Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 13, no. 3, pp. 319–347 (2009)

DAGmaps: Space Filling Visualization of Directed Acyclic Graphs

Vassilis Tsiaras[1,2] *Sofia Triantafilou*[1,2] *Ioannis G. Tollis*[1,2]

[1]Institute of Computer Science,
Foundation for Research and Technology-Hellas, Vassilika Vouton, P.O. Box 1385, Heraklion, GR-71110 Greece
[2]Department of Computer Science, University of Crete, P.O. Box 2208, Heraklion, Crete, GR-71409 Greece

Abstract

Gene Ontology information related to the biological role of genes is organized in a hierarchical manner that can be represented by a directed acyclic graph (DAG). Space filling visualizations, such as the treemaps, have the capacity to display thousands of items legibly in limited space via a two-dimensional rectangular map. Treemaps have been used to visualize the Gene Ontology by first transforming the DAG into a tree. However this transformation has several undesirable effects such as producing trees with a large number of nodes and scattering the rectangles associated with the duplicates of a node around the display rectangle. In this paper we introduce the problem of visualizing a DAG with space filling techniques without converting it to a tree first, we present two special cases of the problem, and we discuss complexity issues.

Submitted:	Reviewed:	Revised:	Accepted:
January 2008	April 2008	August 2008	December 2008
	Final:	Published:	
	January 2009	November 2009	

Article type:	Communicated by:
Regular paper	S.-H. Hong and T. Nishizeki

This work was supported in part by INFOBIOMED code: IST-2002-507585 and the Greek General Secretariat for Research and Technology under Program "ARISTEIA", Code 1308/B1/3.3.1/317/12.04.2002.

E-mail addresses: tsiaras@ics.forth.gr (Vassilis Tsiaras) striant@ics.forth.gr (Sofia Triantafilou) tollis@ics.forth.gr (Ioannis G. Tollis)

1 Introduction

The Gene Ontology project (GO) [1], provides a controlled vocabulary to describe gene and gene product attributes in an organism. The GO is the union of three ontologies, each representing a key concept in molecular biology: the Molecular Function of gene products; their role in multi-step Biological Processes; and their localization to Cellular Components. The building blocks of the three ontologies are the terms which consist of a numerical identifier, a name and a number of attributes such as a definition. The ontologies are continuously updated but since they change very slowly their structure can be considered almost constant. At the time that this paper was written the three ontologies contained 8464, 15841 and 2253 terms respectively. The ontologies are structured as a directed acyclic graph (DAG) where the set of vertices is the set of terms and an edge is a relationship from a more specialized term to a less specialized term. GO terms can be linked by five type of relationships: *is_a*, *part_of*, *regulates*, *positively_regulates* or *negatively_regulates*.

Due to its huge size, visualizing the whole GO with the usual node-link representation leads to visual clutter. The reason for this clutter is that, the node-link representations do not make optimal use of the available space since most of the pixels are used for the background. On the other hand, space filling techniques make optimal use of the available space and have the capacity to show thousands of items legibly. At the core of a space filling visualization is a layout function that takes as arguments a list of k positive numbers $\{x_1, x_2, \ldots, x_k\}$ and a rectangle R and returns a partition of R into rectangles $R_1, R_2, \ldots, R_k$, where $area(R_i)$ is proportional to x_i. The number of possible partitions is huge and finding a solution such as to minimize the size of the perimeters of the rectangles is NP-complete [3]. Heuristic algorithms produce partitions with desirable properties in reasonable time. For example, the squarified layout, introduced by Bruls et al. [5], strives to produce rectangles with aspect ratio as close as possible to one, where $aspect\_ratio = max(width/height,\ height/width)$.

In the case of treemaps the nodes of a tree are visualized as rectangles whose area is proportional to a numeric attribute of the nodes with the property that the value of the attribute in a parent node is equal to the sum of the values of its children nodes. The rectangle that represents the root of a tree is partitioned into rectangles representing its children and the algorithm is repeated recursively

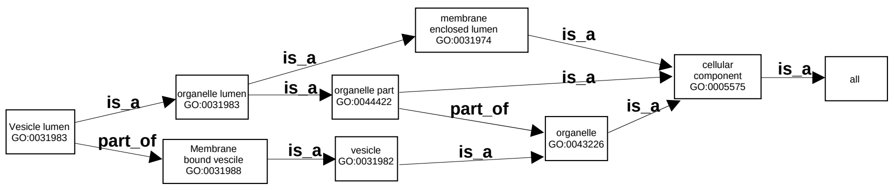

membrane enclosed lumen GO:0031974
is_a
cellular component GO:0005575
is_a
all
is_a
organelle lumen GO:0031983
is_a
organelle part GO:0044422
is_a
is_a
Vesicle lumen GO:0031983
part_of
organelle GO:0043226
is_a
part_of
Membrane bound vescile GO:0031988
is_a
vesicle GO:0031982
is_a

Figure 1: A small subgraph of the GO DAG.

for each node of the tree [4, 5, 15].

In a treemap the hierarchy structure is presented using several approaches including the nested [10, 15], the cascaded [12, 13], and the cushion [18] presentations.

Treemaps have been used to visualize compound graphs that contain both hierarchical (rooted tree) relations and adjacency relations [9]. Space filling techniques are used for the hierarchical or inclusion relations and lines or curves for the adjacency relations.

In the context of GO, treemaps have been used to visualize microarray data, where each gene transcript is assigned all possible paths that start from it and terminate to the most general term (the "all" term) of GO [2]. Symeonidis et al. [16] proposed to decompose the complete GO DAG into a tree by duplicating the vertices with many incoming edges, and then to use a treemap algorithm to visualize the tree (Figure 8). The duplication of a vertex however triggers the duplication of all of its out-neighbors. Therefore the transformation of a DAG

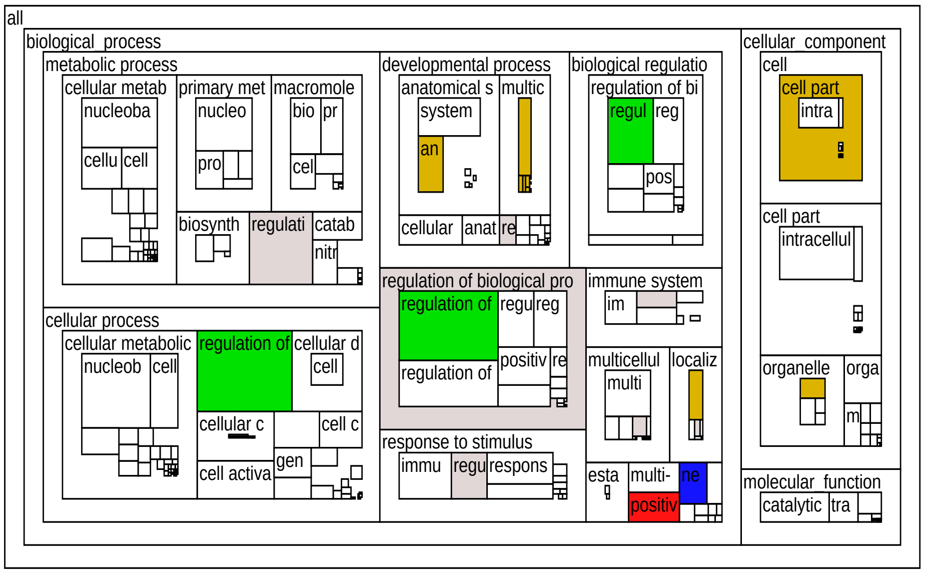


Figure 2: Transforming the GO DAG into a tree and then drawing it as a treemap. The first five layers from the root are shown. The tree structure is visualized via nesting. The multiple copies of the node "regulation of cellular process" are shown with green background color. The color of nodes refers to the relationship between a GO term and its parent GO term. We use white color for *is_a* relationship, light brown color for *part_of*, gray color for *regulates*, red color for *positively_regulates* and blue color for *negatively_regulates*. E.g., "metabolic process" *is_a* "biological process" which *is_a* "all". Also "cell part" *is_a* "cellular component" and *part_of* "cell".

into a tree leads to trees with (potentially exponentially) many more nodes than the original DAG. At the time that this paper was written the initial GO DAG had 26558 vertices, while the produced equivalent tree had 872460 nodes. Another drawback of duplicating the vertices is that the rectangles associated with the multiple replicas of a vertex are scattered around the display rectangle (Figure 2).

In this paper we introduce the problem of drawing a DAG using space filling techniques without converting it to a tree first. We consider several variations of the problem, we present some characterizations of simple families of DAGs that admit such a drawing, and provide complexity results for the general problem.

2 Problem Definition

2.1 Preliminaries

Suppose that $G = (V, E)$ is a directed acyclic graph (DAG) with $n = |V|$ vertices and $m = |E|$ edges. A path of length k from a vertex u to a vertex w is a sequence $v_0, v_1, v_2, \ldots, v_k$ of vertices such that $u = v_0$, $w = v_k$, and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \ldots, k$. There is always a zero-length path from u to u. If there is a path p from u to w, we say that w is reachable from u via p and we write $u \overset{p}{\rightsquigarrow} w$.

A layering of G is a partition of V into subsets $L_1, L_2, \ldots, L_h$, such that if $(u, v) \in E$, where $u \in L_i$ and $v \in L_j$, then $i > j$. A DAG with a layering is a layered DAG. The span of an edge (u, v) with $u \in L_i$ and $v \in L_j$ is $i - j$. The DAG is proper if no edge has a span greater than one. A DAG G can be made proper by replacing each long edge (u, v) of span $k > 1$ with a path $u = v_1, v_2, \ldots, v_k = v$, adding the dummy vertices $v_2, \ldots, v_{k-1}$ [6].

If $e = (u, v) \in E$ is a directed edge, we say that e is incident from u (or outgoing from u) and incident to v (or incoming to v); vertex u is the origin of e and vertex v is the destination of e. The origin of e is denoted by $orig(e)$ and the destination of e by $dest(e)$. For every vertex $u \in V$, $N^+(u) = \{v \mid (u, v) \in E\}$ and $N^-(u) = \{v \mid (v, u) \in E\}$ are the sets of out-neighbors and in-neighbors of vertex u, respectively. Analogously, $\Gamma^+(u) = \{e \in E \mid orig(e) = u\}$ and $\Gamma^-(u) = \{e \in E \mid dest(e) = u\}$ are the sets of edges incident from and to vertex u, respectively. Finally, we denote the set of edges incident from the nodes of a layer L_i, $i \in \{2, \ldots, h\}$ by E_i ($E_i = \cup_{u \in L_i} \Gamma^+(u)$).

2.2 Drawing Constraints

Treemaps display a tree hierarchy via the inclusion invariant. Namely, the drawing rectangle of any node (different from the root) is included within the drawing rectangle of its parent. When the graph is a DAG, the above invariant should be replaced by the invariant that the drawing rectangle of any vertex is included within the union of the rectangles of its in-neighbors. Apart from this invariant it is plausible to assume that the drawing rectangles of two vertices do not overlap when each node is not reachable from the other and that the drawing rectangle of a vertex is covered by the drawing rectangles of its out-neighbors. Another observation is that in DAGs an edge may be visualized as a rectangle which is contained in the intersection of the origin and the destination vertex rectangles.

Let R_u denote the drawing region of a vertex $u \in V$ and similarly R_e denote the drawing region of an edge $e \in E$. Then the above invariant and assumptions are summarized in Definition 1.

Definition 1 (DAGmap drawing) *A DAGmap drawing of a DAG $G = (V, E)$ is a space filling visualization of G that satisfies the following drawing constraints:*

- *B1. Every vertex is drawn as a rectangle (R_u is a rectangle for every $u \in V$).*
- *B2. The union of the rectangles of the sources of G is equal to the initial drawing rectangle ($R = \cup_{s \in S} R_s$, where $S \subset V$ is the set of sources of G).*
- *B3. Every edge is drawn as a rectangle that has non-zero area and which is contained in the intersection of the origin and destination vertex rectangles ($\forall e = (u, v) \in E$, R_e is a rectangle, $R_e \subset R_u \cap R_v$ and $area(R_e) \neq 0$).*
- *B4. For every pair of edges $e_1 = (u_1, v_1), e_2 = (u_2, v_2) \in E$, $e_1 \neq e_2$, such that u_1 is not reachable from v_2 and u_2 is not reachable from v_1, the rectangles R_{e_1} and R_{e_2} do not overlap ($area(R_{e_1} \cap R_{e_2}) = 0$).*
- *B5. The rectangle of every non-source vertex $u \in V$ is equal to the union of the rectangles of edges incident to u ($R_u = \cup_{e \in \Gamma^-(u)} R_e$).*
- *B6. The rectangle of every non-sink vertex $u \in V$ is equal to the union of the rectangles of edges incident from u ($R_u = \cup_{e \in \Gamma^+(u)} R_e$).*

From constraints B1-B6 it is trivial to prove that:

Proposition 1 *In a DAGmap drawing of a DAG G the following hold:*

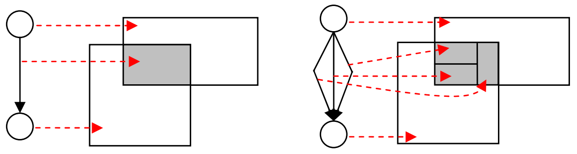
- *a) The rectangle of every non-source vertex $u \in V$ is contained in the union of rectangles of its in-neighbors ($R_u \subset \cup_{v \in N^-(u)} R_v$).*
- *b) The rectangle of every non-sink vertex $u \in V$ is covered by the rectangles of its out-neighbors ($R_u \subset \cup_{v \in N^+(u)} R_v$).*
- *c) For every pair of vertices $u, v \in V$ if there is no path from u to v and from v to u then their rectangles R_u, R_v do not overlap ($area(R_u \cap R_v) = 0$).*

The drawing rules of Definition 1 are quite general since they do not constrain the area of the sink vertices, or how the area of a vertex is distributed to its incoming edges. To simplify the analysis of the problem we constrain these two parameters by making the following assumptions.

Definition 2 (Additional drawing constraints)

A1. The sink vertices are drawn in equal area rectangles.

A2. The rectangles of the edges incident to a vertex have equal areas (For every non-source vertex u and every $e \in \Gamma^-(u)$, $area(R_e) = \frac{area(R_u)}{|\Gamma^-(u)|}$).



(a) Edge drawing (b) Drawing of parallel edges

Figure 3: An example where an edge rectangle is the intersection of the origin and destination vertex rectangles. In the case of multigraphs, if there are k parallel edges between an origin vertex and a destination vertex, the intersection rectangle is arbitrarily partitioned into k equal area rectangles.

In real applications, we may choose to draw only vertex rectangles, only edge rectangles, or both. We usually draw edge rectangles when the DAG has multiple edges or when the edges carry out important information such as the type of relationship between two vertices. See Figure 4 for an example.

Having defined the drawing rules, we can define the following problems:

1. Given a DAG G_1, does G_1 admit a DAGmap?

2. In case that the answer to the first problem is negative, what is the minimum number of vertex duplications that are needed to transform G_1 into a DAG G_2 that admits a DAGmap?

2.3 Examples and Counter-Examples of DAGs that Admit a DAGmap

Examples of DAGs that admit a DAGmap appear in Figure 5. From the counter-example of Figure 6 we see that there are DAGs that do not admit a DAGmap drawing. The DAG in Figure 6(a) cannot be drawn due to adjacency constraint violation. The first-layer vertices e, f, g, h, i, j constrain the pairs of second-layer vertices $\{a, b\}$,$\{a, c\}$,$\{a, d\}$,$\{b, c\}$,$\{b, d\}$,$\{c, d\}$ to be drawn in adjacent rectangles. However we cannot have such a configuration. In this case in order to draw the DAG we can either duplicate one of the vertices e, f, g, h, i, j or relax some of constraints of Definition 1, as we did in Figure 6(b) where we relaxed constraint B1. However, allowing the set of possible drawings of a vertex to include any simply connected region of the plane complicates the problem without offering a guarantee that a DAG will admit a DAGmap. A counter-example, based on the four color (map coloring) theorem, is a two-layer DAG having five second-layer vertices and ten first-layer vertices (one sink for every pair of sources).

2.4 Vertex Duplication

Usually, a DAG encountered in practice does not admit a DAGmap. In this case we should relax one or more of constraints B1-B6, A1-A2 or change the form of the DAG. Symeonidis et al. [16] chose to transform the DAG into a forest of trees by performing vertex duplications. An example of a vertex duplication is shown in Figure 7, where after the creation of two replicas of vertex h the DAG of Figure 6(a) is transformed into a new DAG which admits a drawing.

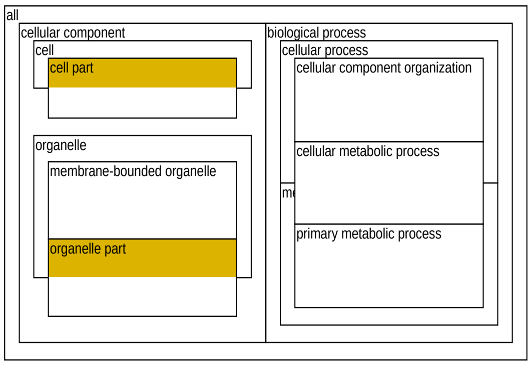
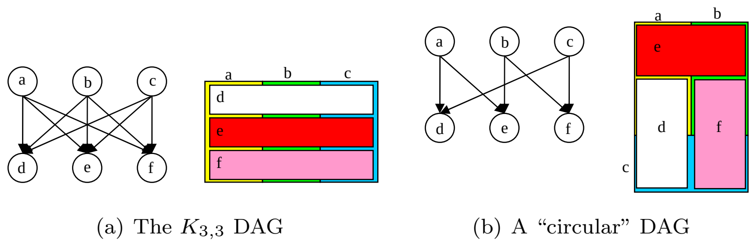
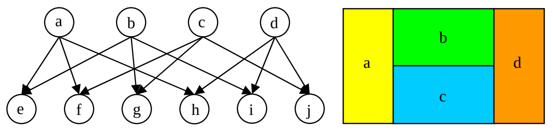

all
cellular component
cell
cell part
organelle
membrane-bounded organelle
organelle part
biological process
cellular process
cellular component organization
cellular metabolic process
me
primary metabolic process

Figure 4: In this example a subgraph of the GO DAG is drawn. The color refers to the relationship between two GO terms. We use white color for *is_a* relationship and light brown color for *part_of* relationship. The term “cellular metabolic process” *is_a* “cellular process” and *is_a* “metabolic process”. The term “cell part” *is_a* “cellular component” and *part_of* “cell”. (Compare the visualization of “cell part” between this figure and Figure 2.)

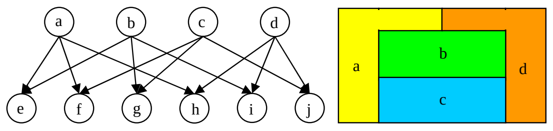


(a) The $K_{3,3}$ DAG (b) A “circular” DAG

Figure 5: Examples of DAGs that admit DAGmaps.



(a) Adjacency constraints violation



(b) Relaxing constraint B1

Figure 6: Example of a DAG that cannot have a drawing that satisfy all constraints B1-B6. However by relaxing constraint B1 the DAG admits a DAGmap.

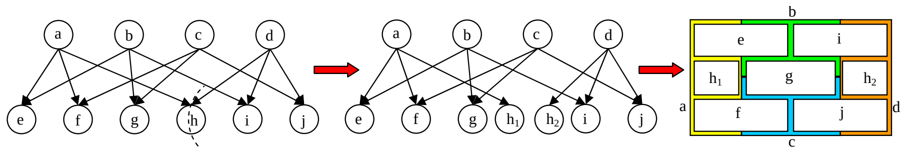


Figure 7: After the duplication of vertex h the DAG of Figure 6(a) is transformed into a DAG that admits a DAGmap.

2.5 Exponential Increase in the Number of Vertices

Transforming a DAG G into a tree via vertex duplication (see Figure 8) guarantees the admissibility of the drawing but may lead to an exponential increase in the number of vertices. In a worst case scenario, DAG G has n vertices arranged in n layers (Figure 9). For simplicity we use the same numbering for vertices

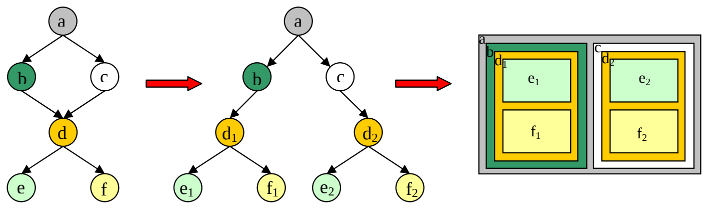


Figure 8: Example of transforming a DAG into a tree and then drawing it as a treemap.

and for layers. DAG G has $n \cdot (n-1)/2$ directed edges. There is a directed edge from every vertex i to every vertex j with $i > j$. Suppose that G is transformed into a tree having T_n nodes. Then the relation between n and T_n is:
$T_1 = 1 = 2^0$, $T_2 = 1 + T_1 = 2^1$, $T_3 = 1 + T_2 + T_1 = 2^2$
And by induction on n:
$T_n = 1 + T_{n-1} + T_{n-2} + \ldots + T_1 = T_{n-1} + T_{n-1} = 2 \cdot T_{n-1} = 2 \cdot 2^{n-2} = 2^{n-1}$

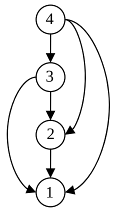


Figure 9: A DAG with four vertices, having the structure described in the worst case scenario.

3 Special Cases

We continue by considering two special cases. The first case is based on a restricted form of DAGs, the second on a restricted form of DAGmaps.

3.1 Two Terminal Series Parallel Digraphs

A Two Terminal Series Parallel (TTSP) digraph is recursively defined as follows [6, 17].

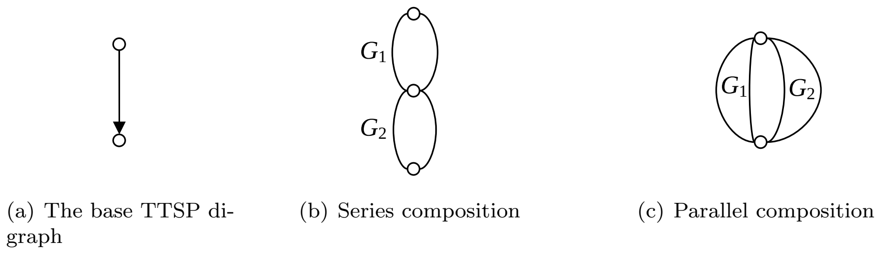
Definition 3 (Two Terminal Series Parallel digraphs)

i) A digraph consisting of two vertices joined by a single edge is TTSP (the base graph);

ii) If G_1 and G_2 are TTSP digraphs, so is the digraph obtained by either of the following operations:

- *a) Series composition: identify the sink of G_1 with the source of G_2.*
- *b) Parallel composition: identify the source of G_1 with the source of G_2 and the sink of G_1 with the sink of G_2.*

A TTSP digraph G is naturally associated with a rooted binary tree T, which is called the decomposition tree (or parse tree) of G, and which provides information on how the graph G is constructed using series and parallel compositions.



(a) The base TTSP digraph
(b) Series composition
(c) Parallel composition

Figure 10: Recursive definition of a TTSP digraph.

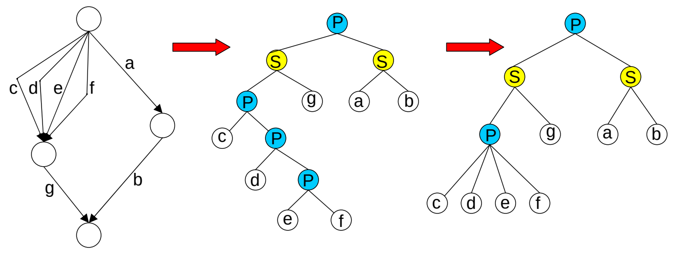


Figure 11: Given a TTSP digraph G, we construct its decomposition tree by performing parallel and series reductions. Then we merge the adjacent P-nodes.

Definition 4 *The decomposition tree T of a TTSP digraph $G = (V, E)$ has three types of nodes: S-nodes, P-nodes and Q-nodes. The leaves of T are Q-nodes and represent base graphs of G. The internal nodes are either P-nodes or S-nodes. T is defined recursively as follows:*

i) If G is a base graph, then T consists of a single Q-node.

ii) If G is created by a parallel composition of TTSP digraphs G_1 and G_2, let T_1 and T_2 be the decomposition trees of G_1 and G_2 respectively, then the root of T is a P-node and has subtrees T_1 and T_2.

iii) If G is created by a series composition of TTSP digraphs G_1 and G_2, where the sink of G_1 is identified with the source of G_2, let T_1 and T_2 be the decomposition trees of G_1 and G_2, respectively, then the root of T is an S-node and has left subtree T_1 and right subtree T_2.

The decomposition tree, which is not unique since several parallel compositions may be combined in different ways and similarly several series compositions, has $O(m)$ nodes and can be computed in $O(m)$ time as a by-product of the series and parallel reduction steps of the TTSP recognition algorithm proposed by Valdes et al. [17]. In order to use the decomposition tree as input to a DAGmap drawing algorithm the neighboring P-nodes are merged to a single node such that the resulting tree may have P-nodes of out-degree larger than two.

The TTSP recognition algorithm [17] maintains a list of vertices that initially includes all vertices except the source and the sink. The algorithm proceeds by removing any vertex v from this list and performing as many parallel reductions on the edges incident to (from) it as it is possible before either leaving the vertex with a single entering edge and a single exiting edge, or discovering that the vertex still has at least two distinct in-neighbors or two distinct out-neighbors. In the first alternative, the vertex is removed by a series reduction and the two vertices adjacent to it added to the unsatisfied list if they are not there already. This process is repeated until the unsatisfied list becomes empty, at which point the same process is applied to the source and the sink (in order to eliminate any multiple edges between them) before stopping. The unsatisfied list becomes empty, either because all vertices (except source and sink) have been deleted by series reductions or because every remaining vertex has two distinct in-neighbors or two distinct out-neighbors. In the first case the DAG is TTSP; in the second it is not.

The following algorithm finds a DAGmap drawing of a TTSP digraph.

Algorithm 1 TTSP DAGmap drawing
Input: TTSP digraph G and a rectangle R
Output: A DAGmap drawing of G

1. Construct the decomposition tree T of G [17] and merge the neighboring P-nodes. In the resulting tree, P-nodes may have more than two children.
2. S-nodes of T are associated with vertices of G as follows. If an S-node is created as a result of series reduction between two edges (v, u) and (u, w) of G, then associate this S-node with vertex u of G. All vertices of G, apart from the source and the sink, have a corresponding S-node in T.
3. Assign sizes to nodes of the decomposition tree T.
4. Assign rectangle R to the root node of T.
5. Let u be the current node of T and R_u the rectangle assigned to it.
6. If u is an S-node then
   a) Let u_L and u_R be the left and right children of node u.
   b) Assign rectangles $R_L = R_u$ and $R_R = R_u$ to nodes u_L and u_R respectively.
   c) Recursively repeat the procedure from step 5 for nodes u_L and u_R.
7. If u is a P-node then
   a) Let $u_1, u_2, \ldots, u_k$ be the children of u and let $x_1, x_2, \ldots, x_k$ be their sizes.

b) Call a layout function with input the rectangle R_u and the sizes $x_1, x_2, \ldots, x_k$ to find a partition of R_u into rectangles $R_1, R_2, \ldots, R_k$ where $area(R_i)$ is proportional to x_i.

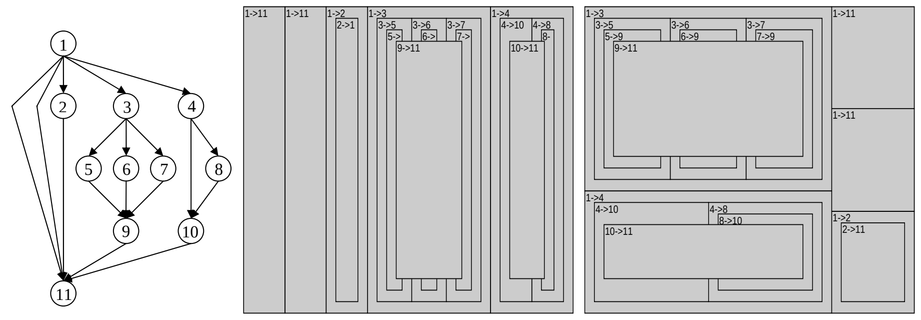
c) for $i = 1 : k$

- assign rectangle R_i to node u_i and recursively repeat the procedure from step 5 for node u_i.

8. When the above recursive procedure finishes, the rectangle assigned to a Q-node of T is also assigned to the associated edge of G. The rectangle assigned to an S-node of T is also assigned to the associated vertex of G.

9. Assign rectangle R to the source and sink of G.

10. Draw vertex and/or edge rectangles according to a priority criterion. □



(a) A TTSP

(b) A DAGmap drawing using a layout that slices the rectangles in the vertical direction

(c) A DAGmap drawing using the squarified layout proposed by Bruls et al. [5]

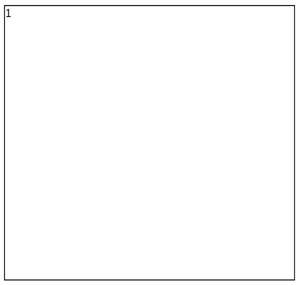
Figure 12: Example of a TTSP digraph DAGmap drawing. Only edge rectangles are shown. The hierarchy structure is shown via nesting. For an example where only vertex rectangles are shown see Figure 13(i).

The tricky part of the algorithm is how to assign sizes to nodes of the decomposition tree. If u is an internal node of the decomposition tree, and $u_1, u_2, \ldots, u_k$ are its children then the constraints are:

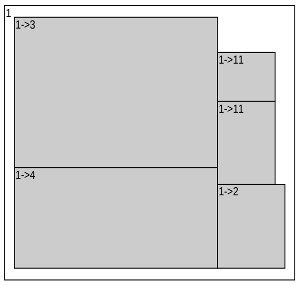
1. If u is an S-node then $size(u) = size(u_1) = size(u_2)$.

2. If u is a P-node then $size(u) = size(u_1) + \ldots + size(u_k)$.

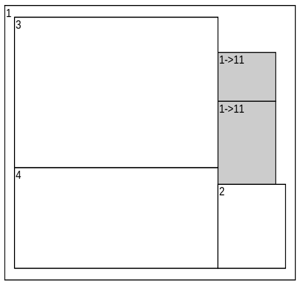
These constraints are not sufficient for a unique solution, and there is some freedom on the choice of the size of some leaf nodes. In the examples of Figure 12 the sizes were calculated using the additional assumption that if among the children nodes of a P-node there are some Q-nodes then all have the same size.



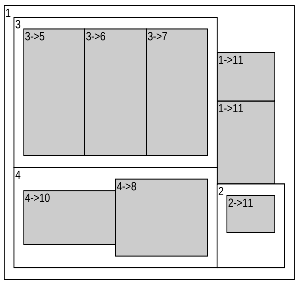
(a) Vertex 1



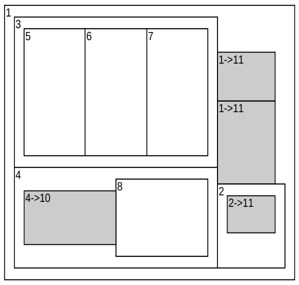
(b) Outgoing edges of vertex 1



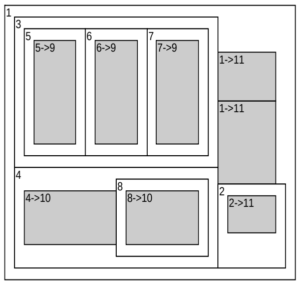
(c) Vertices 2, 3 and 4



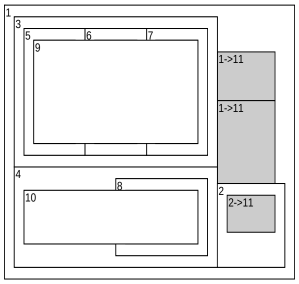
(d) Outgoing edges of vertices 2, 3 and 4



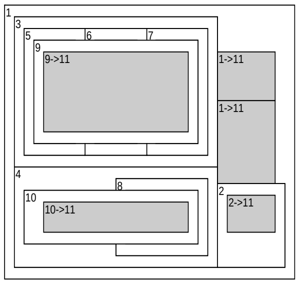
(e) Vertices 5, 6, 7 and 8



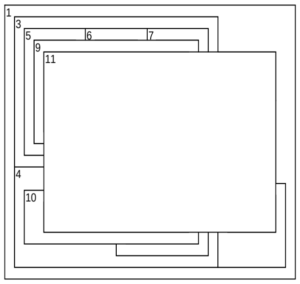
(f) Outgoing edges of vertices 5, 6, 7 and 8



(g) Vertices 9 and 10



(h) Outgoing edges of vertices 9 and 10



(i) Vertex 11

Figure 13: Step by step drawing of the DAG of Figure 12. The vertex rectangles are shown with white color while the edge rectangles are shown with light gray color. The nesting algorithm, used in this example, is slightly different than the nesting algorithm used in Figure 12(c), since care was taken to draw edge rectangles within the intersection of their origin and destination vertex rectangles.

Lemma 1 *Let $G = (V, E)$ be a TTSP digraph and $e_1 = (u_1, v_1)$, $e_2 = (u_2, v_2)$ be two edges of G. If there is no path in G from v_1 to u_2 and from v_2 to u_1 then Algorithm 1 draws edges e_1 and e_2 in non overlapping rectangles.*

Proof: Let $A = \{w \in V \mid w \overset{p}{\rightsquigarrow} u_1 \text{ and } w \overset{p}{\rightsquigarrow} u_2\}$. Notice that A always contains the source. Now, let $a \in A$ be the vertex which has the maximum longest path distance from the source among the vertices of A. We will show that vertex a is uniquely defined. If $u_1 = u_2$, then $a = u_1 = u_2$. If $u_1 \neq u_2$, suppose on the contrary, that there are two vertices $a_1, a_2 \in A$, $a_1 \neq a_2$ having the same longest path distance from the source. There is no path from a_1 to a_2 or from a_2 to a_1, since a_1 and a_2 have the same longest path distance from the source. Then the subgraph of G formed by vertices and edges of the paths $a_1 \overset{p}{\rightsquigarrow} u_1$, $a_1 \overset{p}{\rightsquigarrow} u_2$, $a_2 \overset{p}{\rightsquigarrow} u_1$ and $a_2 \overset{p}{\rightsquigarrow} u_2$ is homeomorphic to $K_{2,2}$. We conclude that G is not a TTSP digraph, which is a contradiction.

Similarly, let $B = \{w \in V \mid v_1 \overset{p}{\rightsquigarrow} w \text{ and } v_2 \overset{p}{\rightsquigarrow} w\}$. Notice that B always contains the sink. Now, let $b \in B$ be the vertex which has the maximum longest path distance to the sink among the vertices of B. Vertex b is unique. The proof is similar to the one for vertex a.

Any path starting at a, terminating at b and containing edge e_1 meets every path starting at a, terminating at b and containing edge e_2 only at the end vertices a and b. Therefore edges e_1 and e_2 belong to two different TTSP subgraphs among the $k \geq 2$ digraphs $G_1, \ldots, G_k$ (all subgraphs of G) that have source a and sink b. If $T_1, \ldots, T_k$ denote the decomposition trees of $G_1, \ldots, G_k$ and $\rho_1, \ldots, \rho_k$ their roots, then the decomposition tree T of G contains a P-node whose children are the nodes $\rho_1, \ldots, \rho_k$. The rectangle assigned to this P-node is partitioned among its children by Algorithm 1. Therefore edges e_1 and e_2 are drawn in non-overlapping rectangles. □

Theorem 1 *Every TTSP digraph admits a DAGmap drawing, which can be computed in* $\Theta(m)$ *time.*

Proof: We will show that the drawing produced by Algorithm 1 is compatible with the constraints of Definition 1.

Algorithm 1 assigns a rectangle to every vertex and edge of G. Every edge of G is assigned the rectangle of the associated Q-node of T. The source and the sink of G are assigned the initial drawing rectangle. Now, suppose that u is a vertex of G that is neither a source nor a sink. Then there is at least one edge incident to u and at least one edge incident from u. The TTSP recognition algorithm after performing a number of series and parallel reductions leaves exactly one edge (v, u) incident to u and exactly one edge (u, w) incident from u. Finally, the TTSP recognition algorithm removes node u by performing a series reduction, in which edges (v, u) and (u, w) are substituted by edge (v, w). The label of edge (v, w) is an S-node having as left subtree the label of edge (v, u) and as right subtree the label of edge (u, w). This S-node of T is associated with vertex u of G and the rectangle assigned to this S-node is also assigned to vertex u.

Clearly, constraints B1 and B2 are satisfied. Next we will show that constraint B3 is satisfied. Let $e = (u, v)$ be an edge of G. It is $area(R_e) \neq 0$ since Algorithm 1 always divides a rectangle into non-trivial rectangles. If u is the source of G then it is assigned the initial drawing rectangle and therefore

$R_e \subset R_u$. Similarly if v is the sink of G then $R_e \subset R_v$. If u is not the source of G then the TTSP recognition algorithm associates u with an S-node of T. Edge e is represented by a Q-node which is located in the right subtree of the tree rooted at this S-node. Similarly, if v is not the sink of G then edge e is represented by a Q-node which is located in the left subtree of the tree rooted at the S-node associated with vertex v. Algorithm 1 assigns the rectangle of an S-node to the left and right subtrees rooted at the S-node. These rectangles may be farther partitioned before Q-nodes of the subtrees are assigned rectangles. Therefore, in all cases we have: $R_e \subset R_u$ and $R_e \subset R_v \Rightarrow R_e \subset R_u \cap R_v$.

Constraint B4 is satisfied due to Lemma 1. We will show that constraint B5 is satisfied by induction on the composition rules of TTSP digraphs. If G is a base graph composed of two vertices connected by an edge then the rectangle of the sink is equal to the rectangle of the edge and constraint B5 is satisfied. Now, suppose that for two TTSP digraphs G_1 and G_2 constraint B5 is satisfied. Let G be the TTSP digraph that is produced by identifying the sink of G_1 with the source of G_2. The rectangle assigned to the source and the sink of G is equal to the rectangles assigned to the source and sink vertices of G_1 and G_2. Using the induction hypothesis, we conclude that the rectangle of every non-source vertex of G is equal to the union of rectangles of its incoming edges.
Now, let G denotes the TTSP digraph that is produced by parallel composition of G_1 and G_2. Rectangle R_s assigned to source s of G is partitioned into two rectangles R_{s_1} and R_{s_2}. Rectangle R_{s_1} is assigned to source s_1 of G_1 and rectangle R_{s_2} to source s_2 of G_2. Rectangle R_t of the sink t of G is partitioned into two rectangles R_{t_1} and R_{t_2} assigned to sink t_1 of G_1 and to sink t_2 of G_2 respectively. It hold that $R_{t_1} = R_{s_1}$ and $R_{t_2} = R_{s_2}$ because the source and sink rectangles of G_1 (resp. G_2) are equal. Then by the induction hypothesis, constraint B5 holds for every non-source and non-sink vertices of G. It remains to show that constraint B5 holds for sink t of G. The set of edges $\Gamma^-_G(t)$ incident to t of G is equal to the union of the sets of edges $\Gamma^-_{G_1}(t_1)$ and $\Gamma^-_{G_2}(t_2)$ incident to t_1 of G_1 and to t_2 of G_2 respectively. Also $R_t = R_{t_1} \cup R_{t_2}$ and by the induction hypothesis R_{t_1} (resp. R_{t_2}) is equal to the union of the rectangles of edges $\Gamma^-_{G_1}(t_1)$ (resp. $\Gamma^-_{G_2}(t_2)$). Therefore R_t is equal to the union of the rectangles of edges $\Gamma^-_G(t) = \Gamma^-_{G_1}(t_1) \cup \Gamma^-_{G_2}(t_2)$.
The proof that the constraint B6 is satisfied, is similar to the proof for constraint B5. Therefore, all constraints of Definition 1 are satisfied.

The $\Theta(m)$ worst case time holds since the TTSP recognition algorithm runs in $\Theta(m)$ time and the drawing algorithm performs one traversal of the decomposition tree which has $\Theta(m)$ nodes. □

3.2 One-Dimensional DAGmaps

We continue by restricting the ways in which the initial rectangle is partitioned; namely we consider only vertical (or only horizontal) partitions.

Definition 5 *A DAGmap is called one-dimensional if the rectangles representing the vertices and the edges of a DAG have their top and bottom (left and* *right) sides on the top and bottom (left and right) sides respectively of the initial drawing rectangle (i.e., the initial rectangle is sliced only along the vertical (horizontal) direction). See Figure 12(b) for an example.*

Since the height (resp. width) of all the rectangles is constant and equal to the height (resp. width) of the initial drawing rectangle, the problem is one-dimensional and the rectangles R_q can be represented by intervals I_q (Figure 14). Next we will define and study the problem of recognizing whether a DAG admits a one-dimensional DAGmap.

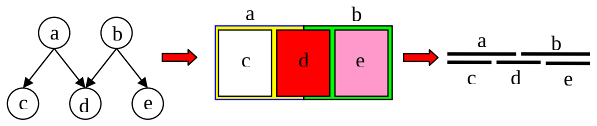


Figure 14: A one-dimensional DAGmap example.

Problem 1 (ONE-DIMENSIONAL DAGMAP)
INSTANCE: A DAG G.
QUESTION: Does G admit a one-dimensional DAGmap?

In this section we study a restricted version of the one-dimensional DAGmap problem. We consider the case that a DAG $G = (V, E)$ is layered with vertex partition $V = L_1 \cup \ldots \cup L_h$, $h > 1$, such that the sources of G are in L_h and the sinks of G are in L_1. Without loss of generality we assume that the layering is proper. To each vertex and edge of G we assign a rational number which is the length of its drawing interval. If $length(I)$ is the length of the initial interval I then each sink vertex $u \in L_1$ is assigned the number $\frac{length(I)}{|L_1|}$, denoted by $size(u)$. The edges incident to a sink vertex u are assigned sizes according to constraint A2 of Definition 2 (for each $e \in \Gamma^-(u)$, $size(e) = \frac{size(u)}{|\Gamma^-(u)|}$), the vertices in L_2 are assigned sizes using constraint B6 of Definition 1 (for each $u \in L_2$, $size(u) = \sum_{e \in \Gamma^+(u)} size(e)$), and so on. After calculating sizes for vertices and edges the following Lemma holds.

Lemma 2 *If the sources of a proper layered DAG $G = (L_1 \cup \ldots \cup L_h, E)$ are in layer L_h, the sinks are in layer L_1 and G has no isolated vertices then:*
a) $\sum_{v \in L_i} size(v) = length(I)$, $i \in \{1, \ldots, h\}$, and
b) $\sum_{e \in E_i} size(e) = length(I)$, $i \in \{2, \ldots, h\}$

Proof: Initially size $\frac{length(I)}{|L_1|}$ is assigned to each sink and the size is propagated toward the sources such that the size of a non-source vertex is equal to the sum of sizes of its incoming edges and the size of a non-sink vertex is equal to the sum of sizes of its outgoing edges. Since there are no sources in layers L_i, $i < h$, no sinks in layers L_i, $i > 1$ and no isolated vertices it is straightforward to see that a) and b) hold. □

To proceed with our analysis we need some definitions. A drawing of a layered graph G in the plane is a layered drawing if the vertices of every L_i, $1 \leq i \leq h$, are placed on a horizontal line $l_i = \{(x, i) \mid x \in \mathbb{R}\}$, and every edge $(u, v) \in E$, $u \in L_i$, $v \in L_j$, $1 \leq j < i \leq h$, is drawn as a line segment between the lines l_i and l_j. A layered drawing of G is called layered planar if no two edges cross except at common endpoints. A layered graph is layered planar if it has a layered planar drawing.

A layered drawing of G determines for every L_i, $1 \leq i \leq h$, a total order $\leq_i$ on the vertices of L_i given by the left to right order of the vertices on l_i. A layered embedding consists of a permutation of the vertices of L_i for every $i \in \{1, \ldots, h\}$ with respect to a layered drawing. A layered embedding with respect to a layered planar drawing is called layered planar.

Theorem 2 *Let $G = (V, E)$ be a proper layered DAG with vertex partition $V = L_1 \cup L_2 \cup \ldots \cup L_h$, where $h > 1$, such that the sources are in L_h and the sinks are in L_1. DAG G admits a one-dimensional DAGmap if and only if it is layered planar.*

Proof: Suppose that G admits a one-dimensional DAGmap. Then, by Proposition 1, there is a total ordering of the intervals of vertices in L_i, $i \in \{1, \ldots, h\}$. The ordering of the intervals defines an ordering on the vertices of L_i. Therefore the one-dimensional DAGmap defines a layered embedding of G. We will show that this embedding is layered planar. It suffices to show that between two consecutive layers L_i and L_{i-1}, $i \in \{2, \ldots, h\}$, there are no edge crossings. For this, suppose that two edges $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ cross. Then $u_1 <_i u_2$ and $v_1 >_{i-1} v_2$ or $u_1 >_i u_2$ and $v_1 <_{i-1} v_2$. Without loss of generality we assume that $u_1 <_i u_2$ and $v_1 >_{i-1} v_2$. Then $I_{u_1} < I_{u_2}$ and $I_{v_1} > I_{v_2}$. The contradiction comes from the fact that we cannot have $I_{e_1} \subset I_{u_1} \cap I_{v_1}$ and $I_{e_2} \subset I_{u_2} \cap I_{v_2}$, with $length(I_{e_1}) \neq 0$ and $length(I_{e_2}) \neq 0$ and $length(I_{e_1} \cap I_{e_2}) = 0$. We arrived at contradiction because we assumed that two edges in E_i cross. Therefore the embedding is layered planar.

Conversely, suppose that G is layered planar. Then G admits a planar layered embedding. This embedding defines a total order on the vertices of each layer L_i, $i \in \{1, \ldots, h\}$ as well as on the edges between two layers since for $(u_1, v_1), (u_2, v_2) \in E_i$, $i \in \{2, \ldots, h\}$, it holds either $u_1 \leq_i u_2$ and $v_1 \leq_{i-1} v_2$ or $u_1 \geq_i u_2$ and $v_1 \geq_{i-1} v_2$.

Now, suppose that we have calculated the size of vertex intervals and of edge intervals for constructing a one-dimensional DAGmap of G. In a drawing, we arrange the intervals $\{I_v \mid v \in L_i\}$, $i \in \{1, \ldots, h\}$ according to the ordering of vertices in L_i. For $i \in \{1, \ldots, h\}$ the intervals $\{I_v \mid v \in L_i\}$ cover the initial rectangle I and pairwise do not overlap. Therefore they constitute a partition of I. Similarly the ordering of edges in E_i, $i \in \{2, \ldots, h\}$ defines an ordering on the corresponding edge intervals $\{I_e \mid e \in E_i\}$ which form a partition of I. We will show that the size and orderings of vertices and edges of G define a one-dimensional DAGmap of G by showing that the constraints of Definition 1 are satisfied. Clearly constraint B1 is satisfied.

Constraint B2 is satisfied because $\sum_{v \in L_h} size(v) = length(I)$ (see Lemma 2). For constraint B3 we have that every edge $e = (u, v) \in E$ is drawn as an interval I_e of non-zero length. It remains to show that $I_e \subset I_u \cap I_v$. For this assume that $e \in E_i$, $i \in \{2, \ldots, h\}$ and consider how the intervals of vertices in L_{i-1} are related to intervals of edges in E_i. Let $v_1, v_2, \ldots, v_k$ be the vertices of L_{i-1} arranged in the order defined by the planar layered embedding of G. In the ordering of edges in E_i the incoming edges of v_1 come first. Therefore $I_{v_1} = \cup_{e' \in \Gamma^-(v_1)} I_{e'}$. Assuming that for $1 \leq j \leq l < k$ it holds that $I_{v_j} = \cup_{e' \in \Gamma^-(v_j)} I_{e'}$ we will show that $I_{v_{l+1}} = \cup_{e' \in \Gamma^-(v_{l+1})} I_{e'}$. The union $\cup_{e' \in \Gamma^-(v_{l+1})} I_{e'}$ is an interval since the incoming edges of vertex v_{l+1} are consecutive in the ordering of edges E_i. Additionally we have that intervals $I_{v_{l+1}}$ and $\cup_{e' \in \Gamma^-(v_{l+1})} I_{e'}$ start at the same point since $I_{v_1} \cup \ldots \cup I_{v_l} = (\cup_{e' \in \Gamma^-(v_1)} I_{e'}) \cup \ldots \cup (\cup_{e' \in \Gamma^-(v_l)} I_{e'})$ and they have the same length since $size(I_{v_{l+1}}) = \sum_{e' \in \Gamma^-(v_{l+1})} size(I_{e'}) = size(\cup_{e' \in \Gamma^-(v_{l+1})} I_{e'})$. Therefore for every $w \in L_{i-1}$ we have $I_w = \cup_{e' \in \Gamma^-(w)} I_{e'}$. Similarly we can show that for every $w \in L_i$ we have $I_w = \cup_{e' \in \Gamma^+(w)} I_{e'}$. Finally we have that $I_e \subset I_u$ and $I_e \subset I_v \Rightarrow I_e \subset I_u \cap I_v$.
From the above arguments it follows that constraints B5 and B6 are satisfied. Constraint B4 is satisfied when two edges belong to the same edge set E_i, $i \in \{2, \ldots, h\}$ since the set $\{I_e \mid e \in E_i\}$ is a partition of interval I. Now, suppose that $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ are two edges such that: a) $v_1 \in L_i$ and $u_2 \in L_j$ with $i > j$, b) there is no path from v_1 to u_2 and c) $length(I_{e_1} \cap I_{e_2}) \neq 0$. We will show that these assumptions lead to contradictions. We have: $I_{e_1} \subset I_{v_1}$ and $I_{e_2} \subset I_{u_2}$. Therefore $I_{e_1} \cap I_{e_2} \subset I_{v_1} \cap I_{u_2} \Rightarrow length(I_{v_1} \cap I_{u_2}) \neq 0$. This together with equation $I_{v_1} = \cup_{e' \in \Gamma^+(v_1)} I_{e'}$ imply that there is an outgoing edge $o_1 = (v_1, w_1)$ of vertex v_1 such that its interval I_{o_1} overlaps with interval I_{u_2}. Therefore $length(I_{o_1} \cap I_{u_2}) \neq 0 \Rightarrow length(I_{w_1} \cap I_{u_2}) \neq 0$. If $w_1 = u_2$ then there is a path from v_1 to u_2, a contradiction. If $w_1 \neq u_2$ then vertex w_1 and u_2 do not belong to the same layer since $length(I_{w_1} \cap I_{u_2}) \neq 0$. Therefore $i - 1 > j$. Continuing in this way we argue that there is an outgoing edge $o_2 = (w_1, w_2)$ of vertex w_1 such that its interval I_{o_2} overlaps with interval I_{u_2}, and so on. This procedure terminates after a finite number of steps and leads either to a path from v_1 to u_2 or to a vertex w_l that belongs to layer L_j such that $length(I_{w_l} \cap I_{u_2}) \neq 0$. The first conclusion contradicts with the hypothesis that there is no path from v_1 to u_2, while the second contradicts with the hypothesis that the set $\{I_v \mid v \in L_j\}$ is a partition of interval I. We arrive at contradictions because we assumed that edge intervals I_{e_1} and I_{e_2} overlap although there is no path from $dest(e_1)$ to $orig(e_2)$. Therefore constraint B4 is satisfied. □

From the above theorem it follows that the ONE-DIMENSIONAL DAGMAP problem is reduced to LAYER PLANARITY TEST. The later problem can be decided in linear time. When the LAYER PLANARITY TEST algorithm indicates that a layered graph G is layered planar then a planar embedding of G can be obtained in linear time using algorithm LAYER PLANAR EMBED [11].

Algorithm 2 ONE-DIMENSIONAL DAGMAP TEST
Input: a layered DAG $G = (L_1 \cup \ldots \cup L_h, E)$ that is proper and has its sources in layer L_h and its sinks in layer L_1.
Output: "true" if G admits a one-dimensional DAGmap, and "false" otherwise.

```
1. return LAYER-PLANARITY-TEST(G)                              □
```

Algorithm 3 ONE-DIMENSIONAL DAGMAP DRAW
Input: a planar layered DAG $G = (L_1 \cup \ldots \cup L_h, E)$ that is proper and has its sources in layer L_h and its sinks in layer L_1, and a rectangle R.
Output: a DAGmap drawing of G.

```
1. Find a planar embedding of $G$ using algorithm LAYER-PLANAR-EMBED.

2. Assign sizes to vertices and edges of $G$.

3. Using the orderings and sizes of vertices and edges draw $G$.    □
```

Theorem 3 *Every layered DAG $G = (L_1 \cup \ldots \cup L_h, E)$, that is proper and has its sources in layer L_h and its sinks in layer L_1, can be recognized for whether it admits a one-dimensional DAGmap or not in time $O(m)$. If G admits a one-dimensional DAGmap then a drawing of G can be produced in time $O(n)$.*

3.3 Minimization of Vertex Duplications in One-Dimensional DAGmaps

Motivated by Algorithms 2 and 3 we pose the question of whether an h-layer graph G_1, that is proper and has its sources in layer L_h and its sinks in layer L_1, can be transformed into a planar h-layer DAG G_2 by performing a minimum number of vertex duplications. We will show that the problem of minimizing the vertex duplications that are needed in order to convert G_1 into G_2 is NP-hard.

Problem 2 (DUPLICATIONS IN ONE-DIMENSIONAL DAGMAPS)
INSTANCE: A DAG G_1 and an integer K.
QUESTION: Can G_1 be transformed into a DAG G_2 that admits a one-dimensional DAGmap by duplicating at most K vertices?

When the input is restricted to two-layer DAGs where each first-layer vertex has in-degree two, the above problem is related to TWO-LAYER PLANARIZATION problem.

Definition 6 *A caterpillar is a connected graph that has a path b called the backbone such that all vertices of degree larger than one lie on b. The edges of a caterpillar that are not on the backbone are the legs of the caterpillar.*

Lemma 3 *[7] A two-layer graph $G = (L_1 \cup L_2, E)$ is two-layer planar if and only if it is a collection of disjoint caterpillars.*

Problem 3 (TWO-LAYER PLANARIZATION)
INSTANCE: A positive integer K and a two-layer graph $G = (L_1 \cup L_2, E)$.
QUESTION: Can G be made two-layer planar by deleting at most K edges?

Theorem 4 *[7] The TWO-LAYER PLANARIZATION problem is NP-complete and remains NP-complete when each vertex in L_1 has degree two.*

Theorem 5 *The DUPLICATIONS IN ONE-DIMENSIONAL DAGMAPS problem is NP-complete and remains NP-complete even when the input is restricted to simple two-layer DAGs where each first-layer vertex has in-degree two.*

Proof: The problem belongs to NP since given K vertex duplications we can check in linear time if the transformed DAG admits a one-dimensional DAGmap. We will show that the problem is NP-hard by reducing the TWO-LAYER PLANARIZATION problem to it. The reduction is trivial. Let G be a two-layer DAG, where each first-layer vertex has in-degree two. If $e = (u, v)$ is an edge of G then deletion of the edge corresponds to duplication of vertex v. Conversely, duplication of a vertex v corresponds to deletion of one of two edges that are incident to v.

Suppose that the TWO-LAYER PLANARIZATION problem has a solution. Then there are K edges $\{e_1, e_2, \ldots, e_K\}$ whose deletion leads to a graph G_2 that is two-layer planar. According to Lemma 3, the graph G_2 is a collection of disjoint caterpillars. Let $\{v_1, v_2, \ldots, v_K\}$ be the first layer vertices incident to edges $\{e_1, e_2, \ldots, e_K\}$. Now suppose that if instead of deleting the edges $\{e_1, e_2, \ldots, e_K\}$, we duplicate the vertices $\{v_1, v_2, \ldots, v_K\}$, and let $\{v'_1, v'_2, \ldots, v'_K\}$ be the replicas of vertices $\{v_1, v_2, \ldots, v_K\}$ which are attached to edges $\{e_1, e_2, \ldots, e_K\}$. The new graph, call it G_3, is a collection of disjoint caterpillars. This is because if we ignore the edges $\{e_1, e_2, \ldots, e_K\}$ and the incident vertices $\{v'_1, v'_2, \ldots, v'_K\}$, graph G_3 is equal to graph G_2 which is a collection of disjoint caterpillars. Then, since vertices $\{v'_1, v'_2, \ldots, v'_K\}$ have degree one the incident edges $\{e_1, e_2, \ldots, e_K\}$ can be considered as legs attached to the backbone of some caterpillar. This implies that G_3 is a caterpillar and from Lemma 3 it follows that G_3 is two-layer planar. Then, from Theorem 2, it follows that G_3 admits a one-dimensional DAGmap.

Conversely, suppose that the DUPLICATIONS IN ONE-DIMENSIONAL DAGMAPS problem has a solution and let $\{v_1, v_2, \ldots, v_K\}$ be a set of K vertices whose duplication leads to a DAG G_3 that admits a DAGmap drawing. According to Theorem 2, DAG G_3 is two-layer planar. DAG G_3 remains two-layer planar if we delete one of the two replicas of vertices $\{v_1, v_2, \ldots, v_K\}$ together with the incident edge. This corresponds to the deletion of K edges from the initial graph G. Therefore the TWO-LAYER PLANARIZATION problem has a solution.

We showed that the DUPLICATIONS IN ONE-DIMENSIONAL DAGMAPS problem has a solution if and only if the TWO-LAYER PLANARIZATION problem has a solution. Therefore the problem is NP-complete. □

4 The General Case

4.1 The Recognition Problem

Suppose that we have a layered DAG. Taking the vertices of a layer L_k isolated from the rest of the DAG, the problem is similar to a floorplan problem where the initial rectangle is dissected into $n_k = |L_k|$ soft rectangles, i.e., rectangles whose area is fixed but their dimensions may vary. The number of possible dissections (the solution space) is bounded below by $\Omega(n_k! 2^{3n}/n_k^4)$ and above by $O(n_k! 2^{5n}/n_k^{4.5})$ [14].

Considering two consecutive layers L_{k+1} and L_k of a DAG, the layouts of the two layers are constrained by the edges among the two layers, according to the drawing rules. The combined solution space may be empty or contain a number of solutions. We will show that deciding whether the solution space is empty or not is NP-complete. We call this decision problem DAGMAP and we define it as:

Problem 4 (DAGMAP)
INSTANCE: A DAG G.
QUESTION: Does G admit a DAGmap?

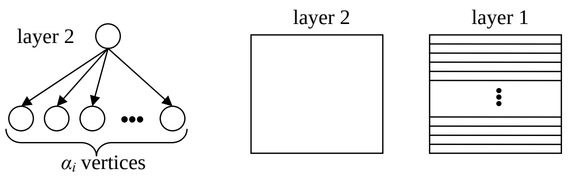
Our hardness result for DAGMAP is based on a transformation from the following decision problem.

Problem 5 (3-PARTITION)
INSTANCE: A multiset A of $3m$ *positive integers* $A = \{\alpha_1, \alpha_2, \ldots, \alpha_{3m}\}$ *where the* α_i*'s are bounded above by a polynomial in* m *and* $\frac{\Sigma}{4} < \alpha_i < \frac{\Sigma}{2}$*, where* $\Sigma = \frac{1}{m}(\alpha_1 + \alpha_2 + \ldots + \alpha_{3m})$.
QUESTION: Can A be partitioned into m *triples* $A_1, A_2, ..., A_m$ *such that each triple has the same sum? Specifically each triple must sum to* Σ.

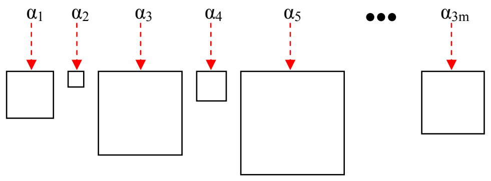
3-PARTITION is strongly NP-complete since it remains NP-complete even when representing the numbers in the input instance in unary [8]. The condition $\frac{\Sigma}{4} < \alpha_i < \frac{\Sigma}{2}$ forces every set of α_i's summing to Σ to have size exactly 3.

Theorem 6 *The DAGMAP problem is NP-complete even if we restrict it to forests of two-layer DAGs.*

Proof: Given a dissection of the initial drawing rectangle into $|L_k|$ rectangles for each layer L_k, $k \in \{1, \ldots, h\}$ of a DAG G, we can check in polynomial time if these dissections correspond to a DAGmap drawing of G. Moreover, in the VLSI layout literature, there are a few techniques for succinctly encoding the partition of a rectangle into soft rectangles [14]. Therefore the DAGMAP problem belongs to NP. Next we will show that DAGMAP is NP-hard. Given an instance $A = \{\alpha_1, \alpha_2, \ldots, \alpha_{3m}\}$ of 3-PARTITION we will construct a forest of two-layer DAGs that admits a DAGmap drawing if and only if the 3-PARTITION problem has a solution.



(a) DAG G_{α_i} and one possible drawing of it



(b) An example of the mapping from integers α_i to rectangles of area α_i

Figure 15: To each $\alpha_i \in A$ we correspond a DAG G_{α_i} with $\alpha_i + 1$ vertices and to G_{α_i} we correspond a drawing rectangle R_{α_i} of area α_i.

Without loss of generality we assume that the first-layer vertices of a DAG are drawn in unit area rectangles. For each $\alpha_i \in A$ we consider a two-layer DAG G_{α_i} which has one vertex in layer two and α_i vertices in layer one. The second-layer vertex is drawn as a rectangle, R_{α_i}, with area α_i, but without any constraint on the aspect ratio of its sides. The first-layer vertices are drawn as unit area rectangles by slicing with parallel horizontal line segments the rectangle of the second-layer vertex (Figure 15). The total area occupied by rectangles R_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ is $\alpha_1 + \alpha_2 + \ldots + \alpha_{3m} = m\Sigma$.

We want to draw rectangles R_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ on m equal and pairwise isolated rectangular regions inside the initial rectangle R. To do this we consider an additional DAG whose drawing leaves m empty rectangular regions (gaps), each of area Σ. We call this additional DAG the enforcer, since its drawing enforces the previously defined $3m$ rectangles to be drawn inside the m gaps. The shape of the enforcer is unique up to left, right, up or down orientation inside the initial rectangle R (Figure 16). In the following, without loss of generality, we assume that the shape of the enforcer is similar to the one of Figure 16(a).

Enforcer: The DAG used as enforcer (Figure 17) has $2m + 2$ vertices on the second layer, the vertices β and $1, 2, \ldots, 2m + 1$. Every odd numbered vertex has Σ exclusive out-neighbors, while for the even numbered vertices there are no exclusive out-neighbors. Therefore vertices $1, 3, \ldots, 2m+1$ have Σ more area than vertices $2, 4, \ldots, 2m$. The role of vertex β and of first-layer vertices with in-degree greater than one is to align rectangles $R_1, R_2, \ldots, R_{2m+1}$ and to force

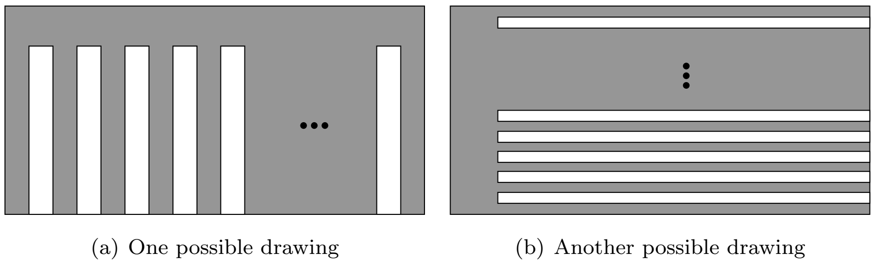

(a) One possible drawing (b) Another possible drawing

Figure 16: The drawing of the enforcer leaves m gaps, each of area Σ.

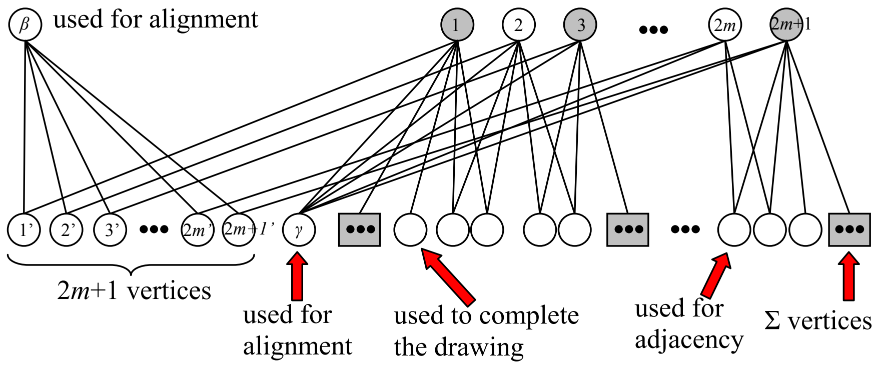


Figure 17: The DAG used as enforcer in the proof.

rectangles R_i and R_{i+1}, $i \in \{1, 2, \ldots, 2m\}$ to be adjacent.

Each of the first-layer vertices $1', 2', \ldots, (2m+1)'$ has two in-neighbors. One is vertex β and the other is the corresponding numbered second-layer vertex. In the drawing rectangle R_β is adjacent to all rectangles $R_1, R_2, \ldots, R_{2m+1}$. Therefore rectangles $R_1, R_2, \ldots, R_{2m+1}$ should be drawn around the sides of rectangle R_β. Rectangle R_γ is drawn on top of rectangles $R_1, R_2, \ldots, R_{2m+1}$ and has zero area intersection with rectangle R_β. Therefore it forces rectangles $R_1, R_2, \ldots, R_{2m+1}$ to be drawn consecutively along one side of rectangle R_β. Additionally, since the area of rectangle R_γ is equally distributed among rectangles $R_1, R_2, \ldots, R_{2m+1}$, it forces these rectangles to have the same width. The second-layer vertices i and $i+1$, $i \in \{1, 2, \ldots, 2m\}$ have a common out-neighbor which constrains their rectangles to be adjacent. The second common out-neighbor of vertices i and $i+1$, is used for completing the drawing. Also for completing the drawing, vertex 1 has an exclusive out-neighbor and similarly vertex $2m+1$. To sum up, the first layer has $(m+1)\Sigma+6m+4$ vertices and therefore the total area occupied by the drawing of the enforcer is $(m+1)\Sigma+6m+4$

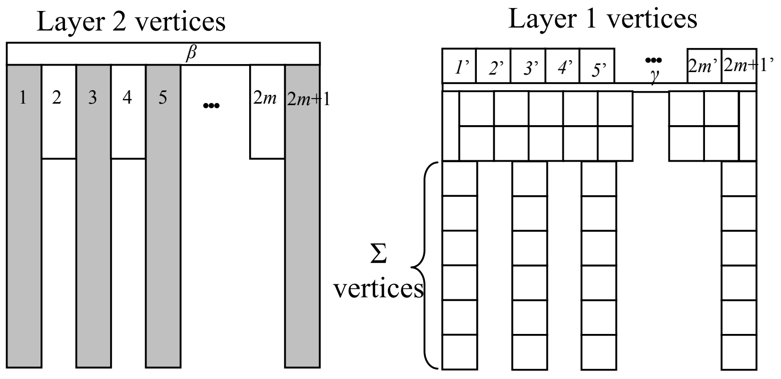
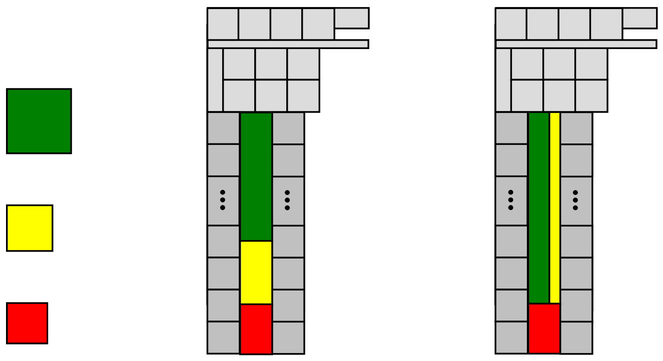


Figure 18: One possible drawing of the enforcer.



(a) Three rectangles (b) One way to fill the gap (c) Another way to fill the gap

Figure 19: An example of how three rectangles, of total area Σ, fill a gap.

(Figure 18). The total area occupied by the drawings of the enforcer and of DAGs G_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ is: $((m+1)\Sigma+6m+4)+m\Sigma = (2m+1)(\Sigma+3)+1$.

Now suppose that the 3-PARTITION instance has a solution. Then the elements of A can be partitioned into m triples $A_1, A_2, \ldots, A_m$ such that each triple has sum Σ. Then the rectangles that correspond to the elements of a triple fit exactly into a gap of area Σ (Figure 19). Therefore the DAGMAP problem has a solution.

Conversely, if the DAGMAP problem has a solution then the $3m$ rectangles fill all the gaps. From the condition $\frac{\Sigma}{4} < \alpha_i < \frac{\Sigma}{2}$ on the 3-PARTITION numbers and therefore on the rectangle areas, a gap is filled by exactly three rectangles. Therefore the $3m$ rectangles are partitioned into m triples each of total area Σ. This partition is also a solution to the 3-PARTITION instance. The reduction from the 3-PARTITION to DAGMAP uses a polynomial number of resources since the numbers involved in 3-PARTITION are positive integers bounded by a polynomial in m. Note that one can achieve the geometric construction using simple geometric operations. □

Theorem 7 *The DAGMAP recognition problem remains NP-complete even if the input is a two-layer DAG.*

Proof: The reduction is similar to the one in Theorem 6. The differences are:

1. The enforcer and the $3m$ DAGs of Theorem 6 now form a single two-layer DAG instead of a forest of DAGs.
2. The α_i vertices have $\alpha_i - 1$ exclusive out-neighbors instead of α_i.
3. Each of vertices $1, 3, \ldots, 2m+1$ has $\Sigma - 1$ exclusive out-neighbors instead of Σ.

Vertices $1, 3, \ldots, 2m+1$ have Σ more area than vertices $2, 4, \ldots, 2m$. In a drawing, rectangle R_δ is adjacent to each one of rectangles $R_1, R_3, \ldots, R_{2m+1}$, since for $i \in \{1, 3, \ldots, 2m+1\}$, vertices δ and i have one common out-neighbor (Figure 20). Their second common out-neighbor exists for completing the drawing. Rectangles R_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ have area equal to α_i and are adjacent to rectangle R_δ, since each vertex α_i and vertex δ have a common out-neighbor. Their second common out-neighbor is used for completing the drawing. Similarly, vertex δ has another $(2m+1)(\Sigma-2)+1$ exclusive out-neighbors. Therefore the area of rectangle R_δ is $(2m+1)(\Sigma-2)+1+3m+(m+1) = (2m+1)\Sigma$ and its horizontal and vertical sides have lengths $2m+1$ and Σ respectively.

The adjacency relations between the second-layer vertices $1, 2, 3, \ldots, 2m+1$, β and δ leads to a shape having m gaps, each one of area Σ (Figure 20). The drawing position of rectangles R_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ is not fixed. The only constraint is that they should be adjacent to rectangle R_δ. However, in order to have a rectangular drawing, triples of rectangles R_{α_i} should fill the gaps. Since rectangles R_{α_i} are adjacent to rectangle R_δ the only way that each of them is drawn inside a gap is with vertical side of length Σ and horizontal side of length $\frac{\alpha_i}{\Sigma}$. The first-layer vertices that are adjacent to vertex δ and to vertex α_i have a drawing with vertical side of length 2Σ and horizontal side of length $\frac{1}{2\Sigma}$.

If the 3-PARTITION problem has a solution then triples of rectangles R_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ fit exactly into the gaps of the drawing and therefore the DAG admits a DAGmap. Conversely, if the DAG admits a DAGmap then rectangles R_{α_i}, $i \in \{1, 2, \ldots, 3m\}$ fill all the gaps of the drawing. And since exactly three rectangles can fill a gap, the solution of the DAGMAP instance provides a partitioning of integers α_i into m triples, each of size Σ. □

4.2 Minimization of Vertex Duplication

Problem 6 (DUPLICATIONS IN DAGMAPS)
INSTANCE: A DAG G_1 and an integer K.
QUESTION: Can G_1 be transformed into a DAG G_2 that admits a DAGmap by duplicating at most K vertices.

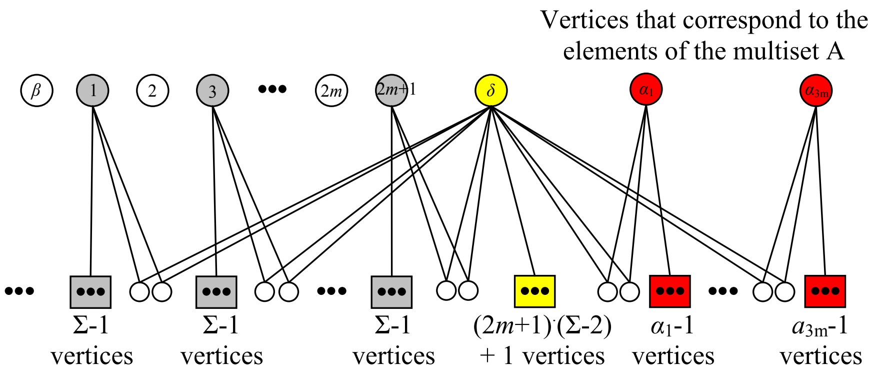
Comment: The problem DUPLICATIONS IN DAGMAPS is NP-hard since its restriction for $K = 0$ is the problem DAGMAP, which is NP-complete.

5 Discussion

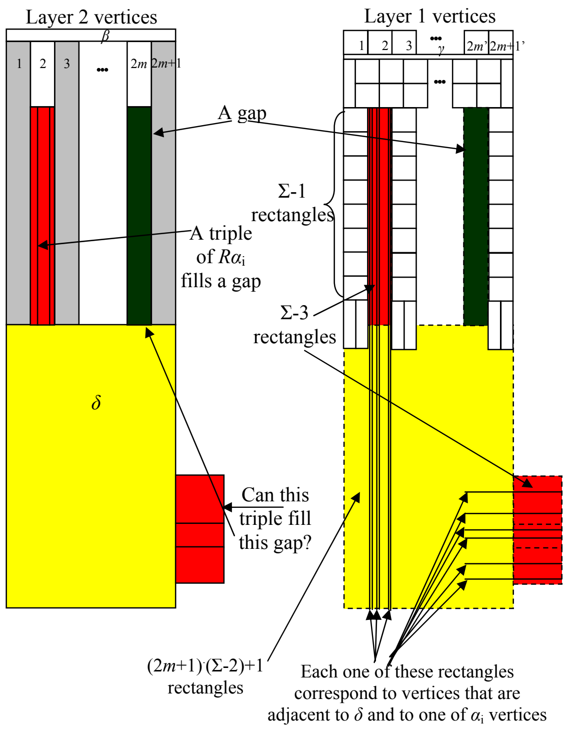
In this paper we introduced the problem of drawing a DAG using space filling techniques. We defined the recognition and vertex duplication minimization problems and we showed that in the general case they are NP-complete and NP-hard respectively. We also considered two special cases by restricting the form of the DAG and of the DAGmap respectively. We are currently investigating drawing heuristics based on relaxations of the drawing constraints and/or restrictions on the form of DAGs. The results of this research will be published in a subsequent paper concerning the application of these techniques to hierarchically organized ontologies.

Acknowledgments

The authors wish to thank Dimitris Andreou for providing us his graph library and the anonymous referees for several useful suggestions that improved the quality of the paper.


Vertices that correspond to the elements of the multiset A
β
1
2
3
2m
2m+1
δ
α1
α3m
Σ-1 vertices
Σ-1 vertices
Σ-1 vertices
(2m+1)·(Σ-2) + 1 vertices
α1-1 vertices
a3m-1 vertices

(a) Only the new connections are shown


Layer 2 vertices
β
1
2
3
2m
2m+1
Layer 1 vertices
γ
1
2
3
2m'
2m+1'
A gap
Σ-1 rectangles
A triple of Rαi fills a gap
Σ-3 rectangles
δ
Can this triple fill this gap?
(2m+1)·(Σ-2)+1 rectangles
Each one of these rectangles correspond to vertices that are adjacent to δ and to one of αi vertices

(b) One possible drawing

Figure 20: The enforcer of the reduction in Theorem 7.

References

[1] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene Ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature Genetics*, 25(1):25–29, 2000.

[2] E. H. Baehrecke, N. Dang, K. Babaria, and B. Shneiderman. Visualization and analysis of microarray and gene ontology data with treemaps. *BMC Bioinformatics*, 5:84, 2004.

[3] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Partitioning a square into rectangles: NP-completeness and approximation algorithms. *Algorithmica*, 34(3):217–239, 2002.

[4] B. B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Transactions on Graphics*, 21(4):833–854, 2002.

[5] M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps. In W. de Leeuw and R. van Liere, editors, *Data Visualization 2000, Proceedings of the Second Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 33–42, 2000.

[6] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, Upper Saddle River, NJ, 1998.

[7] P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoretical Computer Science*, 131(2):361–374, 1994.

[8] M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[9] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006.

[10] B. Johnson and B. Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the IEEE Visualization 91*, pages 284–291, 1991.

[11] M. Jünger and S. Leipert. Level planar embedding in linear time. *Journal of Graph Algorithms and Applications*, 6(1):67–113, 2002.

[12] S. Liu, N. Cao, and H. Lv. Interactive visual analysis of the NSF funding information. In *IEEE Pacific Visualisation Symposium 2008*, pages 183–190, 2008.

[13] H. Lü and J. Fogarty. Cascaded treemaps: examining the visibility and stability of structure in treemaps. In *ACM Proceedings of graphics interface 2008*, pages 259–266, 2008.

[14] Z. C. Shen and C. C. N. Chu. Bounds on the number of slicing, mosaic, and general floorplans. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(10):1354–1361, 2003.

[15] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, 1992.

[16] A. Symeonidis, I. G. Tollis, and M. Reczko. Visualization of functional aspects of microRNA regulatory networks using the gene ontology. In *ISBMDA*, volume 4345 of *LNCS*, pages 13–24. Springer, 2006.

[17] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982.

[18] J. J. van Wijk and H. van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *INFOVIS '99: Proceedings of the 1999 IEEE Symposium on Information Visualization*, pages 73–78, 1999.