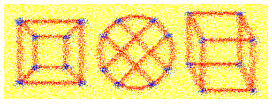

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 11, no. 2, pp. 371–396 (2007)

A 2.5D Hierarchical Drawing of Directed Graphs

Seok-Hee Hong

IMAGEN Program, National ICT Australia Ltd.
School of IT, University of Sydney, NSW, Australia
seokhee.hong@nicta.com.au

Nikola S. Nikolov

Department of CSIS, University of Limerick, Limerick, Ireland
nikola.nikolov@ul.ie

Alexandre Tarassov

Department of CSIS, University of Limerick, Limerick, Ireland
IBM HPC Group, Dublin Software Lab, Ireland
alexandre.tarassov@ul.ie

Abstract

We introduce a new graph drawing convention for 2.5D hierarchical drawings of directed graphs. The vertex set is partitioned both into layers of vertices drawn in parallel planes and into $k \geq 2$ subsets, called walls, and also drawn in parallel planes. The planes of the walls are perpendicular to the planes of the layers. We present a method for computing such layouts and introduce five alternative algorithms for partitioning the vertex set into walls which correspond to different aesthetic requirements. We evaluate our method with an extensive computational study.[1]

Article Type	Communicated by	Submitted	Revised
Regular paper	P. Eades and P. Healy	March 2006	February 2007

[1] An online gallery with examples of our 2.5D hierarchical layouts is available at `http://www.cs.usyd.edu.au/~visual/valacon/gallery/3DHL`.

This work is supported by IRCSET under basic research grant SC/2003/161 and National ICT Australia (NICTA). Two extended abstracts of the paper were published in [20, 19].

1 Introduction

The visual representation of hierarchically organised data has applications in areas such as Social Network Analysis, Bioinformatics, and Software Engineering, to mention a few. Hierarchies are commonly modeled by directed graphs (digraphs) and thus visualized by algorithms for drawing digraphs. Most of the research effort in this area has been related to improvements of various aspects of the Sugiyama framework method, the most popular method for creating 2D hierarchical drawings of digraphs [10, 25].

The increasing availability of powerful graphic displays opens the opportunity for developing new methods for 3D graph drawing. There is evidence that 3D graph layouts combined with novel interaction and navigation methods make graphs easier to comprehend by humans and increase the efficiency of task performance on graphs [26]. Three dimensional graph drawings with a variety of aesthetics and edge representations have been extensively studied by the graph drawing community (see [4, 6, 9, 18, 11, 23]). Examples include algorithms for 3D orthogonal drawing with a limited number of bends, 3D straight-line grid drawing algorithms with good resolution (volume), and 3D graph drawing algorithms that maximise symmetry.

There has been relatively little research on drawing digraphs in 3D. One of the known approaches is the method of Ostry which consists of first computing a 2D hierarchical layout of the digraph and then wrapping it around either a cone or a cylinder [22]. Another approach is the one taken by the graph drawing system GIOTTO3D [17]. GIOTTO3D employs a simple 3-phase method, conceptually different from the Sugiyama framework, for 3D hierarchical digraph drawing. In the first phase the digraph is drawn in 2D by a planarisation method; in the second phase vertices are assigned z-coordinates so that all edges point into the same direction and the total edge span is minimized; and at the third phase the shape of the vertices and the edges is determined.

In the present paper we propose and evaluate a 2.5D hierarchical graph drawing based on the Sugiyama framework for 2D hierarchical graph drawing. A 2.5D graph drawing is a drawing where the graph is partitioned into subgraphs, and each subgraph is drawn in a bounded plane in 2D. The Sugiyama framework is a method for drawing digraphs in 2D as hierarchies with the additional property that vertices are grouped in layers. Typically layers occupy either parallel lines, or concentric circles. In the case of parallel lines as many edges as possible point into the same direction, and in the case of concentric circles as many edges as possible point away from the origin. The work presented in this paper considers the case of layers occupying parallel lines.

The Sugiyama framework method consists of four phases or steps. The first step is to remove all directed cycles from the digraph by converting the direction of some edges. In the second step, the vertices of the digraph are partitioned into layers. In the third step, a linear order is established for the vertices in each layer. The last step assigns x-coordinates to all vertices and determines the shape of the edges. Various algorithms, which emphasize on different properties of the drawing, have been suggested for each step of the Sugiyama method [7].

We propose an extra step between the layer-assignment and the vertex-ordering steps in the Sugiyama framework for 2.5D drawing. The new step consists of partitioning the vertex set into subsets, called *walls*. Any subset of the vertex set can be a wall. Layers occupy parallel planes with all edges pointing into the same direction; walls also occupy parallel planes which are perpendicular to the planes of the layers. Each pair of a wall and a layer intersect into a set of vertices placed along the line which is the intersection of the corresponding wall and layer planes. That is, each wall contains a 2D hierarchical drawing. We propose five different wall partitioning algorithms based on different criteria. Examples of such layouts can be seen in Figure 1; all drawings are made with the visual analysis tool GEOMI [1].

The motivation behind the proposed drawing convention is summarized in the following points:

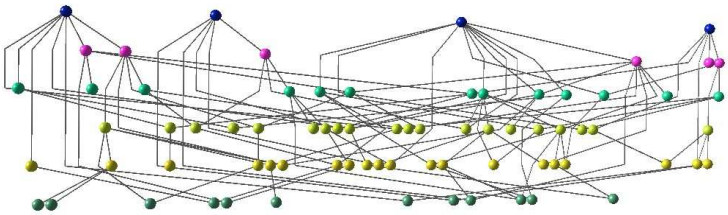
- 2.5D hierarchical drawings of digraphs allow the employment of specific 3D navigation and interaction techniques. For example, each wall can be viewed separately, the camera may move along edges between the walls, etc. [2]
- By grouping the vertices in layers and walls, we decrease the negative effect of occlusion which is a typical obstacle in 3D visualisation.
- The partition of the hierarchy into a set of walls, each containing a smaller 2D hierarchy, allows us to
  - draw the smaller 2D hierarchies efficiently with fast heuristics or even exact algorithms which generally would perform worse if employed for drawing the whole graph as a 2D hierarchy.
  - utilize the extensively developed techniques for drawing hierarchies in 2D.

Our method can be applied to any digraph, such as a class hierarchy that originates from a software engineering application, or a hierarchical relationship in a social network, for example. In particular, we report the results from a computational study with the digraphs in the Rome data set [8].

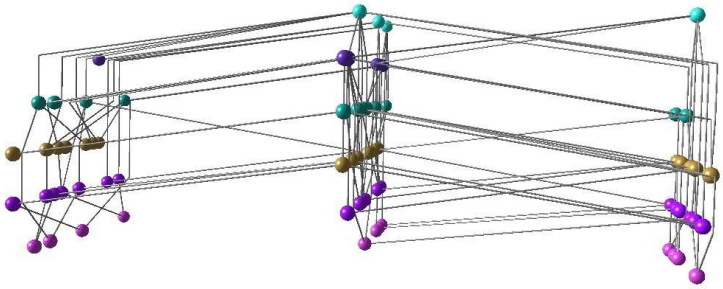
The paper is organised as follows: in the next section, we introduce some definitions and in Section 3 we describe our method and five alternative methods for assigning vertices to walls. The wall-assignment methods are evaluated and compared in an extensive computational study presented in Section 4. In Section 5, we draw conclusions from this work.

2 Terminology

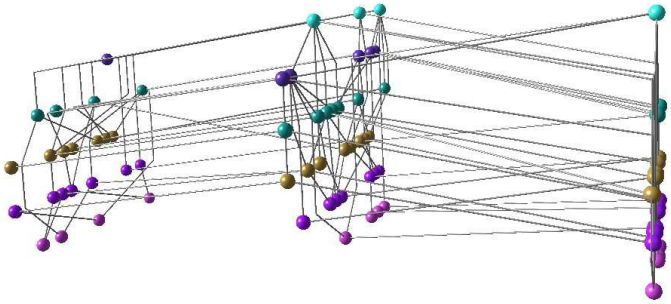
Let $G = (V, E)$ be a digraph without directed cycles. We denote the set of all immediate predecessors of vertex v by $N^-(v) = \{u : (u, v) \in E\}$, and the set of all its immediate successors by $N^+(v) = \{u : (v, u) \in E\}$. A *layering* of G is defined as an ordered partition $L = \{L_1, L_2, ..., L_h\}$ of its vertex set



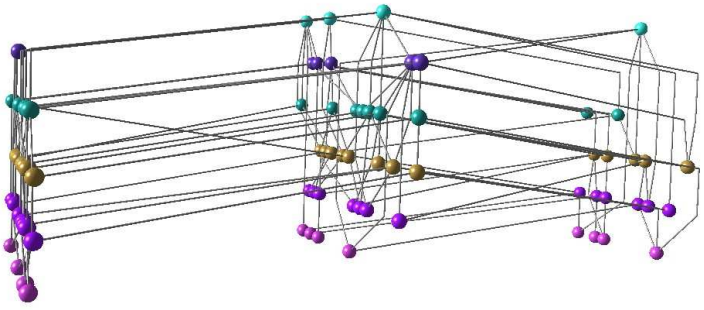
(a) 2D



(b) 2.5D: view 1



(c) 2.5D: view 2



(d) 2.5D: view 3

Figure 1: A 2D and three 2.5D hierarchical drawings of the same graph.

into h subsets, called *layers*, such that $(u, v) \in E$ with $u \in L_i$ and $v \in L_j$ implies $j < i$. A digraph with a layering is a *layered digraph*. A layering is *proper* if all edges are between vertices in adjacent layers. If this is not the case, then after the second step of the Sugiyama method dummy vertices which subdivide long edges, i.e. edges which connect vertices in non-adjacent layers, are introduced. Formally, for each edge $e = (u, v)$ with $u \in L_i$, $v \in L_j$, and $j < i - 1$, we introduce $i - j - 1$ dummy vertices $d^e_{j+1}, d^e_{j+2}, \ldots, d^e_{i-1}$ into layers $L_{j+1}, L_{j+2}, \ldots, L_{i-1}$, respectively. We also replace edge e by edges $(u, d^e_{i-1}), (d^e_{i-1}, d^e_{i-2}), \ldots, (d^e_{j+2}, d^e_{j+1}), (d^e_{j+1}, v)$.

3 2.5D Hierarchical Drawing of Directed Graphs

In this section, we propose a framework for 2.5D hierarchical graph drawing based on the Sugiyama method. In summary, we introduce a new step called *wall assignment* which further partitions the layer into a set of $k > 1$ subsets, called *walls*, after the layering step. Our method is outlined in Algorithm 1.

Algorithm 1 2.5D Hierarchical Digraph Drawing

Step 1 (Cycle Removal): Remove all directed cycles by reversing the direction of some edges.

Step 2 (Layer Assignment): Partition the vertex set into h layers, $L_1, L_2, \ldots, L_h$ with $h \geq 2$.

Step 3 (Wall Assignment): Partition the vertices in each layer L_i into k subsets, $L^1_i, L^2_i, \ldots, L^k_i$ with $k \geq 2$.

Step 4 (Vertex Ordering): Compute a linear order of the vertices which belong to the same layer and wall.

Step 5 (Coordinate Assignment): Assign x-, y-, and z-coordinates to each vertex.

Layers occupy parallel planes and each layer L_i is partitioned into k subsets, $L^1_i, L^2_i, \ldots, L^k_i$. The vertices placed in the j^{th} group of each layer form a *wall*. That is, the set $W^j = \{L^j_1, L^j_2, \ldots, L^j_h\}$ is the j^{th} *wall*. There are k walls in total and they occupy k parallel planes which are perpendicular to the h planes of the layers. In addition, we require all dummy vertices along the same long edge to be in a wall that contains at least one of the endpoints of the edge in order to avoid more than one edge bend along that edge outside the walls.

Since we perform the wall-assignment step after the introduction of dummy vertices, we assume that $G = (V, E)$ is a proper layered digraph with a layering $L = \{L_1, L_2, ..., L_h\}$, i.e. each edge connects vertices in adjacent layers. By partitioning the vertices into $k \geq 2$ walls, we partition the edge set of a digraph into two subsets: *intra wall edges* and *inter wall edges*. Intra wall edges are

edges with both endpoints in the same wall, and inter wall edges are edges with endpoints in different walls. The *span* of an inter wall edge is the absolute value of the difference between the numbers of the two walls which contain the endpoints of that edge. Note that each inter wall edge has at least one endpoint which is not a dummy vertex because we require all dummy vertices along the same long edge to be in the same wall.

The partition of the original vertex set into k walls may originate from the digraph's application domain. They might be the clusters of a given clustered digraph. If no such partition is given, then the vertex set can be partitioned into k walls according to the following optimization criteria:

- **C1.** Even distribution of vertices among walls, i.e. balanced partition of the vertex set into walls.
- **C2.** Minimum number of inter wall edges for avoiding occlusion in the 3D space.
- **C3.** Minimum number of crossings between inter wall edges in the projection of the drawing into a plane which is orthogonal to both the layer planes and the wall planes. This criterion expresses the desire that inter wall edges to be grouped in planes which can only intersect in the walls.
- **C4.** Minimum total edge length of inter wall edges.

These criteria are designed to express the properties of layouts with low visual complexity. They give rise to some hard optimization problems which require the development of efficient algorithms. In the remainder of this section, we propose a few alternative wall-assignment algorithms which are designed to satisfy different subsets of the listed optimization criteria.

3.1 Two-Wall Partitions Based on Minimum Bisection

The minimum number of walls in a 2.5D hierarchical layout is two. The special case of having only two walls is interesting because it combines the advantages of our 2.5D drawing convention with a simple presentation that helps to prevent occlusion. In this and the following sections, we propose two alternative approaches to two-wall assignment according to different optimization criteria.

To find a partition of the vertex set into two walls according to criteria **C1** and **C2** is a special case of the well-known minimum b-balanced cut problem for $b = 1/2$. For a given undirected graph $G = (V, E)$ and a subset of vertices $C \subseteq V$, the *cut* $(C, V \setminus C)$ is the set of all edges with one endpoint in C and one endpoint in $V \setminus C$. Let $w : V \to N$ and $c : E \to N$ be a vertex-weight and an edge cost function, respectively. The minimum b-balanced cut problem is the problem of finding a cut $(C, V \setminus C)$ with the minimum total edge cost, such that $\min\{w(C), w(V - C)\} \geq b.w(V)$ for a given positive $b \leq 1/2$. When $b = 1/2$ and $c(e) = 1$ for each $e \in E$, the problem is also known as the *minimum bisection problem*. This is the same problem that has to be solved if the vertex set is partitioned into two walls according to criteria **C1** and **C2**.

The minimum bisection problem is NP-hard [16]. Various heuristic and approximation algorithms have been proposed for solving it (see [21, 14, 24, 13]) the latest of which is the divide and conquer algorithm by Feige and Krauthgamer which finds a bisection with a cost within ratio of $O(\log^{1.5} n)$ from the minimum in polynomial time [12].

The first wall-assignment algorithm that we propose solves the minimum bisection problem heuristically. It is a simplistic approach to minimum bisection compared to the algorithm of Feige and Krauthgamer, but it is specifically designed for use within the Sugiyama framework. When assigning vertices to walls according to **C1** and **C2**, additional requirements may take higher priority than the exact separation of the vertex set into two equal-size halves with the exact minimum number of edges between them. Such an additional requirement, for example, is to have all dummy vertices along the same edge assigned to the same wall. It is also very important for a wall-assignment algorithm to run fast because it is only a part of a five-step framework.

We propose to assign vertices to walls layer by layer starting with the first layer. For this purpose, we define the minimum one-layer bisection problem as follows.

Minimum One-Layer Bisection Problem
Consider two adjacent layers L_{i-1} and L_i. Let L_{i-1} be partitioned into subsets A_{i-1} and B_{i-1}. Find a partition of L_i into subsets A_i and B_i such that $||A_i| - |B_i|| \leq 1$ and the number of edges between A_{i-1} and B_i plus the number of edges between A_i and B_{i-1} is the minimum.

We propose a greedy algorithm, Algorithm 2, which solves the one-layer bisection problem optimally. It has two phases; at the first phase each vertex in layer L_i is assigned to the wall that contains the biggest number of its immediate successors; at the second phase some vertices are moved from one wall to the other in order to achieve a balanced partition.

Theorem 1 *For given i, A_{i-1}, and B_{i-1}, Algorithm 2 partitions layer L_i into A_i and B_i so that:*

Property 1: $|A_i| = |B_i|$ if $|L_i|$ is even, and $||A_i| - |B_i|| = 1$ if $|L_i|$ is odd.

Property 2: The number of inter wall edges between $A_{i-1} \cup A_i$ and $B_{i-i} \cup B_i$ is the minimum for a partition with Property 1.

Proof: Property 1 is trivially implied by the **while** loop. Assume there exists another partition of layer L_i into subsets A'_i and B'_i which satisfy both Property 1 and Property 2. We will show that the partition defined by A'_i and B'_i must have the same number of inter wall edges as the partition defined by A_i and B_i.

Without loss of generality, assume that $|A_i| \geq |B_i|$ before the **while** loop in Algorithm 2 and let C_i be the set of vertices moved from A_i to B_i after the execution of the **while** loop. Let also A_i^{before} and A_i^{after} denote A_i before and after the **while** loop, respectively. Similarly, let B_i^{before} and B_i^{after} denote B_i before and after the **while** loop, respectively.

Algorithm 2 One-layer-bisection(i)

$A_i \leftarrow \phi$
$B_i \leftarrow \phi$
for all $v \in L_i$ **do**
  if $|N^+(v) \cap A_{i-1}| > |N^+(v) \cap B_{i-1}|$ **then**
    $A_i \leftarrow A_i \cup \{v\}$
  else if $|N^+(v) \cap B_{i-1}| > |N^+(v) \cap A_{i-1}|$ **then**
    $B_i \leftarrow B_i \cup \{v\}$
  else
    if $|A_i| < |B_i|$ **then**
      $A_i \leftarrow A_i \cup \{v\}$
    else
      $B_i \leftarrow B_i \cup \{v\}$
    end if
  end if
end for
if $|A_i| > |B_i|$ **then**
  $X = A$ and $x = B$
else
  $X = B$ and $x = A$
end if
while ($|L_i|$ is even and $|X_i| > |x_i|$) or ($|L_i|$ is odd and $|X_i| > |x_i| + 1$) **do**
  move vertex $v \in X_i$ with the minimum $|N^+(v) \cap X_{i-1}| - |N^+(v) \cap x_{i-1}|$ to x_i
end while

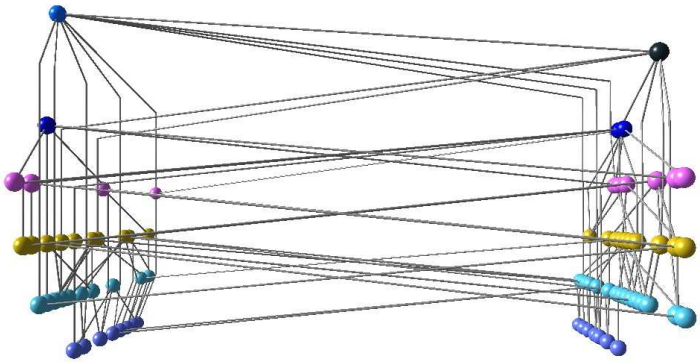


Figure 2: A 2.5D hierarchical layout with a wall partition based on minimum bisection.

Case 1. There is a vertex $v \in A'_i \cap B_i^{before}$. By moving v to B'_i, we subtract from the number of inter wall edges $|N^+(v) \cap B_{i-1}| - |N^+(v) \cap A_{i-1}| \geq 0$. Let $A''_i = A'_i \setminus \{v\}$ and $B''_i = B'_i \cup \{v\}$. If A''_i and B''_i do not satisfy Property 1 because B''_i is too large, then clearly $|B''_i| > |B_i^{before}|$ and there is a vertex $u \in B''_i \cap A_i^{before}$ which can be moved to A''_i to satisfy Property 1 without increasing the number of inter wall edges. If the number of inter wall edges decreases, this will contradict the assumption that the partition (A'_i, B'_i) has the minimum number of inter wall edges. Otherwise we can repeat the same considerations until we end up with $A''_i \cap B_i^{before} = \phi$. Then we continue with the considerations in case 2.

Case 2. $A'_i \subseteq A_i^{before}$. If A'_i is different from A_i^{after}, then $B'_i \cap A_i^{after}$ is not empty. Let $v \in B'_i \cap A_i^{after}$. By moving v to A'_i, we will subtract from the number of inter wall edges $\alpha = |N^+(v) \cap A_{i-1}| - |N^+(v) \cap B_{i-1}| \geq 0$. (Note that $A^{after} \subseteq A^{before}$.) Let $A''_i \leftarrow A'_i \cup \{v\}$ and $B''_i \leftarrow B'_i \setminus \{v\}$. If A''_i and B''_i do not satisfy Property 1 because A''_i is too large, then clearly $|A''_i| > |A_i^{after}|$ and there is a vertex $u \in C_i \cup A''_i$ which is different from v and $\beta = |N^+(u) \cap A_{i-1}| - |N^+(u) \cap B_{i-1}| \leq \alpha$. Let $A'''_i \leftarrow A''_i \setminus \{u\}$ and $B'''_i \leftarrow B''_i \cup \{u\}$. The partition (A'''_i, B'''_i) has $\alpha - \beta \geq 0$ fewer inter wall edges than the partition (A'_i, B'_i). If A'''_i is different from A_i^{after}, we can repeat the considerations for A'''_i and B'''_i. Since this procedure cannot repeat forever, it will eventually end up with $A'''_i = A_i^{after}$. □

For partitioning the whole vertex set into two walls, we propose to partition the first layer into two halves randomly and then apply Algorithm 2 for all layers from L_2 to L_h layer by layer. An example layout is shown in Figure 2.

The **for** loop takes $O(|V| + |E|)$ time in total because each vertex and edge are scanned once. The **while** loop can take additional $O(|V| \log |V|)$ time in

total if A_i and B_i are sorted lists implemented efficiently, e.g. by Fibonacci heaps. Thus, the total worst-case time complexity of running Algorithm 2 for all layers from L_2 to L_h is $O(|V| \log |V| + |E|)$.

Lemma 1 *The total worst-case time complexity of running Algorithm 2 for all layers from L_2 to L_h is $O(|V| \log |V| + |E|)$.*

Since some layers may have an odd number of vertices, Algorithm 2 cannot guarantee that the vertices will be equally split between the two walls. However the balance is good enough for the purpose of hierarchical digraph drawing and our computational study in Section 4 shows evidence that the algorithm guarantees a relatively low total number of inter wall edges by working optimally on the two-layer scale.

Note that the described method does not necessarily place all dummy vertices along the same edge in the same wall. In order to enforce it, we need to forbid the movement of dummy vertices in the **while** loop in Algorithm 2. That may worsen the balance of the partition, but it may also reduce the number of inter wall edges.

3.2 Two-Wall Partitions with Specific Arrangements of Inter Wall Edges

The following two algorithms for partitioning the vertex set into two walls are designed to have all inter wall edges arranged in a particular pattern such that **C3** is satisfied. We call them zig-zag wall partition and dominating-wall partition respectively.

Both algorithms scan all layers one by one from bottom to top and partition each of them into two subsets. We start with a random balanced partition of the first layer. Each next layer L_i is partitioned into L_i^1 and L_i^2 such that $L_i^1 \cup L_i^2 = L_i$ and $L_i^1 \cap L_i^2 = \phi$ based on the partition of layer L_{i-1}.

In both algorithms this is performed by the procedure `DIVIDELAYER`(i, x, y) which sets $L_i^x \leftarrow \{v \in L_i : N^+(v) \cap L_{i-1}^y = \phi\}$ and $L_i^y \leftarrow L_i \setminus L_i^x$.

Algorithm 3 presents the zig-zag wall partition, and algorithm 4 presents the dominating-wall partition. Example layouts are shown in Figures 3 and 4.

Algorithm 3 Zig-zag wall partition

```
Partition L_1 randomly into two halves.
for i = 2..h do
  if i mod 2 = 0 then
    DIVIDELAYER(i, 1, 2)
  else
    DIVIDELAYER(i, 2, 1)
  end if
end for
```

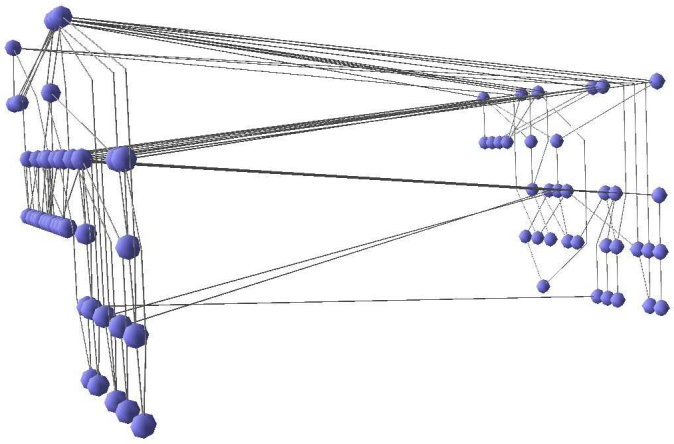


Figure 3: A 2.5D hierarchical layout with a zig-zag wall partition.

In the dominating-wall partition all the inter wall edges have their origins in the same wall, which we call the *dominating wall*, while in the zig-zag wall partitioning both walls may contain origins of inter wall edges.

Algorithm 4 Dominating wall partition

```
Partition L_1 randomly into two halves.
for i = 2..h do
  DIVIDELAYER(i, 2, 1)
end for
```

It is easy to see that both the zig-zag wall partition and the dominating-wall partition place all dummy vertices along edge e into the same wall. Both algorithms take $O(|V|+|E|)$ time because each vertex is examined once and for each vertex, all its immediate successors are also examined.

Lemma 2 *Both the zig-zag wall partition and the dominating-wall partition algorithms assign all dummy vertices along an edge to the same wall and partition the vertex set of the graph into two subsets in linear time.*

3.3 k-Wall Partitions

The next algorithm we present is for partitioning the vertex set into $k \geq 2$ walls according to **C4**. That is, an algorithm that keeps the sum of spans of inter wall edges small.

Similar to the algorithms described above, all the layers are scanned one by one from bottom to top. The first layer is partitioned randomly and each next layer L_i is partitioned on the basis of the partition of layer L_{i-1}.

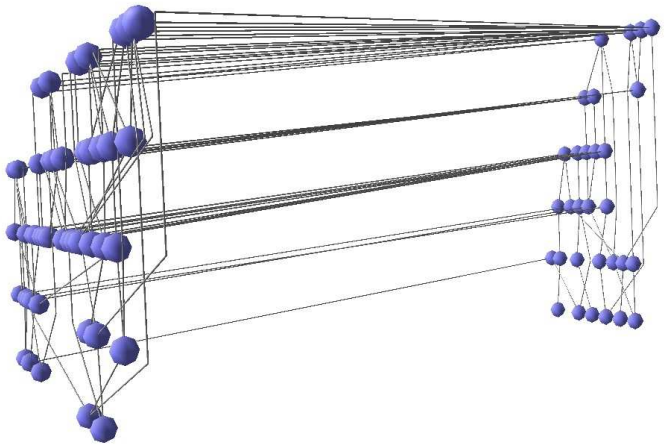


Figure 4: A 2.5D hierarchical layout with a dominating-wall partition.

For partitioning layer L_i into k subsets such that the total span of inter wall edges is small, we apply Algorithm 5. In summary, for each vertex $u \in L_i$, all its immediate successors are considered, and u is placed in the wall whose number is the closest integer to the average of the wall numbers of the immediate successors of u. In other words, the wall u is placed in the barycenter of the walls its immediate successors are placed in. An example layout is shown in Figure 5.

When $k = 2$, Algorithm 5 is basically the same Algorithm 2 without the balancing **while** loop at the end. There is only a slight difference in the treatment of the case when a vertex has the same number of immediate successors in both walls. While Algorithm 2 will distribute those vertices evenly between the two

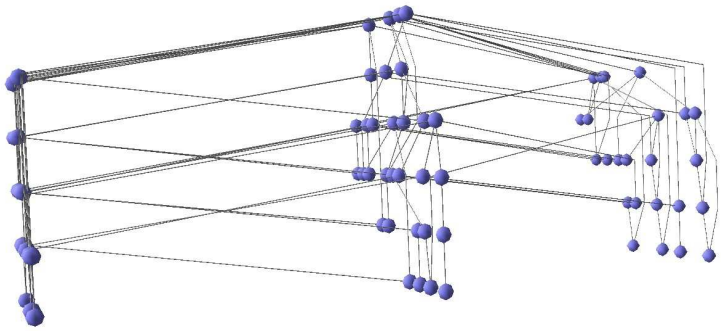


Figure 5: A 2.5D hierarchical layout with a k-wall partition.

Algorithm 5 Barycenter k-wall partition of layer L_i

for all $j = 1..k$ **do**
  $L_i^j \leftarrow \phi$
end for
for all $u \in L_i$ **do**
  if $|N^+(u)| = 0$ **then**
    Let b be a number such that $|L_i^b| = min\{|L_i^1|, |L_i^2|, \ldots, |L_i^k|\}$
  else
    for all $j = 1..k$ **do**
      $neighbours[j] \leftarrow |N^+(u) \cap L_{i-1}^j|$
    end for
    $b \leftarrow \left\lfloor \frac{\sum_{j=1}^{k} j * neighbours[j]}{\sum_{j=1}^{k} neighbours[j]} + 0.5 \right\rfloor$
  end if
  $L_i^b \leftarrow L_i^b \cup \{u\}$
end for

walls, Algorithm 5 will place all of them in the same wall.

Clearly, the proposed algorithm while keeping the number of inter wall edges low does not guarantee balanced distribution of the vertices among the walls. The problem of balanced partitioning the vertex set of a graph into $k \geq 2$ subsets with the minimum number of edges between the subsets is a generalization of the NP-hard minimum bisection problem discussed in Section 3.1. It is known as the (k, ν)-balanced partitioning problem where each subset is required to have size at most $\nu * \frac{|V|}{k}$. In a recent study, Andreev and Räcke have shown that the $(k, 1)$-balanced partitioning problem has no polynomial time approximation algorithm with finite approximation factor unless $P = NP$, and propose a polynomial time algorithm for solving the $(k, 1+\epsilon)$-balanced partitioning problem with an $O(\log^2 |V|/\epsilon^4)$ approximation ratio [3].

In order to achieve a more even distribution of vertices between the walls, i.e. to satisfy **C1**, we propose a technique which is simpler compared to the algorithm of Andreev and Räcke [3], but it runs in linear time and our computational study gives evidence that it achieves a good enough balance. Our method is the following. When computing the barycenter value b, we alternate it for giving preference to the walls with fewer number of vertices.

The implementation of such a procedure is presented in Algorithm 6, which is a generalised version of Algorithm 5. Now the wall for vertex u is computed as

$$b = \left\lfloor \frac{\sum_{j=1}^{k} j * max\{0, neighbours[j] - |L_i^j|\}}{\sum_{j=1}^{k} max\{0, neighbours[j] - |L_i^j|\}} + 0.5 \right\rfloor. \quad (1)$$

in the case $\sum_{j=1}^{k} max\{0, neighbours[j] - |L_i^j|\} > 0$. Otherwise, u is placed in the wall with the fewest number of vertices in the current layer. An example layout is shown in Figure 6.

It is easy to see that Algorithm 5 guarantees that all dummy vertices along

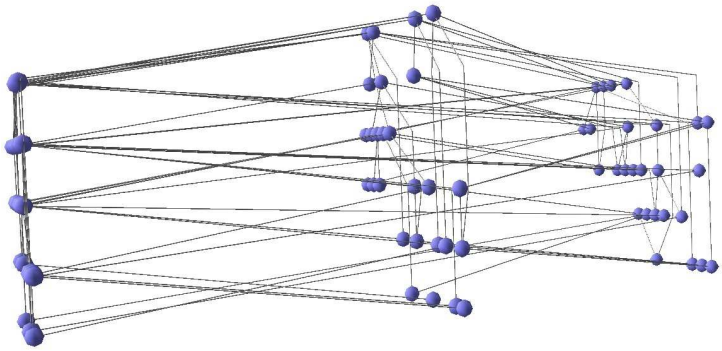


Figure 6: A 2.5D hierarchical layout with a balanced k-wall partition.

Algorithm 6 Balanced barycenter k-wall partition of layer L_i

for all $j = 1..k$ **do**
  $L_i^j \leftarrow \phi$
end for
for all $u \in L_i$ **do**
  if u is a dummy vertex **then**
    Let b be the number such that the only immediate successor of u is assigned to L_{i-1}^b.
  else
    for all $j = 1..k$ **do**
      $neighbours[j] \leftarrow |N^+(v) \cap L_{i-1}^j|$
    end for
    if $\sum_{j=1}^{k} max\{0, neighbours[j] - |L_i^j|\} > 0$ **then**
      $b \leftarrow \left\lfloor \frac{\sum_{j=1}^{k} j * max\{0, neighbours[j] - |L_i^j|\}}{\sum_{j=1}^{k} max\{0, neighbours[j] - |L_i^j|\}} + 0.5 \right\rfloor$
    else
      Let b be a number such that $|L_i^b| = min\{|L_i^1|, |L_i^2|, \ldots, |L_i^k|\}$
    end if
  end if
  $L_i^b \leftarrow L_i^b \cup \{u\}$
end for

an edge belong to the same wall. However, this is no longer guaranteed with the balancing technique in Algorithm 6. Thus, when applying the balancing technique, we first need to check whether u is dummy and if it is then assign it to the wall of its immediate successor.

The time complexity of the proposed k-wall partitioning algorithm is also $O(|V| + |E|)$ because each vertex and each edge are scanned once.

Lemma 3 *Both versions of the k-wall partition algorithm assign all dummy vertices along an edge to the same wall, and partition the vertex set of the graph into $k \geq 2$ subsets in linear time.*

4 Computational Results

In our experimental work we used 5911 DAGs from the well-known Rome graph dataset introduced by di Battista et al. in their experimental studies [8] and available at the GDToolkit website[2]. The copy of the Rome graph set we have consists of 11,530 graphs in LEDA[3] format. Since, by default, a graph in LEDA format is directed, we accepted the default direction of the edges given by the LEDA format and filtered out the graphs with directed cycles. We also filtered out the unconnected graphs leaving 5911 DAGs. The graphs have vertex count between 10 and 100. The x-axis in all the plots below represents the number of original vertices in a graph. We have partitioned all DAGs into groups by vertex count. Each group covers an interval of size 5 on the x-axis, except the last group which represents only the graphs with 100 vertices. We display the average result for each group. Partitioning the DAGs into groups by edge count reveals the same results because the DAGs typically have twice as many edges as vertices.

The objective of our computational study was to compare the different wall assignment techniques introduced in Section 3. Note that the k-wall partition techniques can be applied for two-wall partitioning as well when $k = 2$. The first part of our computational study compares the two-wall partition techniques, and the second compares k-wall partitions for different values of $k \geq 2$.

To each of the test DAGs, we applied our method (Algorithm 1) with the same algorithms for all steps except the wall-assignment step. Since the test digraphs are acyclic, we did not have to remove cycles. For the layer-assignment step, we used the network-simplex layering algorithm introduced by Gansner et al. [15]. To be able to compare the number of edge crossings between intra wall edges we had to adapt the vertex-ordering step in order to take into account the multiple walls.

For the vertex-ordering step, we applied layer-by-layer sweep with the barycenter heuristic for two-layer crossing minimization. In our extended abstracts, we considered ordering the vertices in each wall both independently from other walls and taking into account the neighbours of each vertex in other walls. Our

[2] http://www.dia.uniroma3.it/∼gdt/

[3] http://www.algorithmic-solutions.de/enleda.htm

pilot study gave some evidence that the number of crossings between intra wall edges is slightly lower when the vertices in each wall are ordered independently from other walls [20, 19]. Thus, in the present computational study, we apply the layer-by-layer sweep for each wall independently. For the computational results presented below, we did not have to perform the coordinate assignment step. The example drawings in this paper were made with applying the Brandes-Köpf algorithm for each wall independently [5].

4.1 Two-Wall Partitions

First we compare the proposed methods for partitioning the vertex set into two walls. Here we also include the two k-wall methods with $k = 2$. In the plots, we use the following two-letter references to the wall-assignment methods.

- **MB** Two Wall-partition based on minimum bisection (Algorithm 2).
- **ZZ** Zig-zag wall assignment (Algorithm 3).
- **DW** Dominating-wall assignment (Algorithm 4).
- **KW** k-wall partition (Algorithm 5).
- **BW** Balanced k-wall partition (Algorithm 6).

The plot in Figure 7 compares the standard deviation of the distribution of vertices between the walls. We divided the standard deviation for each DAG by its original vertex count (i.e. before introducing dummy vertices) in order to normalize it. Similar normalization is applied in all other plots.

As we expected **MB** and **BW** have the lowest values which means that they have the most balanced distribution of vertices between the walls. The same wall-assignment algorithms behave best in terms of distribution of intra wall edges as shown in Figure 8. Figures 7 and 8 also demonstrate that the distribution of vertices and intra wall edges is not too unbalanced for **ZZ**, **DW**, and **KW** on average. We did not particularly expect this result.

Figure 9 shows that **BW** has a significantly higher number of inter wall edges than any of the other wall-assignment algorithms. **KW** has the lowest number of inter wall edges as we expected, but it is very closely followed by **ZZ** and **DW**. This figure shows that the balancing technique employed by **BW** is too simplistic as it fails to keep the number of inter wall edges low.

Finally, Figure 10 compares the number of crossings between intra wall edges for all wall assignment methods and the 2D case (i.e. without partitioning into walls). The low number of edge crossings for **BW** is due to the high number of inter wall edges. In general, Figure 10 gives evidence that by partitioning a hierarchy into two walls by any of the proposed methods, the number crossings between intra wall edges can be reduced at least twice.

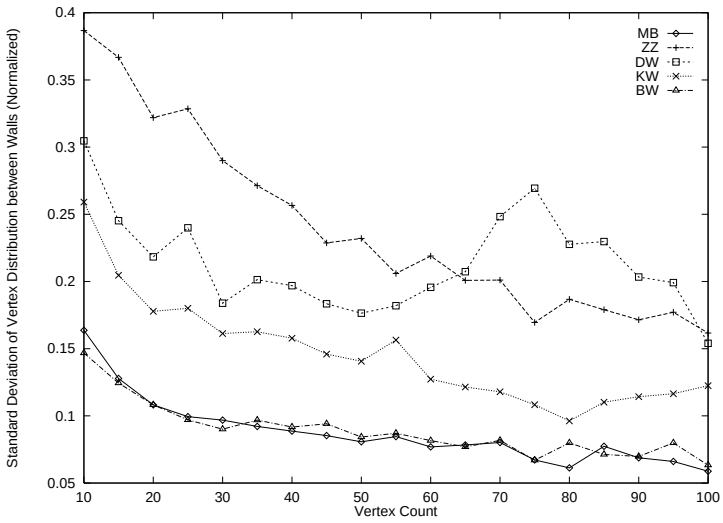


Figure 7: Standard deviation of the distribution of vertices between the walls divided by the number of original vertices.

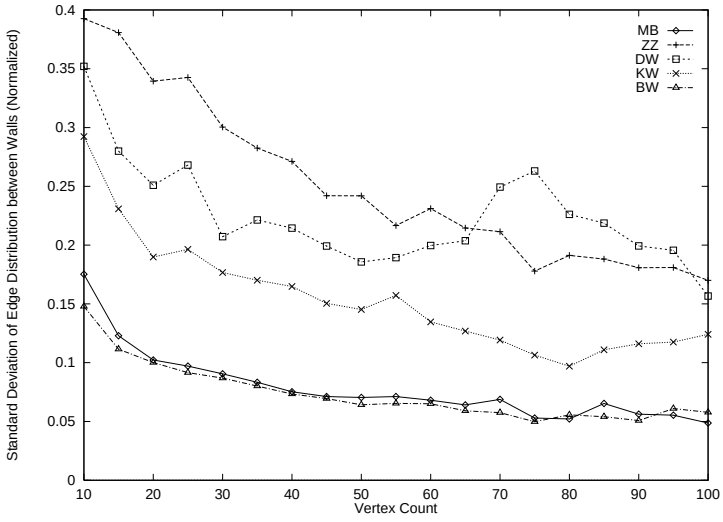


Figure 8: Standard deviation of the distribution of intra wall edges between the walls divided by the number of original edges.

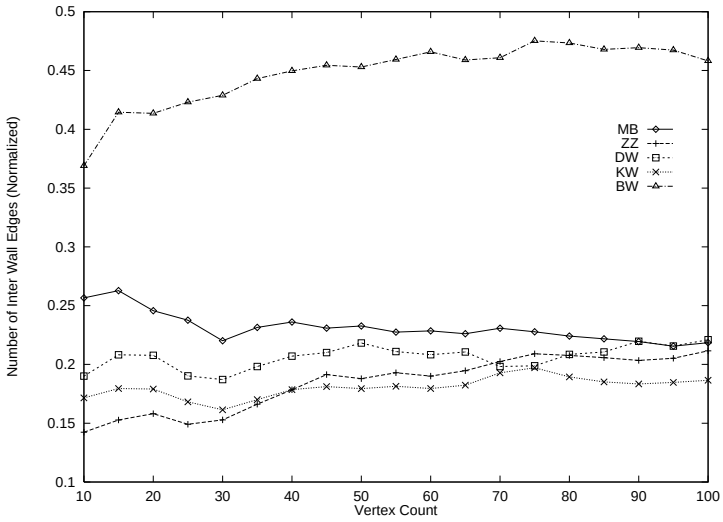


Figure 9: Number of inter wall edges divided by the number of original edges.

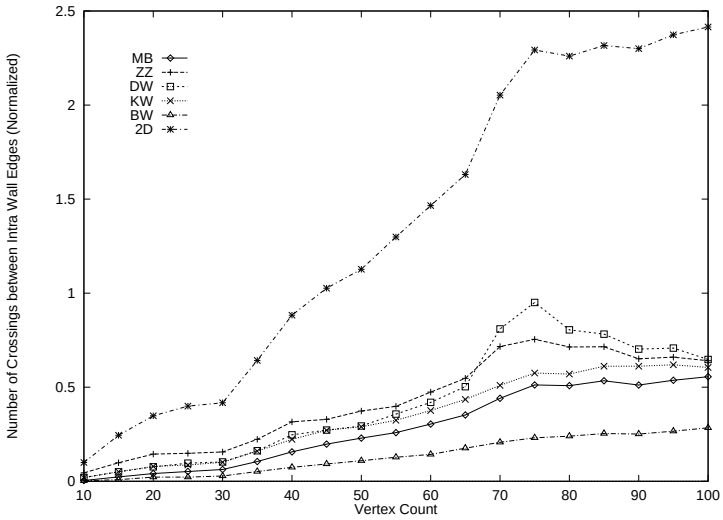


Figure 10: Number of crossings between intra wall edges divided by the number of original edges.

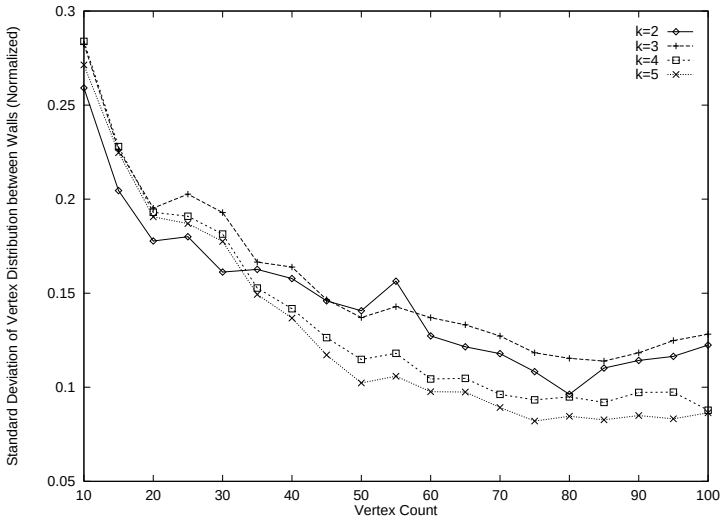


Figure 11: Standard deviation of the distribution of vertices between the walls divided by the number of original vertices.

4.2 k-Wall Partitions

Our study for $k = 2$ as well as our practical experience with $k > 2$ showed evidence that **BW** while achieving balanced partitioning has a higher number of inter wall edges than **KW**. In this section we study the behavior only of **KW** for different values of k.

Figures 11 and 12 demonstrate that the distribution of vertices and intra wall edges does not depend on the number of walls too heavily. It can be expected that the distribution becomes more balanced with increasing the number of walls. However, with the growth of the number of walls, grows the number of inter wall edges and their total span, as seen in Figures 13 and 14. Based on this evidence, we would suggest partitioning of digraphs with up to 100 vertices into no more than four walls. Figure 15 shows the maximum number of edges between adjacent walls, i.e. the maximum edge density between the walls. It can be observed that it does not depend on the number of walls for the considered dataset and values of k.

We also experimented with partitioning the vertex set of each DAG into walls whose count depends on the size of the vertex set. For this purpose, we set the number of walls at $\max\{2, n/30\}$, where n is the total number of original and dummy vertices in the DAG. Figure 16 presents the average number of walls for our dataset. Figure 17 shows an interesting result. It compares the number of crossings between intra wall edges in **KW** to the number of edge crossings in 2D layouts. While the average number of edge crossings grows with the growth of the size of the graph in the 2D layouts, it is virtually constant in the **KW** layouts when the number of walls depends on the size of the graph.

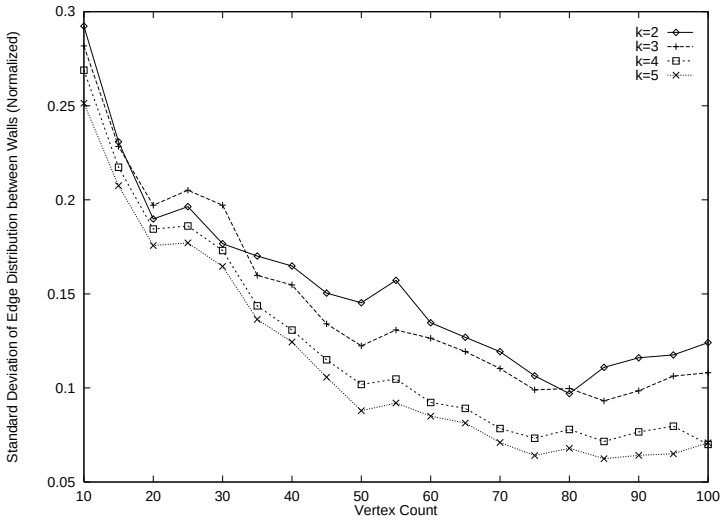


Figure 12: Standard deviation of the distribution of intra wall edges between the walls divided by the number of original edges.

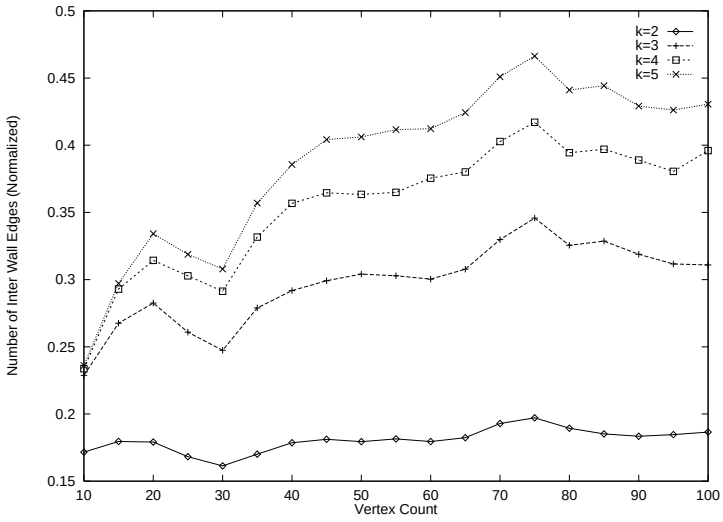


Figure 13: Number of inter wall edges divided by the number of original edges.

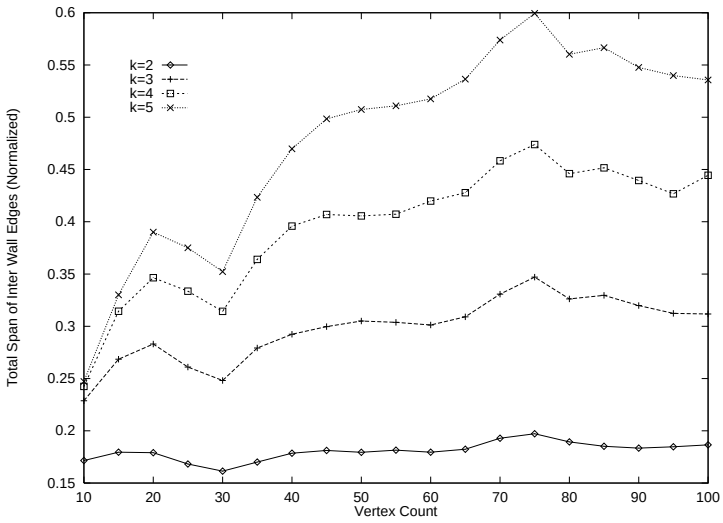


Figure 14: Total span of the inter wall edges divided by the number of original edges.

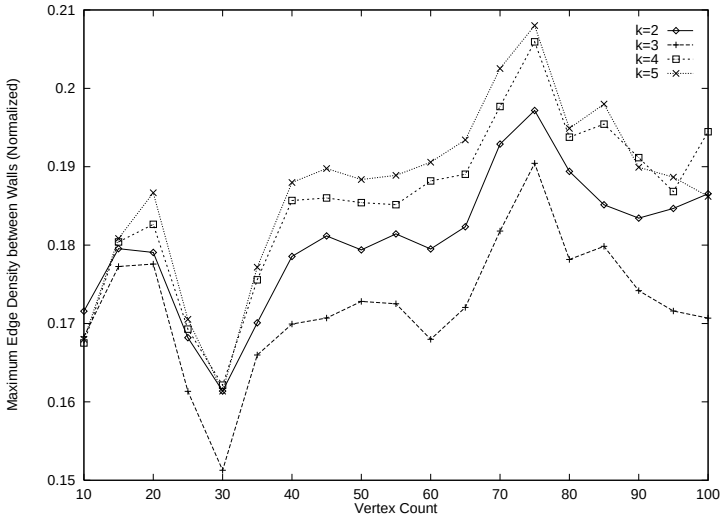


Figure 15: Maximum number of edges between adjacent walls divided by the number of original edges.

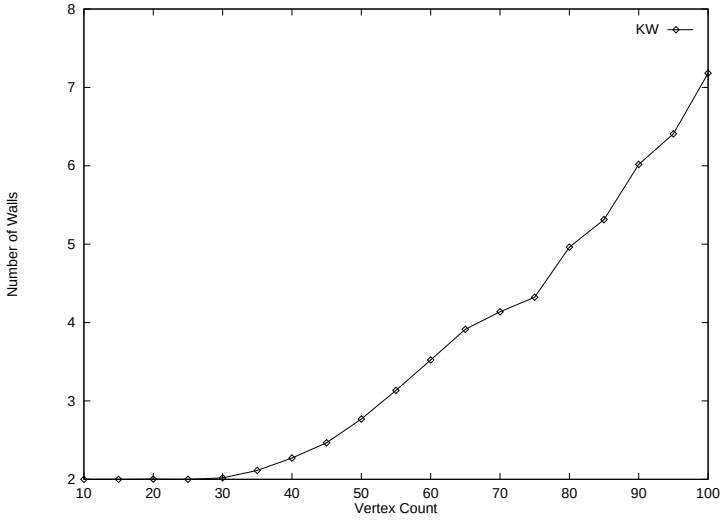


Figure 16: Variable number of walls in **KW** layouts dependent on the size of the DAG.

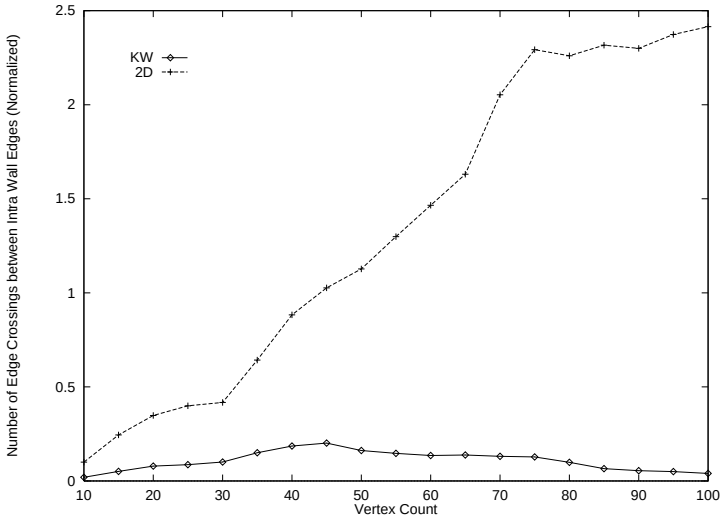


Figure 17: Number of crossings between intra wall edges in 2D layouts and in **KW** layouts with number of walls dependent on the size of the DAG.

5 Conclusions

We introduce a framework for 2.5D hierarchical graph drawing based on the Sugiyama method. We propose to partition the vertex set into $k \geq 2$ parallel planes, called walls; each wall containing a 2D drawing of a layered digraph. This is done by introducing an additional wall assignment step into the Sugiyama method after the layer assignment step.

We propose five alternative fast wall-assignment algorithms and evaluate them in an extensive computational study with a large dataset. The choice of a particular wall assignment algorithm may highly depend on the interaction and navigation techniques used for visualizing the layout [2]. We were able to show evidence that four of the proposed wall-assignment methods perform well in practice in terms of various properties of the corresponding 2.5D layouts.

Additional improvement in the proposed 2.5D layouts may come from vertex-ordering and coordinate-assignment algorithms specifically developed for 2.5D hierarchical layouts. Another direction of future research might be related to defining new optimization problems arising from 3D drawing aesthetic criteria.

Acknowledgments

The authors would like to thank Michael Forster for the valuable discussion and suggestions, and to the whole GEOMI team for implementing our algorithms.

References

[1] A. Ahmed, T. Dwyer, M. Forster, X. Fu, J. Ho, S.-H. Hong, D. Koschützki, C. Murray, N. S. Nikolov, R. Taib, A. Tarassov, and K. Xu. GEOMI: GEOmetry for Maximum Insight. In P. Healy and N. S. Nikolov, editors, *Graph Drawing, 13th International Symposium, GD 2005*, volume 3843 of *Lecture Notes in Computer Science*, pages 468–479. Springer-Verlag, 2005.

[2] A. Ahmed and S.-H. Hong. Navigation techniques for 2.5D graph layout. In *Proceedings of APVIS 2007 (Asia-Pacific Symposium on Information Visualisation)*. IEEE, to appear.

[3] K. Andreev and H. Räcke. Balanced graph partitions. In *Proc. of the 16th SPAA*, pages 120–124, 2004.

[4] T. Biedl, T. Thiele, and D. R. Wood. Three-dimensional orthogonal graph drawing with optimal volume. In J. Marks, editor, *Graph Drawing: Proceedings of 8th International Symposium, GD 2000*, volume 1984 of *Lecture Notes in Computer Science*, pages 284–295. Springer-Verlag, 2001.

[5] U. Brandes and B. Köpf. Fast and simple horizontal coordinate assignment. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing: Proceedings of 9th International Symposium, GD 2001*, volume 2265 of *Lecture Notes in Computer Science*, pages 31–44. Springer-Verlag, 2002.

[6] R. Cohen, P. Eades, T. Lin, and F. Ruskey. Three dimensional graph drawing. *Algorithmica*, 17(2):199–208, 1996.

[7] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, 1999.

[8] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Computational Geometry: Theory and Applications*, 7:303–316, 1997.

[9] V. Dujmović, P. Morin, and D. R. Wood. Path-width and three-dimensional straight-line grid drawing of graphs. In M. Goodrich and S. Koburov, editors, *Graph Drawing: Proceedings of 10th International Symposium, GD 2002*, volume 2528 of *Lecture Notes in Computer Science*, pages 42–53. Springer-Verlag, 2002.

[10] P. Eades and K. Sugiyama. How to draw a directed graph. *Journal of Information Processing*, 13(4):424–437, 1990.

[11] P. Eades, A. Symvonis, and S. Whitesides. Three dimensional orthogonal graph drawing algorithms. *Discrete Applied Mathematics*, 103:55–87, 2000.

[12] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM J. Comput*, 31(4):1090–1118, 2002.

[13] U. Feige, R. Krauthgamer, and K. Nissim. Approximating the minimum bisection size. In *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC)*, pages 530–536, 2000.

[14] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *Proceedings of the 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[15] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, March 1993.

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

[17] A. Garg and R. Tamassia. GIOTTO: A system for visualizing hierarchical structures in 3D. In S. North, editor, *Graph Drawing: Symposium on Graph Drawing, GD '96*, volume 1190 of *Lecture Notes in Computer Science*, pages 193–200. Springer-Verlag, 1997.

[18] S.-H. Hong. Drawing graphs symmetrically in three dimensions. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing: Proceedings of 9th International Symposium, GD 2001*, volume 2265 of *Lecture Notes in Computer Science*, pages 189–204. Springer-Verlag, 2002.

[19] S.-H. Hong and N. S. Nikolov. Hierarchical layouts of directed graphs in three dimensions. In P. Healy and N. S. Nikolov, editors, *Graph Drawing, 13th International Symposium, GD 2005*, volume 3843 of *Lecture Notes in Computer Science*, pages 251–261. Springer-Verlag, 2005.

[20] S.-H. Hong and N. S. Nikolov. Layered drawings of directed graphs in three dimensions. In S.-H. Hong, editor, *Information Visualisation 2005: Asia-Pacific Symposium on Information Visualisation (APVIS2005)*, volume 45, pages 69–74. CRPIT, 2005.

[21] B. W. Kernigham and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–308, 1970.

[22] D. Ostry. Some three-dimensional graph drawing algorithms. Master's thesis, University of Newcastle, 1996.

[23] J. Pach, T. Thiele, and G. Toth. Three-dimensional grid drawings of graphs. In G. Di Battista, editor, *Graph Drawing: Proceedings of 5th International Symposium, GD 1997*, volume 1353 of *Lecture Notes in Computer Science*, pages 47–51. Springer-Verlag, 1998.

[24] H. Saran and V. V. Vazirani. Finding k cuts within twice the optimal. *SIAM J. Comput.*, 24(1):101–108, 1995.

[25] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transaction on Systems, Man, and Cybernetics*, 11(2):109–125, February 1981.

[26] C. Ware and G. Franck. Viewing a graph in a virtual reality display is three times as good as a 2D diagram. In *IEEE Conference on Visual Languages*, pages 182–183, 1994.