Fast and Simple Horizontal Coordinate Assignment

Ulrik Brandes and Boris Köpf

Department of Computer & Information Science,
University of Konstanz, Box D 188, 78457 Konstanz, Germany
{Ulrik.Brandes|Boris.Koepf}@uni-konstanz.de

Abstract. We present a simple, linear-time algorithm to determine horizontal coordinates in layered layouts subject to a given ordering within each layer. The algorithm is easy to implement and compares well with existing approaches in terms of assignment quality.

1 Introduction

In layered graph layout, vertices are placed on parallel lines corresponding to an ordered partition into layers. W.l.o.g. these lines are horizontal, and we assume a polyline representation in which edges may bend where they intersect a layer. The standard approach for layered graph layout consists of three phases [18]: layer assignment (vertices are assigned to layers), crossing reduction (vertices and bend points are permuted) and coordinate assignment (coordinates are assigned to vertices and bend points).

The third phase is usually constrained to preserve the ordering determined in the second phase, and to introduce a minimum separation between layers, and between vertices and bend points within a layer. Criteria for readable layout include length and slope of edges, straightness of long edges, and balancing of edges incident to the same vertex. Note that, if an application does not prescribe vertical coordinates, it is easy to determine layer distances that bring about a minimum edge slope. We therefore confine ourselves to horizontal coordinate assignment.

Previous approaches for horizontal coordinate assignment either optimize a constrained objective function of coordinate differences, iteratively improve a candidate layout using one or more of various heuristics, or do both [18,10,6, 17,8,14,7,15,16,5]. A recently introduced method [3] successfully complements some of the above heuristics with new ideas to determine visually compelling assignments in time $\mathcal{O}(N \log^2 N)$, where N is the total number of vertices, bend points and edges. We present a much simpler algorithm that runs in time $\mathcal{O}(N)$ without compromising on layout quality.

In Sect. 2, we define some terminology to state the horizontal coordinate assignment problem formally. Several important ideas introduced in previous approaches are reviewed in Sect. 3, and our new method is described in Sect. 4.

P. Mutzel, M. Jünger, and S. Leipert (Eds.): GD 2001, LNCS 2265, pp. 31–44, 2002.
© Springer-Verlag Berlin Heidelberg 2002

2 Preliminaries

A *layering* $L = (L_1, \ldots, L_h)$ of a graph $G = (V, E)$ is an ordered partition of V into non-empty *layers* L_i such that adjacent vertices are in different layers. Let $L(v) = i$ if $v \in L_i$. Oblivious whether G is directed or undirected, an edge incident to $u, v \in V$ is denoted by (u, v) if $L(u) < L(v)$. An edge (u, v) is *short* if $L(v) - L(u) = 1$, otherwise it is *long* and *spans* layers $L_{L(u)+1}, \ldots, L_{L(v)-1}$. Let $N_v^- = \{u : (u, v) \in E\}$ ($N_v^+ = \{w : (v, w) \in E\}$) denote the *upper* (*lower*) *neighbors* and $d_v^- = |N_v^-|$ ($d_v^+ = |N_v^+|$) the *upper* (*lower*) *degree* of $v \in V$.

A layering is *proper*, if there are no long edges. Any layering can be turned into a proper one by subdividing long edges (u, v) with *dummy vertices* $b_i \in L_i$, $i = L(u) + 1, \ldots, L(v) - 1$, that represent potential edge bends.

A *layered graph* $G = (V \cup B, E; L)$ is a graph G together with a proper layering L, where the vertices are either *original* vertices in V or dummy vertices (having upper and lower degree one) in B. The edges of a layered graph are often called (edge) *segments*, and segments between two dummy vertices are called *inner* segments. A maximal path in G whose internal vertices are all dummy vertices is also called a long edge. Let $N = |V \cup B| + |E|$ denote the size of a layered graph.

An *ordering* of a layered graph is a partial order $\prec$ of $V \cup B$ such that either $u \prec v$ or $v \prec u$ if and only if $L(u) = L(v)$. We sometimes denote the vertices by $v_j^{(i)}$, where $L_i = \{v_1^{(i)}, \ldots, v_{|L_i|}^{(i)}\}$ with $v_1^{(i)} \prec \ldots \prec v_{|L_i|}^{(i)}$. The *position* $\mathrm{pos}[v_j^{(i)}] = j$ and the *predecessor* of $v_j^{(i)}$ with $j > 1$ is $\mathrm{pred}[v_j^{(i)}] = v_{j-1}^{(i)}$. An edge segment (u, v) is said to *cross* an edge segment (u', v'), if $u, u' \in L_i$, $v, v' \in L_{i+1}$, and either $u \prec u'$ and $v' \prec v$, or $u' \prec u$ and $v \prec v'$.

Given a layered graph together with an ordering, the *horizontal coordinate assignment problem* is to assign coordinates to the vertices such that the ordering and a *minimum separation* $\delta > 0$ are respected, i.e.

> **Horizontal Coordinate Assignment Problem:** For a layered graph $G = (V \cup B, E; L)$ with ordering $\prec$, find real values $x(v)$, $v \in V \cup B$, such that $x(u) + \delta \leq x(v)$ if $u \prec v$ (*minimum separation constraint*).

A horizontal coordinate assignment should additionally satisfy two main criteria which appear to govern the readability of a layered drawing with given ordering:

- edges should have small length,
- vertex positions should be balanced between upper and lower neighbors,
- and long edges should be as straight as possible.

3 Previous Work

Apparently, an early algorithm to determine horizontal coordinates is used in a system for control flow diagrams, but we were unable to find a detailed description [11]. In the following, we summarize more recent approaches, since they nicely illustrate the rationale behind our approach. Horizontal coordinate assignment is also discussed in [4, Section 9.3] and [1].

Optimization approaches. In their seminal paper [18], Sugiyama et al. present a quadratic program for the horizontal coordinate assignment problem. The objective function is a weighted sum of terms

$$\sum_{(u,v)\in E} (x(u) - x(v))^2 \tag{1}$$

and

$$\sum_{v\in V} \left(x(v) - \sum_{u\in N_v^-} \frac{x(u)}{d_v^-} \right)^2 + \sum_{v\in V} \left(x(v) - \sum_{w\in N_v^+} \frac{x(w)}{d_v^+} \right)^2 . \tag{2}$$

The first term penalizes large edge lengths, while the second serves to balance the influence of upper and lower neighbors. The objective function is subject to the minimum separation constraint and, to enforce vertical inner segments, $x(u) = x(v)$ if $(u, v) \in E$ for $u, v \in B$. Note that the quadratic program is infeasible if $\prec$ implies a crossing between inner segments of long edges.

A related approach is introduced in [5]. A necessary condition for an optimal solution of (1) is that all partial derivatives are zero. Equivalently, every vertex is placed at the mean coordinate of its neighbors. The system of linear equations thus obtained is modified so that upper and lower neighbors contribute equally,

$$x(v) = \frac{\sum_{u\in N_v^-} \frac{x(u)}{d_v^-} + \sum_{w\in N_v^+} \frac{x(w)}{d_v^+}}{2} . \tag{3}$$

If the coordinates of vertices in the top and bottom layer are fixed, say equidistantly, to exclude the trivial assignment of all-equal coordinates, the system has a unique solution that can be approximated quickly using Gauß-Seidel iteration. Though a given ordering may not be preserved, the resulting layouts have interesting properties with respect to planarity and symmetry [5].

Another quadratic program is discussed in [4, p. 293f]. For every directed path $v_1, \ldots, v_k$ where $v_1, v_k \in V$ and $v_2, \ldots, v_{k-1} \in B$, terms

$$\left(x(v_i) - x(v_1) - \frac{i-1}{k-1}(x(v_k) - x(v_1)) \right)^2 ,$$

$i = 2, \ldots, k-1$, are introduced to make long edges straight, and only the minimum separation constraints are enforced.

Some popular implementations [9,13] use a piecewise linear objective function introduced in [8] reading

$$\sum_{e=(u,v)\in E} \omega(e) \cdot |x(u) - x(v)| , \tag{4}$$

subject to the minimum separation constraint. The weight $\omega(e)$ reflects the importance of drawing edge e vertically, and weights of 1, 2, and 8 are used for edges incident to 0, 1, and 2 dummy vertices, respectively. The corresponding integer optimization problem is solved to optimality using a clever transformation and the network simplex method. This objective function is the main justification for our heuristic.

Iterative heuristics. After an initial, say leftmost, placement respecting the ordering and minimum separation constraint, several heuristics can be applied to improve the assignment with respect to the criteria stated in Sect. 2.

The method proposed in [18] sweeps up and down the layering. While sweeping down, vertices in each layer are considered in order of non-increasing upper degree. Each vertex is shifted toward the average coordinate of its upper neighbors, but without violating the minimum separation constraint. To increase the available space, lower priority vertices may be shifted together with the current one. The reverse sweep is carried out symmetrically. A post-processing to straighten long edges without moving original vertices is applied in [7].

In addition to the average coordinates of upper or lower neighbors, the average of all neighbors is considered in [14,16]. Instead of a priority order based on degree, violation of the minimum separation constraint is avoided by grouping vertices and averaging over their independent movements. In [15], this kind of grouping is extended to paths of dummy vertices, so as to constrain inner segments of long edges to be vertical.

Several heuristic improvements are proposed in [8]. They are applied iteratively and the best assignment with respect to (4) is kept after each iteration. One of these heuristics straightens long edges in apparently the same way as [7]. Another one is similar to the method of [18], but, since the objective function consists of absolute instead of squared differences, uses the median instead of the average of the neighbor's coordinates.

A fast non-iterative heuristic is presented in [3] and implemented in the AGD library [13]. Similar to [15], the dummy vertices of each long edge are grouped, and leftmost and rightmost top-to-bottom placements of all vertices subject to this grouping are determined. Dummy vertices are fixed at the mean of their two positions thus obtained. In a rather involved second phase, original vertices are placed so as to minimize the length of some short edges as measured by (4) without changing positions of dummy vertices.

Since local improvements, such as straightening edges after y-coordinates have been determined [2], can be applied to layouts obtained from any method, all examples in this paper have been prepared without such postprocessing to better facilitate comparison.

4 The Algorithm

We present a heuristic approach for the horizontal coordinate assignment problem that guarantees vertical inner segments, and yields small edge lengths and a fair balance with respect to upper and lower neighbors.

We essentially follow the inherent objective of (4), i.e. we define the length of an edge segment (u, v) by $|x(u) - x(v)|$. Recall that $\sum_{i=1}^{k} |x - x_i|$ is minimized if x is the median of the x_i. Therefore, we align each vertex vertically with its median neighbor wherever possible. To achieve a balance between upper and

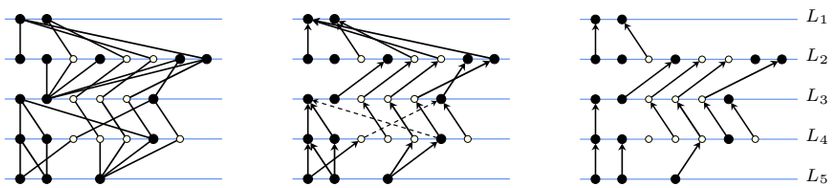
lower neighbors similar to the one that motivated (2) and (3), their medians are considered separately and the results are combined.

The algorithm consists of three basic steps. The first two steps are carried out four times. In the first step, referred to as vertical alignment, we try to align each vertex with either its median upper or its median lower neighbor, and we resolve alignment conflicts (of type 0) either in a leftmost or a rightmost fashion. We thus obtain one vertical alignment for each combination of upward and downward alignment with leftmost and rightmost conflict resolution. In the second step, called horizontal compaction, aligned vertices are constrained to obtain the same horizontal coordinate, and all vertices are placed as close as possible to the next vertex in the preferred horizontal direction of the alignment. Finally, the four assignments thus obtained are combined to balance their biases.

Details on vertical alignment are given in the following two sections, though only for the case of upward alignment to the left. The other three cases are symmetric. Balanced combination of assignments is described in Sect. 4.3. thus obtained.

4.1 Vertical Alignment

We want to align each vertex with a median upper neighbor. Two alignments are conflicting if their corresponding edge segments cross or share a vertex. We classify conflicts according to the number of inner segments involved.



(a) layered graph (b) candidates (c) alignment

Fig. 1. Leftmost alignment with median upper neighbors (dummy vertices are outlined, non-inner segments involved in type 1 conflicts are dashed)

Type 2 conflicts correspond to a pair of crossing inner segments and prevent at least one of them from being vertical. One or both of the involved segments can therefore be marked as non-vertical and ignored when alignments are determined. Since vertical inner segments appear to improve readability dramatically, however, we assume that type 2 conflicts have been avoided in the crossing reduction phase (as, e.g., in [15]). Alternatively, one can eliminate type 2 conflicts in a preprocessing step prior to the horizontal coordinate assignment, e.g. by

swapping the two lower vertices involved until the crossing is no longer between two inner segments [3,2]. Note that this changes the ordering, and potentially the number of crossings. If the ordering is more important than vertical inner segments, the original ordering can be restored in the final layout. Finally, type 2 conflicts can also be treated as described below for type 0 conflicts.

Alg. 1: Preprocessing (mark type 1 conflicts)

```
for i ← 2, ..., h − 2 do
    k_0 ← 0;  l ← 1;
    for l_1 ← 1, ..., |L_{i+1}| do
        if l_1 = |L_{i+1}| or v_{l_1}^{(i+1)} incident to inner segment between L_{i+1} and L_i
        then
            k_1 ← |L_i|;
            if v_{l_1}^{(i+1)} incident to inner segment between L_{i+1} and L_i then
                k_1 ← pos[upper neighbor of v_{l_1}^{(i+1)}];
            while l ≤ l_1 do
                foreach upper neighbor v_k^{(i)} of v_l^{(i+1)} do
                    if k < k_0 or k > k_1 then mark segment (v_k^{(i)}, v_l^{(i+1)});
                l ← l + 1;
            k_0 ← k_1;
```

Type 1 conflicts arise when a non-inner segment crosses an inner segment. Again because vertical inner segments are preferable, they are resolved in favor of the inner segment. We mark type 1 conflicts during a preprocessing step given by Alg. 1. The algorithm traverses layers from left to right (index l) while maintaining the upper neighbors, $v_{k_0}^{(i)}$ and $v_{k_1}^{(i)}$, of the two closest inner segments. It clearly runs in linear time and marks non-inner segments involved in type 1 conflicts so that they can be ignored when determine alignments are determined. Observe that it is easy to modify this preprocessing to either mark type 2 conflicts or eliminate them on the fly by swapping the lower vertices of crossing inner segments.

Finally, a *type 0* conflict corresponds to a pair of non-inner segments that either cross or share a vertex. We say that a segment (u, v) is *left* of a segment (u', v'), if either $v \prec v'$, or $v = v'$ and $u \prec u'$. Type 0 conflicts are resolved greedily in a leftmost fashion, i.e. in every layer we process the vertices from left to right and for each vertex we consider its median upper neighbor (its left and right median upper neighbor, in this order, if there are two). The pair is aligned, if no conflicting alignment is left of this one. The resulting bias is mediated by the fact that the symmetric bias is applied in one of the other three assignments.

By executing Alg. 2 we obtain a leftmost alignment with upper neighbors. A maximal set of vertically aligned vertices is called a *block*, and we define the *root* of a block to be its topmost vertex. Observe that blocks are represented

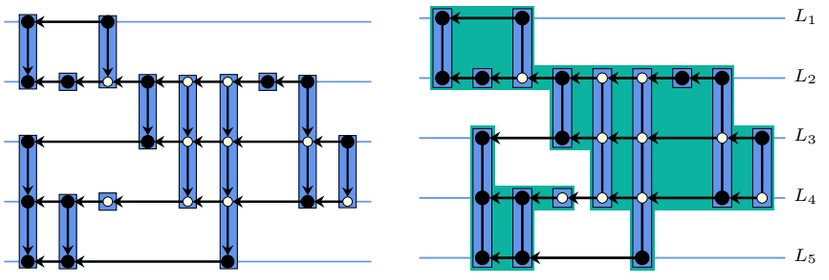
Alg. 2: Vertical alignment

initialize root$[v] \leftarrow v$, $v \in V \cup B$;
initialize align$[v] \leftarrow v$, $v \in V \cup B$;
for $i \leftarrow 1, \ldots, h$ **do**
  $r \leftarrow 0$;
  for $k \leftarrow 1, \ldots, |L_i|$ **do**
    if $v_k^{(i)}$ *has upper neighbors* $u_1 \prec \ldots \prec u_d$ *with* $d > 0$ **then**
      for $m \leftarrow \lfloor \frac{d+1}{2} \rfloor, \lceil \frac{d+1}{2} \rceil$ **do**
        if $align[v_k^{(i)}] = v_k^{(i)}$ **then**
          if $(u_m, v_k^{(i)})$ *not marked and* $r < pos[u_m]$ **then**
            align$[u_m] \leftarrow v_k^{(i)}$;
            root$[v_k^{(i)}] \leftarrow$ root$[u_m]$;
            align$[v_k^{(i)}] \leftarrow$ root$[v_k^{(i)}]$;
            $r =$ pos$[u_m]$;

by cyclically linked lists, where each vertex has a reference to its lower aligned neighbor, and the lowest vertex refers back to the topmost. Moreover, each vertex has an additional reference to the root of its block. These data structures are sufficient for the actual placement described in the next section.

4.2 Horizontal Compaction

In the second step of our algorithm, a horizontal coordinate assignment is determined subject to a vertical alignment, i.e. all vertices of a block are assigned the coordinate of the root.



(a) blocks (b) classes

Fig. 2. Blocks and classes with respect to the alignment of Fig. 1(c)

Consider the *block graph* obtained by introducing directed edges between each vertex and its predecessor (if any) and contracting blocks into single vertices. See Fig. 2(a) and note that the root of a block that is a sink in this acyclic graph is always a leftmost vertex in its layer, so that there is at most one vertex of this kind in each layer.

We partition the block graph into *classes*. The class of a block is defined by that reachable sink which has the topmost root.[1] Within each class, we apply a longest path layering, i.e. the relative coordinate of a block with respect to the defining sink is recursively determined to be the maximum coordinate of the preceding blocks in the same class, plus minimum separation.

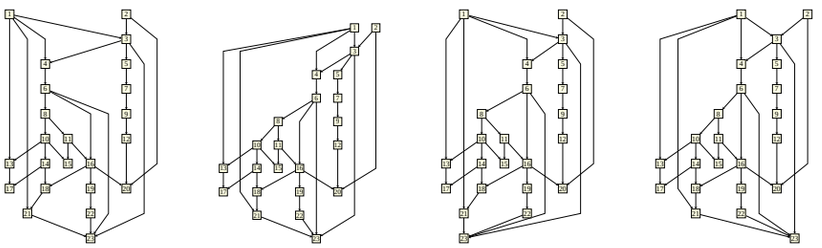
Alg. 3: Horizontal compaction

```
function place_block(v)
begin
    if x[v] undefined then
        x[v] ← 0;   w ← v;
        repeat
            if pos[w] > 1 then
                u ← root[pred[w]];
                place_block(u);
                if sink[v] = v then sink[v] = sink[u];
                if sink[v] ≠ sink[u] then
                    shift[sink[u]] ← min{shift[sink[u]], x[v] − x[u] − δ};
                else
                    x[v] ← max{x[v], x[u] + δ};
            w ← align[w];
        until w = v;
end

initialize sink[v] ← v, v ∈ V ∪ B;
initialize shift[v] ← ∞, v ∈ V ∪ B;
initialize x[v] to be undefined, v ∈ V ∪ B;

// root coordinates relative to sink
foreach v ∈ V ∪ B do if root[v] = v then place_block(v);

// absolute coordinates
foreach v ∈ V ∪ B do
    x[v] ← x[root[v]];
    if shift[sink[root[v]]] < ∞ then
        x[v] ← x[v] + shift[sink[root[v]]]
```

[1] A similar definition is given in [3]. However, our blocks may contain original vertices and non-inner segments, and therefore give rise to bigger classes. Note that, within a class, coordinates are easy to determine.

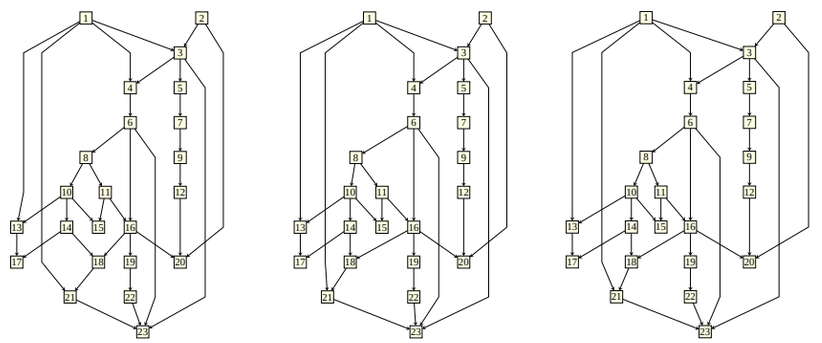
For each class, from top to bottom, we then compute the absolute coordinates of its members by placing the class with minimum separation from previously placed classes.

The entire compaction step is implemented in Alg. 3. The first iteration invokes a recursive version of a technique known as longest path layering to determine the relative coordinates of all roots with respect to the sink of their corresponding classes. Another variant of this algorithm is used to determine visibility representations of planar layered graphs [12]. At the same time, we determine for each sink the minimum distance of a vertex in its class from its neighboring vertex in a class with a higher sink. The second iteration distributes this information from the roots to all vertices to obtain the absolute coordinates.



leftmost upper rightmost upper leftmost lower rightmost lower

Fig. 3. Biased assignments resulting from leftmost/rightmost alignments with median upper/lower neighbors for the running example of [2,3]



AGD implementation of involved heuristic [3] | our method with the same separation | our method with uniform separation

Fig. 4. Final assignments compared

4.3 Balancing

Although the coordinate assignments computed in the first two steps result in vertical inner segments and in general display short edge lengths, they are governed by their specific choices of a vertical alignment direction and horizontal preference. We even out their directional tendencies by combining them into a balanced horizontal coordinate assignment.

First, we align the layouts to the one with smallest width by shifting the two assignments for the leftmost (rightmost) alignments so that their minimum (maximum) coordinate agrees with the minimum coordinate in the smallest-width assignment. Out of the four resulting candidate coordinates we fix, for each vertex separately, the final coordinate to be the *average median*, which for k values $x_1 \leq \ldots \leq x_k$ equals $(x_{\lfloor (k+1)/2 \rfloor} + x_{\lceil (k+1)/2 \rceil})/2$. This choice is justified by the following lemma.

Lemma 1. *The average median is order and separation preserving.*

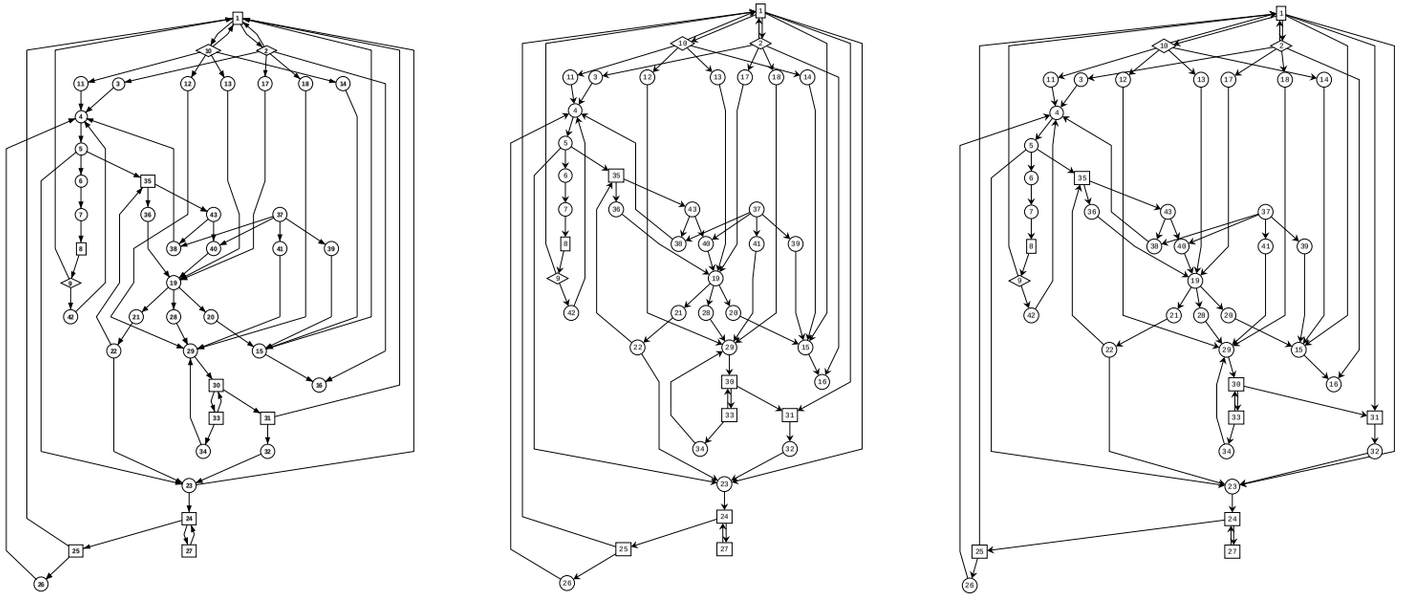
Proof. Let x_i, y_i, $i = 1, \ldots, k$ be pairs of real values with $x_i + \delta \leq y_i$ for some $\delta \geq 0$. W.l.o.g assume that $x_1 \leq \ldots \leq x_k$, which implies $x_i + \delta \leq y_j$ for $1 \leq i \leq j \leq k$. In particular, there are at least $k - i$ values among $y_1, \ldots, y_k$ larger than or equal to $x_i + \delta$. Let π be a permutation of $\{1, \ldots, k\}$ with $y_{\pi(1)} \leq \ldots \leq y_{\pi(k)}$. It follows that $x_i + \delta \leq y_{\pi(i)}$ and therefore $\frac{1}{2}(x_{\lfloor (k+1)/2 \rfloor} + x_{\lceil (k+1)/2 \rceil}) + \delta \leq \frac{1}{2}(y_{\pi(\lfloor (k+1)/2 \rfloor)} + y_{\pi(\lceil (k+1)/2 \rceil)})$. □

We have chosen the average median over the mean because it appears to suit better the way that biased assignments are determined. Extreme coordinates specific for a particular combination of vertical and horizontal directions of preference are dropped, and the two closer ones are averaged. In ideal situations, vertices end up at the average coordinate of their median upper and lower neighbors, thus balancing between upward and downward edge length minimization. Moreover, if a vertex is aligned twice, say, with its median upper neighbor while it is positioned unevenly to the left and right in the other two assignments, the average median lets straightness take precedence over averaging.

Alg. 4: Horizontal coordinate assignment

```
preprocessing using Alg. 1;
for vertical direction up, down do
    for horizontal direction left, right do
        vertical alignment with Alg. 2;
        horizontal compaction with Alg. 3;
align to assignment of smallest width;
set coordinates to average median of aligned candidates;
```

Our method for horizontal coordinate assignment is summarized in Alg. 4.



(a) daVinci [7] (b) AGD implementation of [3] (c) our method

Fig. 5. Example graph used in several publications on layered layout [7, p. 5]. The layout of the daVinci system, where horizontal coordinates are determined using the iterative heuristic of [18] with additional edge straightening, is used as a reference to fix layers, ordering, and y-coordinates. The other two layouts are computed with the minimum separation values of [3] and without postprocessing

Fig. 6. Impact of alignment on layout width

The final assignment obtained for the example from Fig. 3 is shown in Fig. 4 and compared to the result obtained with the heuristic of [3]. Note that our algorithm does not depend on uniform minimum separation. In fact, the minimum separation can be chosen independently for each pair of neighboring vertices. A common choice, used in many implementations, is the sum of half of the vertex widths plus some constant, though we obtain better layouts with uniform separation when vertex widths do not differ significantly.

Theorem 1. *Algorithm 4 computes a horizontal coordinate assignment in time $\mathcal{O}(N)$, where N is the total number of vertices and edge segments. If the minimum separation is even, the assigned coordinates are integral.*

Proof. Since the median upper and lower neighbors needed in Alg. 2 can be determined once in advance, each of Algs. 1, 2, and 3 requires time proportional to the number of vertices and edges in the layered graph.

Note that all coordinates in a biased assignment are multiples of the minimum separation. Since the final coordinates are averages of two biased coordinates, they are integral if the minimum separation is even. □

5 Discussion

We presented a simple linear-time algorithm for the horizontal coordinate assignment problem which requires no sophisticated data structures and is easy to implement. Preliminary computational experiments suggest that our algorithm is not only faster (also in practice), but that its coordinate assignments compare well with those produced by more involved methods. Figure 5 gives a realistic example.

Figure 6 illustrates that the alignment constraint has a significant impact on the width of the layout. Clearly, the presence of many long edges increases chances of large width requirements. A closer investigation of this trade-off may suggest means to control for this effect. Moreover, edges in the block graph could be assigned a cost corresponding to the number of edge segments connecting the same two blocks. With such costs, an adaptation of the minimum cost flow approach for one-dimensional compaction of orthogonal representations with rectangular faces described in [4, Sect. 5.4] can yield smaller edge lengths. Our aim here, however, was a simple approach that is easy to implement.

Our algorithm has several generic elements that could be instantiated differently. While it has already been mentioned that crossing inner segments can be

dealt with in many ways, one could alter also the alignment (e.g., to only consider one of two medians, or to break conflicts in favor of high degree vertices), the method of compaction (e.g., using adaptive schemes of separation), and the final combination (e.g., aligning the central axes and fixing the average).

References

1. Oliver Bastert and Christian Matuszewski. Layered drawings of digraphs. In Michael Kaufmann and Dorothea Wagner, editors, *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes in Computer Science*, pages 104–139. Springer, 2001.
2. Christoph Buchheim, Michael Jünger, and Sebastian Leipert. A fast layout algorithm for k-level graphs. Technical Report 99-368, Department of Economics and Computer Science, University of Cologne, 1999.
3. Christoph Buchheim, Michael Jünger, and Sebastian Leipert. A fast layout algorithm for k-level graphs. In Joe Marks, editor, *Proceedings of the 8th International Symposium on Graph Drawing (GD 2000)*, volume 1984 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2001.
4. Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.
5. Peter Eades, Xuemin Lin, and Roberto Tamassia. An algorithm for drawing a hierarchical graph. *International Journal of Computational Geometry & Applications*, 6:145–156, 1996.
6. Peter Eades and Kozo Sugiyama. How to draw a directed graph. *Journal of Information Processing*, 13(4):424–437, 1990.
7. Michael Fröhlich and Mattias Werner. The graph visualization system daVinci — a user interface for applications. Technical Report 5/94, Department of Computer Science, University of Bremen, 1994.
8. Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.
9. Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software—Practice and Experience*, 30(11):1203–1233, 2000.
10. Emden R. Gansner, Stephen C. North, and Kiem-Phong Vo. DAG – A program that draws directed graphs. *Software—Practice and Experience*, 17(1):1047–1062, 1988.
11. Lois M. Haibt. A program to draw multilevel flow charts. In *Proceedings of the Western Joint Computer Conference*, volume 15, pages 131–137, 1959.
12. Xuemin Lin and Peter Eades. Area minimization for grid visibility representation of hierarchically planar graphs. In Takao Asano, Hiroshi Imai, Der-Tsai Lee, Shinichi Nakano, and Takeshi Tokuyama, editors, *Proceedings of the 5th International Conference on Computing and Combinatorics (COCOON '99)*, volume 1627 of *Lecture Notes in Computer Science*, pages 92–102. Springer, 1999.
13. Petra Mutzel, Carsten Gutwenger, Ralf Brockenauer, Sergej Fialko, Gunnar W. Klau, Michael Krüger, Thomas Ziegler, Stefan Näher, David Alberts, Dirk Ambras, Gunter Koch, Michael Jünger, Christoph Buchheim, and Sebastian Leipert. A library of algorithms for graph drawing. In Sue H. Whitesides, editor, *Proceedings of the 6th International Symposium on Graph Drawing (GD '98)*, volume 1547 of *Lecture Notes in Computer Science*, pages 456–457. Springer, 1998.

14. Georg Sander. Graph layout through the VCG tool. In Roberto Tamassia and Ioannis G. Tollis, editors, *Proceedings of the DIMACS International Workshop on Graph Drawing (GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 1995.
15. Georg Sander. A fast heuristic for hierarchical Manhattan layout. In Franz J. Brandenburg, editor, *Proceedings of the 3rd International Symposium on Graph Drawing (GD '95)*, volume 1027 of *Lecture Notes in Computer Science*, pages 447–458. Springer, 1996.
16. Georg Sander. Graph layout for applications in compiler construction. *Theoretical Computer Science*, 217(2):175–214, 1999.
17. Kozo Sugiyama and Kazuo Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man and Cybernetics*, 21(4):876–892, 1991.
18. Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, February 1981.