. In particular, convention 5a forbids knock-knees which would not be distinguishable from edge crossings (Fig. 3).

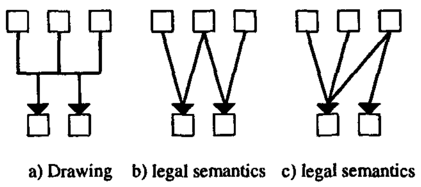

Fig. 4. Confusing Sit. with Conv. 5b

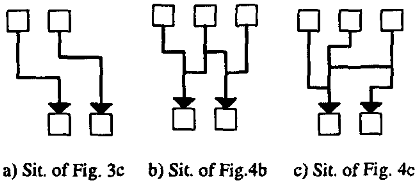

Fig. 5. Drawings with Conv. 5a

In tree layouts, convention 5a is often relaxed (see later Fig. 10):

5b) Two horizontal segments of different edges may share a subsegment, if they also share the vertical segment adjacent to the subsegment (Fig. 2, Sit. c).

However in general graphs, drawings according to convention 5b are sometimes misleading. For instance, the drawing Fig. 4a can represent the semantics sketched by Fig. 4b and Fig. 4c. Note that the corresponding drawings according to convention 5a are more readable (Fig. 5).

3 Hierarchy Mapping

Definition 1 *A* **proper** n**-level hierarchy** *is a directed graph* $G = (V, E)$ *which satisfies the following conditions:*

- *The set of nodes* V *is partitioned into* n *disjoint sequences* $V_1, \ldots, V_n$.
- *The set of edges* E *is partitioned into* $n-1$ *disjoint subsets* $E_1, \ldots, E_{n-1}$ *with* $E_i \subseteq V_i \times V_{i+1}$.

Furthermore $\mathbf{lev}(v) = i$ *iff* $v \in V_i$. *We call* V_i *the* **layer** i. *We write* $\mathbf{pos}(v) = k$ *if* v *is the* k*th node of a layer, i.e. the* k*th node in a sequence* V_i.

Methods to convert an arbitrary directed graph into a proper hierarchy by introducing dummy nodes are discussed in [15, 11, 6], and [10]. We use the notions predecessor $\mathbf{pred}(v)$ of a node v, successor $\mathbf{succ}(v)$, $\mathbf{indeg}(v) = |\mathbf{pred}(v)|$, $\mathbf{outdeg}(v) = |\mathbf{succ}(v)|$ with respect to the proper hierarchy. Note: a predecessor of v in a proper hierarchy need not to be a predecessor of v in the original graph. For instance, an edge of the original graph may occur reverted in the proper hierarchy.

Definition 2 *A* **linear segment** *S is a maximal sequence of nodes $w_1, \ldots, w_m$ with*
(1) $\mathbf{lev}(w_i) = \mathbf{lev}(w_{i+1}) - 1$
(2) $\mathbf{indeg}(w_1) \leq 1, \mathbf{outdeg}(w_m) \leq 1$, $\mathbf{succ}(w_1) = \{w_2\}$, $\mathbf{pred}(w_m) = \{w_{m-1}\}$
(3) $\mathbf{pred}(w_i) = \{w_{i-1}\}$ *and* $\mathbf{succ}(w_i) = \{w_{i+1}\}$ *for* $i \in \{2, \ldots, m-1\}$.

A linear segment is a sequence of nodes that should be drawn as a straight vertical line (Fig. 6). Each node belongs to at most one linear segment. We can easily partition the proper hierarchy into disjoint linear segments and remaining nodes in time $O(|V| + |E|)$.

Edge crossing reduction in the proper hierarchy is done as follows: We traverse the hierarchy from layer 1 to layer n; for each layer, we calculate a weight W_p for each node and change the positions $\mathbf{pos}(v)$ by sorting the nodes of the layer according to this weight. Then, we traverse the hierarchy from layer n to layer 1 and reorder according to a weight W_s. This is iterated several times [11].

Proposition 1 *Assume that*
(1) $(\forall (v', w') \in \mathbf{pred}(v) \times \mathbf{pred}(w) \;\; \mathbf{pos}(v') < \mathbf{pos}(w')) \;\Rightarrow\; W_p(v) < W_p(w)$
(2) $(\forall (v', w') \in \mathbf{succ}(v) \times \mathbf{succ}(w) \;\; \mathbf{pos}(v') < \mathbf{pos}(w')) \;\Rightarrow\; W_s(v) < W_s(w)$
Then after a crossing reduction traversal, for two linear segments $S_a \neq S_b$ $\nexists v_i, v_{i+1} \in S_a, w_j, w_{j+1} \in S_b$ *with* $\mathbf{lev}(v_i) + 1 = \mathbf{lev}(w_j) + 1 = \mathbf{lev}(v_{i+1}) = \mathbf{lev}(w_{j+1})$ *and* $\mathbf{pos}(v_i) < \mathbf{pos}(w_j) \wedge \mathbf{pos}(v_{i+1}) > \mathbf{pos}(w_{j+1})$

Proposition 1 implies that after crossing reduction, two linear segments will never cross. Assume that the proposition does not hold, i.e. that such v_i, v_{i+1}, w_j, w_{j+1} would exist. Obviously, v_{i+1} and w_{j+1} would be reordered in a top down traversal of the layers, or v_i and w_j would be reordered in a bottom up traversal, respectively. Good selections of weights W_p and W_s that satisfy assumptions (1) and (2) can be found in [11] and [6].

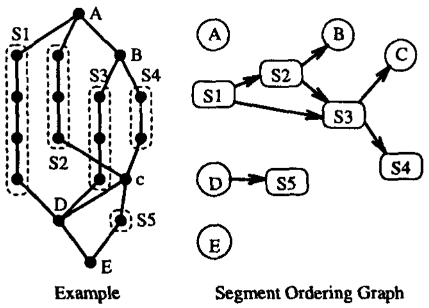

Fig. 6. Linear Segments

4 Positioning of Nodes

Now, we try to find positions of nodes such that (a) layers 1 to n are positioned top down, (b) nodes with **pos** $= 1 \ldots m$ are positioned left to right, (c) all nodes of a layer are on a horizontal line, (d) the layout is balanced, (e) all nodes of a linear segment are on a vertical line. In particular, this implies that the edge segments and dummy nodes of a linear segment that formerly represented an edge are now drawn in a long vertical line without bends. First, we calculate the x-coordinates of the nodes. To simplify the explanation, each node v that does not belong to a linear segment forms a trivial linear segment with one element $\{v\}$. This implies that the whole graph is now partitioned into disjoint linear segments. The linear segments are ordered according to the relation *'is left of'* ($\sqsubseteq$) by using the segment ordering graph SG (Fig. 6):

```
(1) Nodes( $SG$ ) := all linear segments; Edges( $SG$ ) := $\emptyset$
(2) for each layer $V_i$ do
(3)     for $j$ := 2 to $|V_i|$ do
(4)         let $v_{j-1} \in S_a$ and $v_j \in S_b$
(5)         /* it holds $\mathbf{lev}(v_{j-1}) = \mathbf{lev}(v_j)$ and $\mathbf{pos}(v_{j-1}) = \mathbf{pos}(v_j) - 1$ */
(6)         Edges( $SG$ ) := $\{(S_a, S_b)\} \cup$ Edges( $SG$ )
(7)     od
(8) od
(9) topological_sort( $SG$ )
```

SG is acyclic because of proposition 1. Thus topological sorting is possible, and each linear segment S has an ordering number **spos**(S) afterwards. The calculation of SG needs time $O(|V|)$ if the layers are represented by ordered, doubly linked lists. Topological sorting needs time $O(|SG|)$, but because the number of linear segments does not exceed the number of nodes, and each node introduces only one edge in SG, the whole calculation of ordering numbers can be done in time $O(|V|)$. Now, we can produce an initial position $x(v)$ for each node that satisfies (b) and (e):

```
(10) for all $S$ in increasing order of spos do
(11)     for all $v \in S$ do
(12)         $x_{min}(v)$ := leftmost possible x-coordinate
(13)     od
(14)     $X_{min}$ := max $\{x_{min}(v) \mid v \in S\}$
(15)     for all $v \in S$ do $x(v) := X_{min}$ od
(16) od
```

The leftmost possible x-coordinate must be a position right to the segments that are already placed. This initial position is not yet balanced. We balance the graph by using a variant of the pendulum method [10]. However, different from [10], the movable entities are not the nodes but the linear segments, thus linear segments remain straight vertical lines. We sketch the pendulum method again: The linear segments are the ball and the edges between the linear segments are the strings of the pendulum. If the uppermost segments are fixed on a ceiling, the balls on the strings swing to a balanced layout driven by their gravity. Balls attached by several strings swing into the middle position of their predecessor balls. Neighbored balls may influence each other. For instance, if the left ball is pulled to the right and the right ball is pulled to the left, the balls form a region which is positioned such that the sum of pulling forces of the region becomes zero. We simulate this simplified physical model.

Definition 3 *The* **predecessor deflection** *of an edge $e = (s, t)$, of a node v, of a linear segment S and of a region of linear segments $\{S_1, \dots, S_k\}$ is defined as*

$$\mathbf{D}_p(e) = x(s) - x(t) \qquad \mathbf{D}_p(v) = \frac{\sum_{(w,v)\in E} \mathbf{D}_p((w,v))}{\mathbf{indeg}(v)}$$

$$\mathbf{D}_p(S) = \sum_{v\in S} \mathbf{D}_p(v) \qquad \mathbf{D}_p(\{S_1, \dots, S_k\}) = \frac{\sum_{i\in\{1,\dots,k\}} \mathbf{D}_p(S_i)}{k}$$

The **successor deflection** $\mathbf{D}_s$ *can be defined similarly.*
Two segments S_a and S_b are **touching** *if there is a node $v \in S_a$ and a directly neighbored node $w \in S_b$ in the same layer and the distance between both nodes is the minimal allowed distance.*

If $\mathbf{D}_p(S) > 0$, the segment is pulled to the right, and if $\mathbf{D}_p(S) < 0$, it is pulled to the left. Because each linear segment S is always positioned as a straight vertical line, $\mathbf{D}_p(S) = \mathbf{D}_p(v_1)$ holds where v_1 is the first (topmost) node of S, because $\mathbf{D}_p(v_i) = 0$ for $v_i \in S$ with $v_i \neq v_1$. We start with a trivial partition PR of segments into regions $R_1 \dots, R_m$ and subsequently replace regions that influence each other by their union.

```
(17) $PR := \{R_i \mid R_i = \{S_i\}$ for each $S_i\}$
(18) repeat
(19)     if $S_a \sqsubseteq S_b$ are touching then let $S_a \in R_a$ and $S_b \in R_b$
(20)         if $\mathbf{D}_p(R_a) \geq \mathbf{D}_p(R_b)$ then $PR := PR +$ union( $R_a, R_b$ ) $-R_a - R_b$ fi
(21)     fi
(22) until the regions don't change anymore.
```

Finally, we try to correct the position of each segment S of a region R by moving all nodes of S horizontally by the minimum of $|\mathbf{D}_p(R)|$ and the space between S and its neighboring segments. We move to the left if $\mathbf{D}_p(R) < 0$, otherwise to the right. In this way, the nodes never overlap. The whole process is repeated iteratively with the predecessor deflection $\mathbf{D}_p$ and the successor deflection $\mathbf{D}_s$ until the layout is balanced.

The time overhead of this algorithm with respect to the original algorithm in [10] is constant:[1] The lines (19)-(21) are in fact identical to the original algorithm.

[1] This holds for the worst case (this is, if we have $n - 1$ unions for all n nodes of the graph, i.e. if the result is one large region containing all nodes/segments). For the normal case, a comparison of the complexity is difficult: Working on segments instead of nodes might require more unions of regions, i.e. more iterations of (18)-(22). On the other hand, our experience is, that the pendulum method comes faster to a balanced situation, if it works on segments instead of nodes.

We store for each node, which segment it belongs to, and for each segment, which region it belongs to. In order to check whether two segments are touching, we check whether two nodes are touching (we did this in the original algorithm, too) and get the segment and the region from the nodes in constant time (this is the overhead).

5 Adjustment to the Horizontal Grid

Before we calculate the relative positions of the edges, which will also give the positions $y(v)$, we adjust the positions $x(v)$ to a grid with raster d_v: We traverse the segments in increasing order of **spos** (i.e. from the leftmost to the rightmost segment) and move each segment to the nearest possible grid point. This might reduce the balance by a small degree, but it ensures that vertical edge segments will have a minimal distance d_v.

Edges of the proper hierarchy connecting nodes of the same linear segment can be drawn as straight vertical lines. Edges connecting nodes of different linear segments have two potential bend points. Since an original edge of the initial directed graph may consist of a linear segment of dummy nodes and two edges that connect the linear segment with the remaining graph, the maximal number of bends of the original edge is four. In this section, we calculate the end points of the edges. In the next section, we derive the bend points from the end points.

The end points of the edges depend on the port points where the edge is adjacent to the node. Ports are points at the border of the node. A node v needs at most **indeg**(v) incoming ports and **outdeg**(v) outgoing ports. The calculation of the incoming ports starts with the ordered sequence $SP(v)$ of trivial ports and subsequently unifies neighboring ports if all end points of the corresponding edges have the same arrowhead style. The arrowhead style indicates the existence of an arrowhead, it might further indicate different colors, sizes or drawing modes of arrowheads. Edges of the same port share a vertical edge segment.

```
(23) $SP(v) := (P_1(v), \ldots, P_{indeg(v)}(v))$ with $P_j(v) = \{(w, v) \in E \mid \mathbf{pos}(w) = j\}$
(24) for $j$ from 2 to indeg(v) do   /* $SP(v) = (\ldots, P_{j-1}(v), P_j(v), \ldots)$ */
(25)     if for some $e_a \in P_{j-1}(v), e_b \in P_j(v)$ is $\text{ahead\_style}_{in}(e_a) = \text{ahead\_style}_{in}(e_b)$
(26)         then $SP(v) := (\ldots, P_{j-1}(v) \cup P_j(v), \ldots)$
(27)     fi
(28) od
```

Note that all edges in $P(v)$ always have the same arrowhead style. Thus in line (25), we need only one element $e_a \in P_{j-1}(v)$ and another element $e_b \in P_j(v)$ in order to check whether for all $e \in P_{j-1}(v), e' \in P_j(v)$ holds $\text{ahead\_style}_{in}(e) = \text{ahead\_style}_{in}(e')$. This algorithm needs time $O(\mathbf{indeg}(v))$. The calculation of the outgoing ports is symmetrical. It is done for all nodes. Thus, we get all ports of the whole graph in time $O(|V| + |E|)$. Note: Since the predecessor nodes w are already sorted according to **pos**(w) after the crossing reduction, the creation of the ordered sequence $SP(v)$ is possible without sorting the $P_j(v)$. Traverse the predecessor nodes s in that order, create for each outgoing edge (s, t) a set

$P = \{(s,t)\}$ and append P to the sequence of ports $SP(t)$. In this way, the sequence of ports will automatically be ordered.

The ports are now assigned to x-coordinates. The corresponding end points of all edges of a port get the same x-coordinate. The x-coordinate of a port must be on the grid with raster d_v. If a node needs more ports than available on its border, this can be corrected by reducing the raster d_v or by increasing the size of the node. However, both solutions require a recalculation of the previous steps. For simplicity, our current solution combines ports in this case until the number of ports fits the number of available raster points on the border of the node. Unfortunately, this fallback solution decreases the readability, but these cases are rare: they occur only if the degree of a node is very high, there are nearly as much backward edges as forward edges, and the schedule of forward and backward edges at the node happens to be very unfavorable.

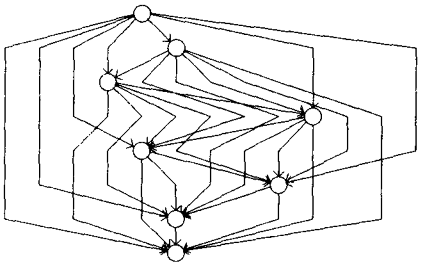
Fig. 7. K8 by Original Sugiyama Layout

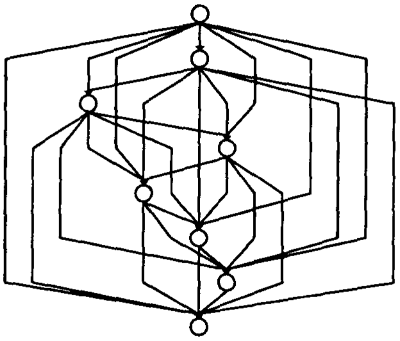
Fig. 8. K8 by Nearly Manhattan Layout

Is is important to note that until this step the layout process is not strictly related to Manhattan layout. The algorithm enforces long edges to be drawn straight vertical, which is a necessary precondition for the Manhattan convention. But even without the following final step, the algorithm would produce a nice, balanced drawing, that produces more straight vertical lines than the original algorithm of Sugiyama et al. [11] (compare Fig.7, made by GraphEd [7], with Fig.8. Figure 7 contains much more zig-zags).

6 Positioning of Edges

We have the following situation: Each node v of the proper hierarchy has a position $x(v)$ and a level $\mathbf{lev}(v)$ but not yet a position $y(v)$. Each edge e between two layers has a start position $x_s(e)$ and an end position $x_e(e)$ derived from the positions of the incoming and outgoing ports e belongs to. The start position of e is in the upper layer and the end position in the lower layer. Next, the free vertical space between layers V_i and V_{i+1} is divided into k_i line rows each containing horizontal line segments (Fig. 9). The rows have the vertical distance d_h. Thus, the distance between two layers must be $(k_i+1)\ d_h$. It holds $k_i \leq |E_i|$, but in order to avoid wasting of vertical space, we calculate a minimal k_i. From the k_i the y-coordinates of the nodes and edges can be derived directly.

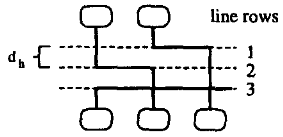

Fig. 9. Line Rows

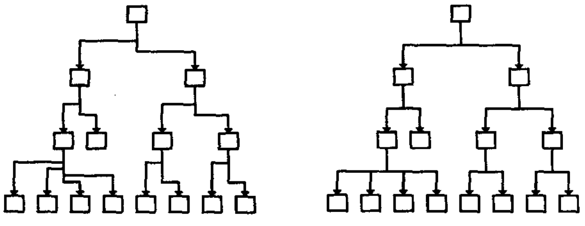

Fig. 10. Tree by VCG Manhattan Layout

Each edge e with $x_s(e) \neq x_e(e)$ is drawn by two straight vertical segments and a connecting horizontal segment that belongs to line row $r(e)$. We calculate the line rows $r(e)$ of the edges between layers V_i and V_{i+1} by a plain sweep algorithm: We traverse each E_i of the proper hierarchy by a sweep line in increasing order of the x-coordinate. When the sweep line touches an edge e the first time, the edge is added to the set of unfinished edges U and gets its $r(e)$. The horizontal line segment of e is in conflict only with those edges which are in U at that time. In order to fulfill the drawing convention 5a, $r(e)$ must be a number that is currently not used by the edges in U. When the sweep line touches an edge the last time, the edge is removed from U. The value k_i is the maximal $r(e)$. The algorithm requires a list C of pairs (e, x), sorted according to x. For each edge $e \in E_i$, there are two pairs $(e, x_s(e))$ and $(e, x_e(e))$. Similar as the ordered sequence of port sets, the list C can be constructed in time $O(|V|+|E|)$ without the call of a sorting procedure, since the nodes of the layers are already sorted after crossing reduction and the edges are already sorted after the port calculation.

```
(29) $U := \emptyset$;  $k_i := 0$
(30) for each $(e, x) \in C$ in increasing order do
(31)     if $x_s(e) \neq x_e(e)$ then
(32)         if $x = min\{x_s(e), x_e(e)\}$ then
(33)             $r(e) := 1 + max\{r(e')|e' \in U\}$
(34)             if $k_i < r(e)$ then $k_i = r(e)$ fi
(35)             $U := U + e$
(36)         else/* $x = max\{x_s(e), x_e(e)\}$ */
(37)             $U := U - e$
(38)         fi
(39)     fi
(40) od
```

The algorithm can easily be adapted to drawing convention 5b by a modification of line (33). Further, with some time overhead, line (33) can be changed to search for smaller values for $r(e)$ that currently are not in U. This avoids wasting of vertical space. The loop (30)-(40) is executed $2|E_i|$ times. We implement U as doubly linked linear list, i.e. each edge e has a reference to its occurrence in U. Thus, insertion and deletion can be done in constant time. The maximal $r(e)$ in U is found in time $O(k_i)$, i.e. the whole algorithm needs time $O(|V| + |E_i|k_i)$.

At last, the y-coordinates of nodes and edges are derived. For nodes $v \in V_0$ we set $y(v) = 0$. For nodes $v \in V_i$ we set $y(v) = (k_i + 1) * d_h + max\{y(w) + height(w) \mid w \in V_{i-1}\}$. Assume that for instance the shape of the nodes are boxes. Then, for edges $e = (s,t)$ we set $y_s(e) = y(s) + height(s)$ and $y_e(e) = y(t)$. For edges e with $x_s(e) \neq x_e(e)$ we set further the two bend points $(x_s(e), y_s(e) + r(e)d_h)$ and $(x_e(e), y_s(e) + r(e)d_h)$. This completes the calculation of the Manhattan layout.

7 Experiences

Table 1 shows the time and space of some examples laid out by the normal algorithm [15, 11, 10] and by our extended Manhattan layout algorithm. It is difficult to analyze the time complexity of the algorithms. It depends heavily on the number of iterations made during crossing reduction and node positioning. However, it is known that the normal algorithm is very fast even for large graphs. Compared to it, we need additional time $O(|V| + |E| \max\{k_i\})$ in the worst case. In some cases (graph 2), Manhattan layout is even faster than normal layout, because less iteration are needed for node positioning.

Example	*Nodes*	*Edges*	*Crossings*	t_{norm}	W_{norm}	t_{manh}	W_{manh}	*Bendings*$_{\text{manh}}$
Graph 1	33	38	0	2.3	490	2.4	620	16
Graph 2	20	190	2251	14.3	960	12.7	1370	628
Graph 3	40	131	585	3.1	1780	3.3	1830	418
Graph 4	50	151	190	5.8	270	6.1	340	474
Graph 5	615	1310	15640	61.5	15420	65.0	26970	2261

Times (sum of user time and system time in secs) for parsing, layout and drawing (Sun Sparc 10/30, 32 MB mem., X11R6). $t_{\text{norm}}/W_{\text{norm}}$ are time/width of normal layout [10], and $t_{\text{manh}}/W_{\text{manh}}$ are time/width of Manhattan layout. The number of crossings is independent of the layout method, because in both cases the same crossing reduction method is used. The number of bendings is measured for Manhattan layout.

Table 1. Statistics

In dense graphs (graph 2, 3, 4) the number of bend points tends to the maximal number $4|E|$. Manhattan layout needs more space (W of graph 1, 2, 5) because long linear segments block close node positions. With very long linear segments the plane tend to be separated into a grid with large meshes. Most nodes are placed on the grid points while only few nodes are inside the meshes. This might be not satisfying because of the waste of space. In this case, the reduction of bend points of edges has minor importance than the usage of space. As solution the size of the linear segments can be restricted. In this variant of our algorithm edges may have more than four bends but the nodes are closer together.

8 Acknowledgment

We like to thank R. Wilhelm, C. Fecht, and R. Heckmann for their comments on the presentation of the algorithm.

References

1. Baker, B.S.; Bhatt, S.N.; Leighton, F.T.: An Approximation Algorithm for Manhattan Routing, *in* Preparata, F.P., ed.: Advances in Computing Research, Vol. 2, pp. 205-229, JAI Press, Greenwich, Connecticut, 1984.
2. Batini, C.; Nardelli, E.; Tamassia, R.: A Layout Algorithm for Data Flow Diagrams, IEEE Trans. on Software Engineering, SE-12(4), pp. 538-546, 1986,
3. Bhatt, S.N.; Cosmadakis, S.S.: The Complexity of Minimizing Wire Lengths in VLSI Layouts, Information Processing Letters, 25, pp. 263-267, 1987.
4. Biedl, T.; Kant, G.: A Better Heuristic for Orthogonal Graph Drawings, Technical Report UU-CS-1995-04, Utrecht University, 1995, *also in* Proc. 2nd Ann. European Symposium on Algorithms (ESA '94), LNCS 855, pp. 24-35, Springer-Verlag, 1994.
5. Even, S.; Granot, G.: Grid Layouts of Block Diagrams – Bounding the Number of Bends in Each Connection, *in* Tamassia, R.; Tollis, I.G., eds.: Graph Drawing, Proc. DIMACS Intern. Workshop GD'94, LNCS 894, pp. 64-75, Springer-Verlag, 1995.
6. Gansner, E.R.; Koutsofios, E.; North, S.C.; Vo, K.-P.: A Technique for Drawing Directed Graphs, IEEE Trans. on Software Engineering, 19(3), pp. 214-230, 1993.
7. Himsolt, M.: GraphEd – A Graphical Platform for the Implementation of Graph Algorithms, *in* Tamassia, R.; Tollis, I.G., eds.: Graph Drawing, Proc. DIMACS Intern. Workshop GD'94, LNCS 894, pp. 182-193, Springer-Verlag, 1995.
8. Johnson, D.: The NP-completeness column: An ongoing guide, Journal on Algorithms, 3(1), pp. 215-218, 1982
9. Protsko, L.B.; Sorenson, P.G.; Tremblay, J.P.; Schaefer, D.A.: Towards the Automatic Generation of Software Diagrams, IEEE Trans. on Software Engeneering, 17(1), pp. 10-21, 1991,
10. Sander, G.: Graph Layout Through the VCG Tool, *in* Tamassia, R.; Tollis, I.G., eds.: Graph Drawing, Proc. DIMACS Intern. Workshop GD'94, LNCS 894, pp. 194-205, Springer-Verlag, 1995. The VCG tool is publicly available via http://www.cs.uni-sb.de:80/RW/users/sander/html/gsvcg1.html.
11. Sugiyama, K., Tagawa, S., Toda, M.: Methods for Visual Understanding of Hierarchical Systems, IEEE Trans. Sys., Man, and Cybernetics, SMC 11(2), pp. 109-125, 1981.
12. Tamassia, R.: On Embedding a Graph in the Grid with the Minimum Number of Bends, SIAM Journal of Computing, 16(3), pp. 421-444, 1987.
13. Tamassia, R., Tollis, I.G.: Planar Grid Embedding in Linear Time, IEEE Trans. on Circuits and Systems, 36(9), pp. 1230-1234, 1989.
14. Vijayan G.; Wigderson A.: Rectilinear Graphs and Their Embeddings, SIAM Journal of Computing, 14(2), pp. 355-372, 1985.
15. Warfield, N.J.: Crossing Theory and Hierarchy Mapping, IEEE Trans. Sys., Man, and Cybernetics, SMC 7(7), pp. 505-523, 1977.
16. Wieners-Lummer, C.: Manhattan Channel Routing with Good Theoretical and Practical Performance, ACM SIAM Symp. on Disc. Alg., pp. 465-474, 1990.