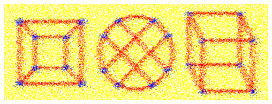

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 23, no. 3, pp. 553–578 (2019)
DOI: 10.7155/jgaa.00502

A New Framework for Hierarchical Drawings

Giacomo Ortali[1] *Ioannis G. Tollis*[2]

[1]University of Perugia
[2]Computer Science Department, University of Crete, Heraklion, Crete, Greece
and Tom Sawyer Software, Inc. Berkeley, CA 94707 U.S.A.

Abstract

We present a new approach to visualize directed graphs and their hierarchies that departs from the classical four-phase framework of Sugiyama and computes readable hierarchical visualizations that focus on the reachability information of a directed acyclic graph. Additionally, our approach has the feature that a few transitive edges are not drawn in the drawing, thus reducing the visual complexity of the resulting drawing. Furthermore, the problems involved in our framework require only polynomial time. The channel decomposition is a partition of the vertex set of the graph into channels, where a channel is a relaxed path. Our framework offers a suite of solutions depending upon the requirements, and it consists of only two steps: (a) the cycle removal step (if the directed graph contains cycles) and (b) the channel decomposition and hierarchical drawing step. Our framework does not introduce any dummy vertices and it keeps the vertices of a path/channel *vertically aligned*. The time complexity of the main drawing algorithms of our framework is $O(km)$, where k is the number of paths/channels, typically much smaller than n (the number of vertices).

Submitted:	Reviewed:	Revised:	Reviewed:
November 2018	February 2019	June 2019	July 2019
Revised:	Accepted:	Final:	Published:
July 2019	July 2019	July 2019	September 2019
Article type:		Communicated by:	
Regular paper		T. Biedl and A. Kerren	

This research was performed in part while the first author was visiting the University of Crete.
E-mail addresses: giacomo.ortali@gmail.com (Giacomo Ortali) tollis@csd.uoc.gr (Ioannis G. Tollis)

1 Introduction

Many applications in several areas of science and business require the visualization of directed (often acyclic) graphs. This is the case because such graphs often represent hierarchical relationships between objects in a structure (the graph). In their seminal paper of 1981, Sugiyama, Tagawa, and Toda [29] proposed a four-phase framework for producing hierarchical drawings of directed graphs. This framework is known in the literature as the "Sugiyama" framework, or algorithm. Most problems involved in the optimization of various phases of the Sugiyama framework are NP-hard. In this paper we present a new approach to visualize directed graphs and their hierarchies that departs from the classical four-phase framework of Sugiyama and computes readable hierarchical visualizations that contain the complete reachability information of a graph. Our framework reduces the visual complexity of the resulting drawing by (a) drawing the vertices of the graph in some vertical lines, and (b) by removing the transitive edges within each path and showing only the rest of the edges in the output drawing. Furthermore, the problems involved in our framework require polynomial time.

Figure 1 shows an example of two different hierarchical drawings: Part (a) shows the drawing of a directed graph G computed by Tom Sawyer Perspectives [1] (a tool of Tom Sawyer Software) that follows the Sugiyama framework; part (b) shows a hierarchical drawing computed by our framework taking G as input. Notice that in (b) the transitive edges within each vertical path are not shown. Additionally notice that there are two edges that cross twice. This is fixed in the next figures at the expense a few extra bends. The purpose of our figures is not to make a direct comparison between the drawings based on the Sugiyama Framework and the drawings based on our framework.

The Sugiyama framework for producing hierarchical drawings of directed graphs consists of four main phases [29]: (a) Cycle Removal, (b) Layer Assignment, (c) Crossing Reduction, and (d) Horizontal Coordinate Assignment. The reader can find the details of each phase and several proposed algorithms to solve various of their problems and subproblems in Chapter 9 of Di Battista et al.'s Graph Drawing book [8]. Other books have also devoted significant portions of their Hierarchical Drawing Algorithms chapters to the description of this framework [20, 24].

The Sugiyama framework has been extensively used in practice, as manifested by the fact that various systems have chosen it to implement hierarchical drawing techniques. Several systems such as *AGD* [26], *da Vinci* [12], *GraphViz* [15], *Graphlet* [16], *dot* [14], *OGDF* [7], and others implement this framework in order to hierarchically draw directed graphs. Even commercial software such as the Tom Sawyer Software TS Perspectives [1] and yWorks [2] essentially use this framework in order to offer automatic hierarchical visualizations of directed graphs. More recent information regarding the Sugiyama framework and newer details about various algorithms that solve its problems and subproblems can be found in [24].

Even though this framework is very popular, it has several limitations: As

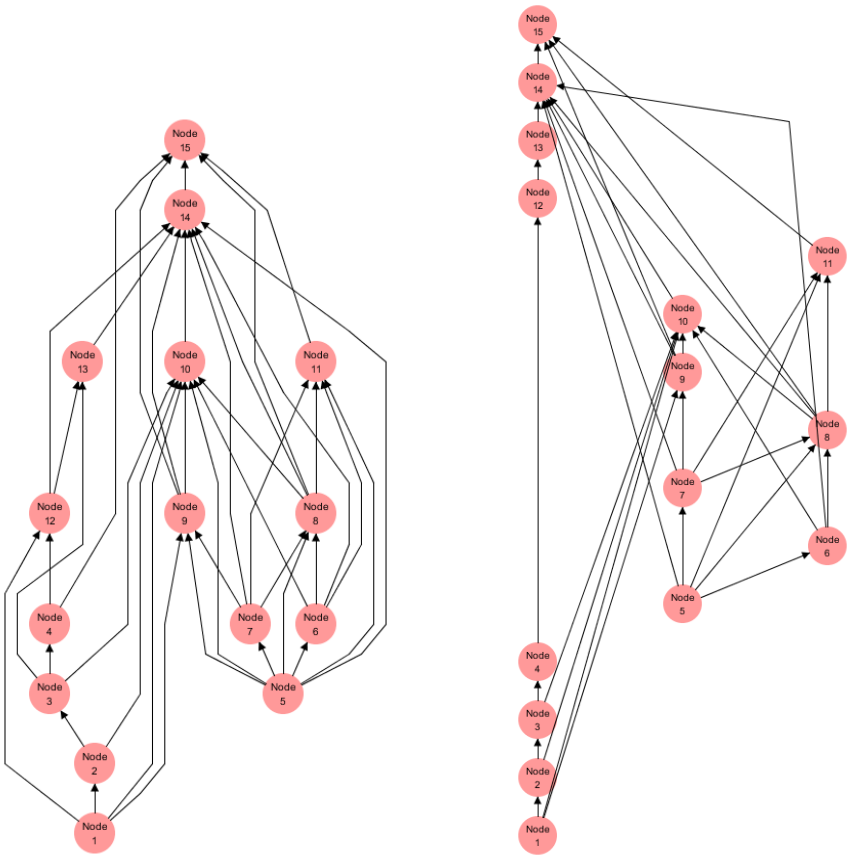


Figure 1: Two different hierarchical drawings: Part (a) shows the drawing of a directed graph G computed by Tom Sawyer Perspectives (a tool of Tom Sawyer Software) that follows the Sugiyama framework; part (b) shows a hierarchical drawing computed by our framework taking G as input.

discussed above, most problems and subproblems that are used to optimize the results of each phase have turned out to be NP-hard. Several of the heuristics employed to solve these problems give results that are not bounded by any approximation. Additionally, the required manipulations of the graph often increase substantially the complexity of the graph itself. For example, let $G = (V, E)$ be a directed graph with n vertices and m edges. The number of dummy vertices produced in phase (b) can be as high as $O(nm)$. The overall time complexity of this framework (depending upon implementation) can be as high as $O((nm)^2)$, or even higher if one chooses algorithms that require exponential time. Finally, the main limitation of this framework is the fact that the heuristic solutions and decisions that are made during previous phases (e.g., crossing reduction) will influence severely the results obtained in later phases. Nevertheless, previous decisions cannot be changed in order to obtain better results.

A *path* and a *channel* are both ordered sets of vertices. In a path every vertex is connected by a direct edge to its successor, while in a channel any vertex is connected to its successor by a directed path. The concept of channel can be considered as a generalization (or a relaxation) of the concept of path. In the literature the channels are also called *chains* [18].

Here we propose a new framework that departs from the typical Sugiyama framework and its four phases. Our framework is based on the idea of partitioning the vertices of a graph G into paths or channels; we call this a *path (or channel) decomposition* of G. We say that two graphs have the same *reachability properties* if their transitive closures are equivalent. After we partition the vertices of G into paths/channels, we compute a new graph Q which is closely related to G and has the same reachability properties as G. The new graph consists of the vertices of G, *path/channel edges* that connect vertices that are in the same path/channel, and *cross edges* that connect vertices that belong to different paths/channels; the transitive edges within a path or channel are omitted. Our framework draws either (a) graph G without the transitive "path/channel edges" or (b) a condensed form of the transitive closure of G. Additionally, since the vertices in each path/channel will be vertically aligned, an important aspect of the new framework is that (some or all of) the paths can be prescribed by the application or they can be user defined, at will. Our idea is to compute a hierarchical drawing of Q and, since Q has the same reachability properties as G, this drawing contains most edges of G and gives us all the reachability information of G. The "missing" incident (transitive) edges of a vertex can be drawn interactively on demand.

Our framework offers a suite of solutions depending upon the requirements of the user. For example, notice that edge $(8, 14)$ and edge $(11, 16)$ are crossing two times in Figure 1. This effect is unnecessary and it can be disturbing. Figure 2 shows two different versions of the drawing shown in Figure 1: Part (a) shows a drawing where a few bends are introduced to make the drawing more readable and remove the unnecessary crossings by (1) incrementing the angles between some edges incident to a same node and (2) bundling edges

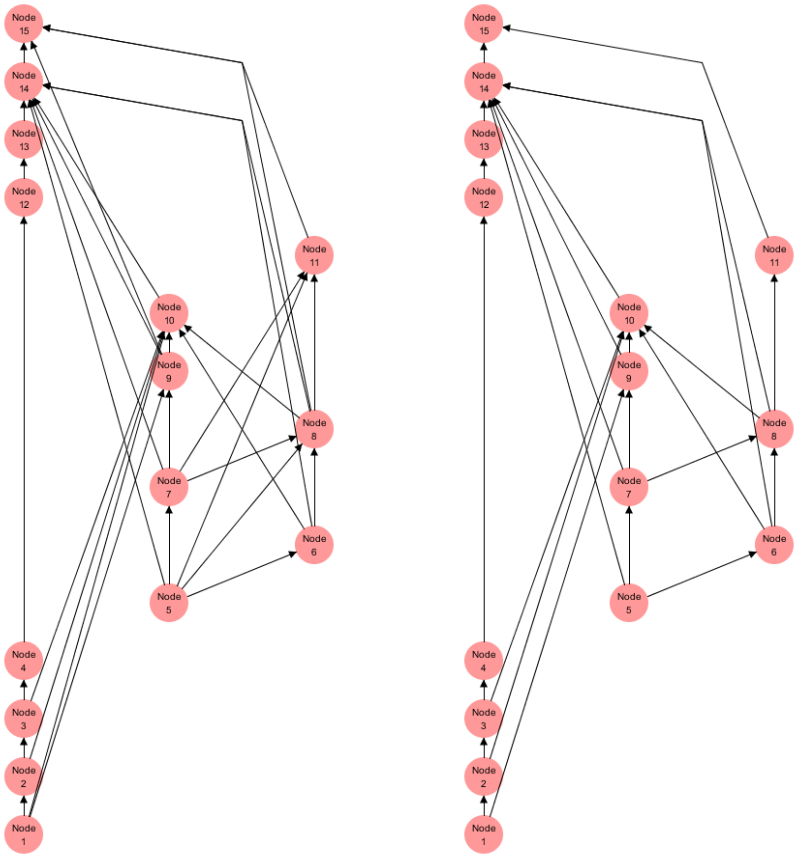


Figure 2: Two variants of the drawing shown in Figure 1.

that are entering into the same node, which reduces the number of crossings as well (edges that exit from the same node could be easily bundled as well); part (b) shows a reduced version of the drawing in (a), where a selected set of transitive edges is removed in order to make the drawing less cluttered. These and other alternative solutions make our framework very flexible and they will be presented in detail in the future.

Notice that in Figures 1 and 2 nodes $1, 2, 3$, and 4 could be easily shifted up in order to make the drawing more compact. This solution is not explored here, since it is not necessary in order to prove the bounds obtained by our framework. Clearly, a post-processing step based on simple compaction strategies will reduce the height of the drawings significantly. This solution can be exploited in various applications and it will be explored in future experimental works.

Our framework consists of only two steps: (a) the cycle removal step (if the graph contains cycles) and (b) the path/channel decomposition and hierarchical drawing step. Our framework does not introduce any dummy vertices, keeps the vertices of a path/channel *vertically aligned* and it offers answers to reachability queries between vertices by traversing at most one cross edge. Let k be the number of paths/channels and m' be the number of cross edges in the graph Q. We show that $m' = O(kn)$ (notice that Q could have fewer cross edges than G). The number of bends we introduce is at most $O(m')$ and the required area is at most $O(kn)$. The total time complexity of the algorithms of our framework is $O(km)$ plus the time required to compute the path/channel decomposition of G, which depends upon the type of decomposition required.

Our paper is organized as follows: The next section presents necessary preliminaries including a brief description of the phases of the Sugiyama framework, the time complexity of the phases, and a description of "bad" choices. In Section 3 we present the concept of path decomposition of a Directed Acyclic Graph (or simply DAG) and of path graph and we present a new algorithm for hierarchical drawing which is based on any (computed) path decomposition of a DAG. Section 4 presents the concepts of channel decomposition of a DAG and of channel graph (where channels are not just paths) and a new algorithm for hierarchical drawing which is based on any (computed) channel decomposition of a DAG. Section 5 presents a variation of the previous algorithms where some paths/channels are predefined by a user, and it discusses briefly an approach on the global minimization of crossings. In Section 6 we present the properties of the drawings obtained by our framework, we offer theoretical comparisons with the drawings obtained by traditional techniques, and present our conclusions and interesting open problems.

2 Sugiyama Framework

Given a directed graph $G = (V, E)$ with n vertices and m edges, a *hierarchical drawing* of G requires that all edges are drawn in the same direction upward (downward, rightward, or leftward) monotonically. If G contains cycles this is clearly not possible, since in any hierarchical drawing of the graph some edges have to be oriented backwards. The Sugiyama framework contains the Cycle Removal Phase in which a (small) subset of edges is selected and the direction of these edges is reversed, so that no cycles remain in G. Since it is important to maintain the character of the input graph, the number of the selected edges has to be minimum. This is a well known NP-hard problem, called the *Feedback Arc Set* problem. A well known approximation algorithm, called *Greedy-Cycle-Removal*, runs in linear time and produces sets that contain at most $m/2 - n/6$ edges. If the graph is sparse, the result is further reduced to $m/3$ edges [8].

Since the input graph G may contain cycles our framework also needs to deal with them. One approach is to use a cycle removal algorithm (similar to Sugiyama's first step) but instead of reversing the edges, we could remove them, since reversing them could lead to an altered transitivity of the original graph.

A reversed edge will be a transitive edge in the new graph and hence it may significantly affect the reachability of the graph and the resulting drawing. By the way, this is another disadvantage of such techniques: Since the choice of the edges to be reversed is done without any knowledge of the topology of the graph, it might be the case that nodes that should be placed close to each other (because they belong to a cycle) are finally placed very far from each other because the reversed edge is typically a long transitive edge. Since the removal and/or reversal of such edges will create a graph that will have a "different character" than the original graph we prefer to use another approach that will work well if the input graphs do not contain long cycles. It is easy to (a) find the *Strongly Connected Components (SCC)* of the graph in linear time, (b) cluster and collapse each SCC into a supernode, and then the resulting graph G' will be acyclic. This approach has been used in previous papers for various applications, see [13, 23]. Even if both techniques are acceptable, we believe that the second one will be able to better preserve the character of the original input graph (and its cycles). On the other hand, this technique may not be very useful if most vertices of a graph are included in a very long cycle. From now on, we assume that the given graph is acyclic after using either of the techniques described above.

In the Layer Assignment Phase of the Sugiyama framework the vertices are assigned to a layer and the layering is made *proper*, see [8, 24, 29]. In other words, long edges (typically transitive edges) that span several layers are broken down into multiple edges by introducing dummy vertices, so that every edge that starts at a layer terminates at the very next layer. Clearly, in a graph that has a longest path of length $O(n)$ and $O(m)$ transitive edges, the number of dummy vertices can be as high as $O(nm)$. This fact impacts the running time (and space) of all the subsequent phases, with heaviest impact on the next phase, the Crossing Reduction Phase.

The Crossing Reduction Phase is perhaps the most difficult and most time-consuming phase. It deals with various difficult problems that have attracted a lot of attention both by mathematicians and computer scientists. It is outside the scope of this paper to describe the various techniques for crossing reduction, however, the reader may see [8, 24] for further details. The most popular technique for crossing reduction is the *Layer-by-Layer Sweep* [8, 24]. This technique solves multiple instances of the well-known *Two-Layer-Crossing Problem* by considering the layers in pairs going up (or down). Of course, a solution for a specific two layer crossing problem "fixes" the relative order of the vertices (real and dummy) for the next two-layer crossing problem, and so on. Therefore, "bad" choices will propagate. Please note that the Two-Layer Crossing Minimization Problem is NP-complete [10]. The heuristics employed here tend to reduce crossings by various techniques, but notice that the number of crossings may be as high as $O(LM^2)$, where L is the number of layers and M is the average number of edges between the vertices of two adjacent layers. In [11], the authors present a heuristic that runs in $O(n+m\log(m))$ time and $O(n+m)$ space.

Finally, in the last phase the exact x-coordinates of the vertices can be

computed by quadratic-programming techniques [8, 24], which require considerable computational resources. Faster approaches are described in [4, 19]. The dummy vertices are replaced by bends. This implies that the number of bends can be as high as the number of dummy vertices. Modern techniques can draw edges with at most 2 bends per edge, avoiding the use of additional dummy vertices [5].

3 Path Based Drawings

Let $G = (V, E)$ be a DAG. In this paper we define a *path decomposition* of G as a set of vertex-disjoint paths $S_p = \{P_1, ..., P_k\}$ such that $V(P_1), ..., V(P_k)$ is a partition of $V(G)$. A path $P_h \in S_p$ is called a *decomposition path*. The vertices in a decomposition path are clearly ordered in the path, and we denote by v_i^j the fact that v is the jth vertex of path P_i. The *path decomposition graph*, or simply path graph, of G associated with path decomposition S_p is a graph $H = (V, A)$ such that $e = (u, v) \in A$ if and only if $e \in E$ and (a) u, v are consecutive in a path of S_p (called *path edges*) or (b) u and v belong to different paths (called *cross edges*). In other words, an edge of H is a *path edge* if it connects two consecutive vertices of the same decomposition path, else it is a *cross edge*. Notice that the edges belonging to G but not to H are transitive edges between vertices of the same path of G.

A *path based hierarchical drawing*, or simply path based drawing, Γ based on G given S_p is a hierarchical drawing of H such that two vertices are drawn on the same vertical line (i.e., same x-coordinate) if and only if they belong to the same decomposition path. In this section we propose an algorithm that computes path based drawings assigning to each vertex the x-coordinate of the path it belongs to and for y-coordinate we use its rank in a topological sorting. We use this y-coordinate assignment in order to simplify our mathematical proofs. As discussed in the Introduction, in practice a compaction strategy can be used in order to reduce the height of the drawing. We prove that this assignment lets us obtain good results in terms of both area and number of bends.

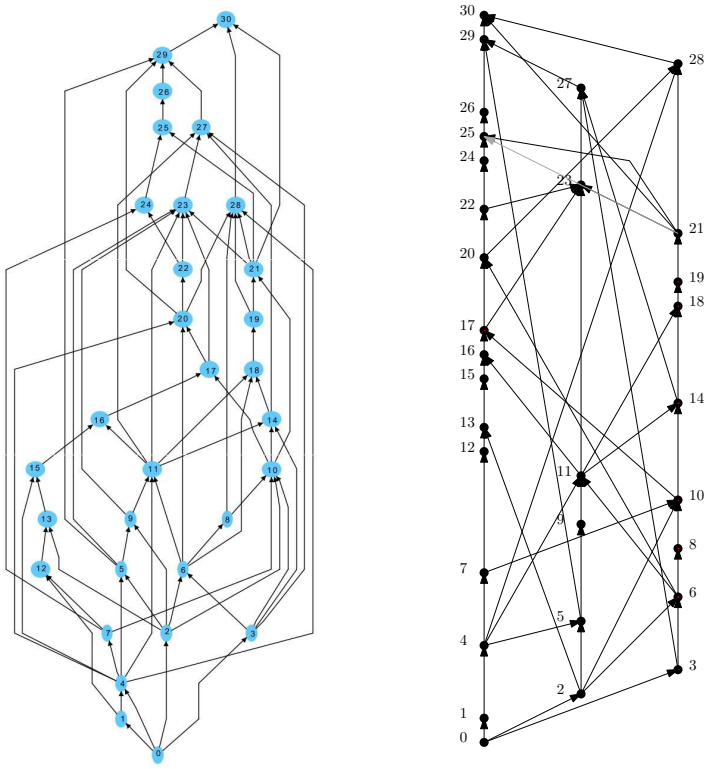
Next we present Algorithm PB-Draw that computes a path based drawing Γ of G such that every edge of G bends at most once. We assume here that the path decomposition is given, and discuss options for choosing it later. We denote by $X(P_h)$ the x-coordinate of path P_h and by $X(v), Y(v)$ the x-coordinate and the y-coordinate of any vertex v. Let P_v be the path of S_p containing v. By definition of path based drawing we have that $X(v) = X(P_v)$. Let T be a topological sorting of G and let $T(v)$ be the position of v in this sorting. PB-Draw associates to every path, and consequently to every vertex of the path, an x-coordinate that is an even number and to every vertex a y-coordinate that corresponds to its topological order, i.e., $Y(v) = T(v)$ (Steps 1-4). The algorithm draws every edge $e = (u, v)$ as a straight line if the drawn edge does not intersect a vertex w different from u and v in Γ (Steps 5-7). Otherwise it draws edge e with one bend b_e such that: Its x-coordinate $X(b_e)$ is equal to $X(u)+1$ if $X(u) < X(v)$, or $X(u) - 1$ if $X(u) > X(v)$. The y-coordinate of bend b_e $Y(b_e)$

is equal to $Y(v) - 1$ (Steps 8-14).

Algorithm PB-Draw($G = (V, E)$, $S_p = \{P_1, P_2, ..., P_k\}$, $H = (V, A)$)
1. **For** $i = 0$ to $k - 1$ do
2. $X(P_i) = 2i$
3. **For** any $v \in V$
4. $(X(v), Y(v)) = (X(P_v), T(v))$
5. **For** any $e = (u, v) \in A$
6. **If** the straight line drawing of e does not intersect a vertex different from u, v:
7. Draw e as a straight line
8. **Else**:
9. **If** $X(u) < X(v)$:
10. $X(b_e) = X(u) + 1$
11. **Else**:
12. $X(b_e) = X(u) - 1$
13. $Y(b_e) = Y(v) - 1$
14. Draw e with one bend at point $(X(b_e), Y(b_e))$

Figure 3 shows an example of a drawing computed by Algorithm PB-Draw. In (a) we show the drawing of a graph G as computed by Tom Sawyer Perspectives which follows the Sugiyama framework. In (b) we show the drawing Γ of H computed by Algorithm PB-Draw. The path decomposition that we used to compute the drawing is $S_p = \{P_1, P_2, P_3\}$, where: $P_1 = \{0, 1, 4, 7, 12, 13, 15, 16, 17, 20, 22, 24, 25, 26, 29, 30\}$; $P_2 = \{2, 5, 9, 11, 23, 27\}$; $P_3 = \{3, 6, 8, 10, 14, 18, 19, 21, 28\}$. Edge $e = (21, 25)$ is the only one bending. In grey we show edge e drawn as straight line, intersecting vertex 23. Notice that edge $(21, 25)$ and edge $(21, 30)$ are adjacent and crossing edges. This effect can be disturbing for the user, but it can be simply avoided by incrementing the number of edges bending, i.e., in this case, we can simply add a bend to edge $(21, 30)$. As discussed before, the purpose of our figures is not to make a direct comparison between the drawings based on the Sugiyama Framework and the drawings based on our framework, but to show the output of our algorithms and the difference between the graph that we draw and the original graph that is usually drawn. Hence, a consistent scaling of the figures is not necessary for this and all other figures of this paper.

Any drawing Γ computed by Algorithm PB-Draw has several interesting properties. First, the area of Γ is typically less than $O(n^2)$. By construction, Γ has height $n - 1$ and width of $2k - 1$. Hence $Area(\Gamma) = O(kn)$. Given S_p and the topological order of the vertices of G, every vertex needs $O(1)$ time to be placed. Every edge $e = (u, v)$ needs $O(k)$ time to be placed, since before drawing it we need to check if its straight line drawing would intersect a vertex different from u, v (Step 6). Since the drawing of e must be monotone, it can intersect at most one edge per path, so we have to check if the drawing of e intersects some vertex in correspondence of every path placed between the path



(a) (b)

Figure 3: (a) Drawing of a DAG G computed by Tom Sawyer Perspectives (b) path based drawing of H computed by Algorithm PB-Draw.

of u and v in Γ. To check whether e intersects the path P at x-coordinate l_x, we compute the y-coordinate l_y where the straight-line drawing of e would intersect the vertical line containing P. Then check whether l_y is an integer, and if so, whether the vertex w with $T(w) = l_y$ belongs to path P. This takes constant time per path, presuming the inverse of $T(\cdot)$ is stored as array. Hence we have:

Theorem 1 *Algorithm PB-Draw computes a drawing Γ of a graph H based on the DAG G in $O(km)$ time. Furthermore, $Area(\Gamma) = O(kn)$.*

Lemma 1 *A cross edge $e = (u, v)$ does not intersect a vertex different from u and v in Γ.*

Proof: If e is drawn as a straight line the lemma is true by the construction of Algorithm PB-Draw. Otherwise, e is composed of two segments: (u, b_e) and (b_e, v). Both segments are diagonals of the rectangles inside which there is no vertices, so the two segments, and consequently e, cannot intersect any vertex different from u, v. □

Lemma 2 *Let $e = (u, v)$ and $e' = (u', v')$ be two cross edges drawn with a bend in Γ. Their bends are placed in the same point if and only if u and u' are in the same decomposition path and $v = v'$.*

Proof: We can observe that $v = v'$, since $Y(b_e) = Y(b_{e'}) = Y(v) - 1$ and since there is no vertex $w \neq v$ such that $Y(w) = Y(v)$. Therefore, u and u' are in the same side. We have $X(u) = X(u')$, since $X(b_e) = X(b_{e'}) = X(u) \pm 1 = X(u') \pm 1$. In this case u and u' are in the same path of the path decomposition. □

In the case described by the above lemma, two edges have overlapping segments (b_e, v) and $(b_{e'}, v)$. We consider this feature acceptable, or even desirable for two edges that have the same endpoint. This typical merging of edges has been used in the past, see for example [3, 21, 27]. However, in the case that this feature is not desirable, we propose two alternative solutions that avoid this overlap. The price to pay is either larger area, or fewer edges drawn:

1. Larger area option: We can shift horizontally by one unit the position of bend $b_{e'}$ and all the vertices v and bends b such that $X(v) > X(b_e)$ and $X(b) > X(b_e)$. In this case we have no overlaps, but the area of Γ can be as large as $O(knm)$.

2. Fewer edges option: We can define the path decomposition graph differently by removing some transitive cross edges from H. For every vertex v we remove the edge (u, v) if there exists an edge (u', v) such that u' and u are in the same decomposition path P and u precedes u' in the order of P. It is easy to prove that H' is a subgraph of H and that $A - A'$ contains only transitive edges of G. By definition of H', given a decomposition path P, for any vertex v there exists at most one cross edge $e = (u, v)$ such that $u \in P$. According to Lemma 2, there are no bends overlapping. The area of a drawing Γ computed using H' is $Area(\Gamma') = O(kn)$. However, we pay for the absence of overlapping bends by the exclusion from the drawing of some transitive cross edges of G.

In Figure 4 we show an example of the edge overlap described above in a drawing of H. Part (a) shows a simple drawing where two edges, $e_1 = (u, v)$ and $e_2 = (u', v)$, overlap. The edges in grey are alternative drawings of e_1 and e_2, respectively, as straight lines. Please notice that both of them intersect a vertex. In part (b) we shift horizontally the drawing, removing the overlap but, of course, increasing the area. Part (c) shows the drawing of H', where edge (u, v) is removed since u' has a higher order than u in their path.

Alternatively we propose to draw every cross edge adjacent to two vertices belonging to two different and not-consecutive channels with a bend. In this case we can avoid Step 6, so we can obtain the drawing in $O(n + m)$ time. Of course, we pay for the reduced time complexity by having more bends in the drawing.

Notice that graphs H and H' are computed from G by simply removing some transitive edges. Hence we have the following:

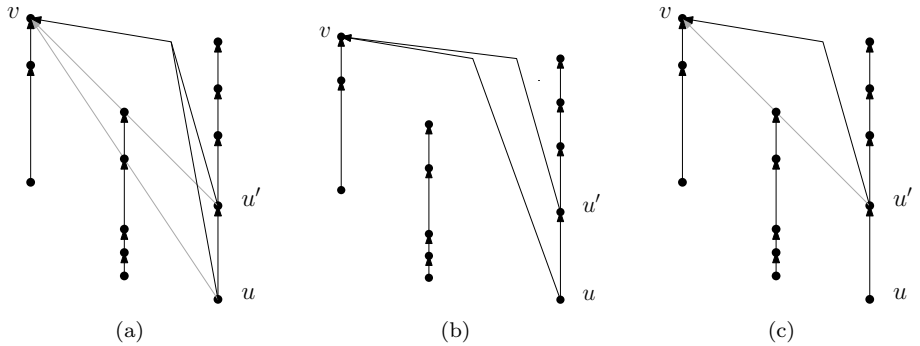


Figure 4: Examples of bend and edge overlaps in a drawing of H.

Theorem 2 *The path decomposition graphs H and H' have the same reachability properties of G.*

Theorem 2 is rather simple, but it is very important, since it tells us that in a visualization of a hierarchical drawing of H or H' we can read and understand correctly any reachability relation between the vertices of G.

Figure 5 shows a hierarchical drawing of a DAG G computed by Tom Sawyer Perspectives following the Sugiyama Framework. Figure 6 shows a path based drawing computed starting from G. This drawing represents one of the alternatives introduced in Figure 2 in a bigger scale: In this case the graph drawn is the graph H' described above and all the cross edges adjacent to two vertices belonging to two different and not-consecutive channels are drawn with one bend.

Algorithm PB-Draw works for any path decomposition S_p of a DAG G, since S_p is part of the input to the algorithm. S_p could be (a) the result of another process of some application, (b) computed/designed manually by a user, or (c) computed by an algorithm. However, a path decomposition S_p of a DAG G with a small number of paths lets us compute a readable path based drawing of H, since the number of decomposition paths influences the area of the drawing and its number of bends. Since a cross edge can intersect at most one vertex of every decomposition path, the number of decomposition paths influences the number of bends of the drawing. Furthermore, the number of paths k influences the time to find the minimum number of crossings between cross edges and paths, as it is described at the end of Section 5. Several algorithms solve the problem of finding a path decomposition of minimum size [17, 22, 25, 28]. The algorithm of [22] is the fastest one for sparse and medium DAGs. In Section 4 we introduce a relaxed definition of path, the channel, and a way to obtain hierarchical drawings based on a channel decomposition. Since paths are constrained versions of channels, we expect the minimum size of a channel decomposition to be lower than or in the worst case equal to the minimum size of a path decomposition. Now we turn our attention to the concept of channel decomposition.

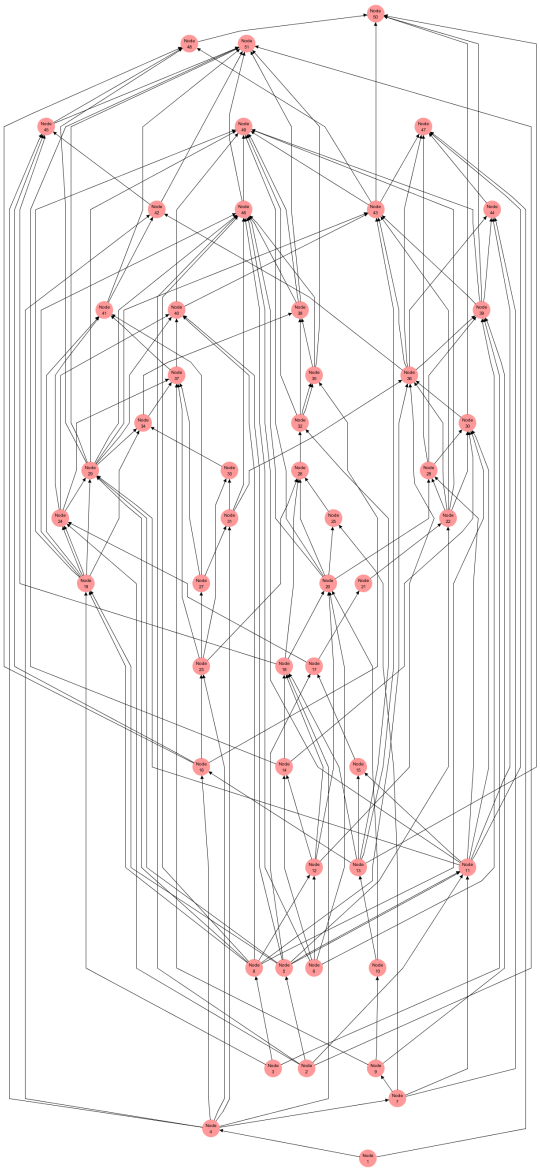


Figure 5: Hierarchical drawing computed by Tom Sawyer Perspectives following the Sugiyama framework.

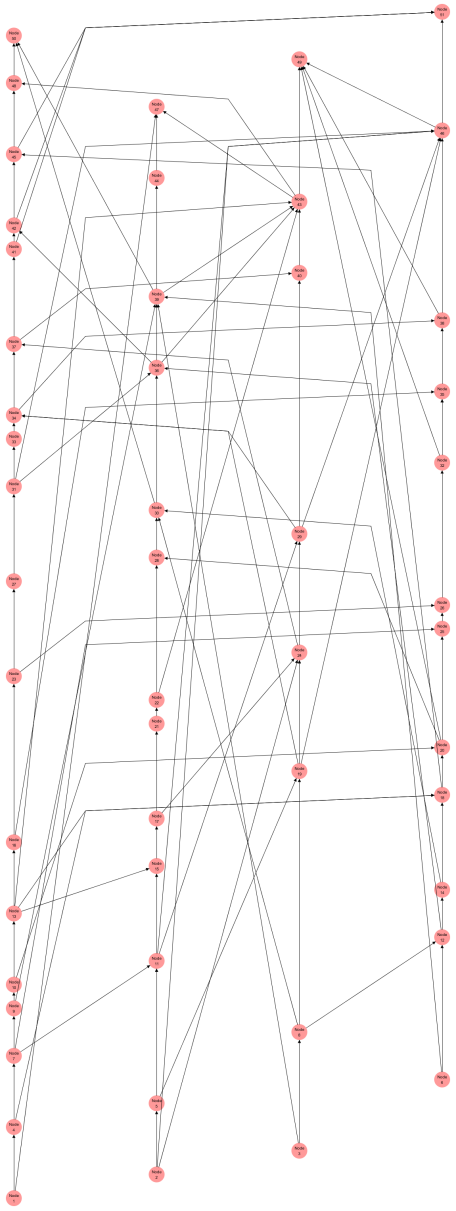


Figure 6: Path based drawing computed starting from the graph represented in Figure 5. This drawing represents one of the alternatives introduced in Figure 2 in a bigger scale: In this case the graph drawn is the graph H' described above and all the cross edges adjacent to two vertices belonging to two different and not-consecutive channels are drawn with one bend.

4 Channel Based Drawings

Let $G = (V, E)$ be a DAG. A *channel* C is an ordered set of vertices such that any vertex $u \in C$ is connected by a directed path to each of its successors in C. In other words, given any two vertices $v, w \in C$, v precedes w in the order of channel C if and only if w is reachable from v in G. A channel can be seen as a generalization of a path, since a path is always a channel, but a channel may not be a path (see, for example, $0 \to 1 \to 2$ in Figure 9, which allows 0 and 2 to be in one channel even though 1 is not). A *channel decomposition* $S_c = \{C_1, ..., C_k\}$ is a partition of the vertex set V of the graph into channels. We write v_i^j if v is the jth vertex of channel C_i. The channel decomposition graph $H'' = (V, A'')$ and a *channel based hierarchical drawing* (channel based drawing) of G are defined in a similar fashion as we defined the path decomposition graph H and the path based drawing of H in the previous section. Notice that, since the channel is a generalization of a path, the concepts of channel decomposition graph and channel based drawing are generalizations of the concepts of path decomposition graph and path based drawing.

A channel decomposition with a small number of channels lets us compute readable channel based drawings. The *width* b of a DAG G is the maximum cardinality of a subset of V of pairwise incomparable vertices of G, i.e., there is no path between any two vertices in the subset. In [9] it is proved that the minimum value of the cardinality of S_c, is b and in [18] an algorithm is given to compute S_c with $k = b$ in $O(n^3)$ time. The time complexity is improved to $O(bn^2)$ in [6]. Clearly, since paths are a restricted type of channels, the minimum size of S_c is less than or equal to the minimum size of a path decomposition S_p.

We can define Algorithm CB-Draw in a similar fashion as Algorithm PB-Draw, and its pseudocode is similar to the pseudocode of Algorithm PB-Draw. The only difference is that Algorithm CB-Draw takes as input a channel decomposition instead of a path decomposition and that its output is a channel based drawing instead of a path based drawing. Algorithm CB-Draw is clearly a generalization of Algorithm PB-Draw. Due to these similitudes we simply present CB-Draw without describing in details all its steps, as we did for PB-Draw.

Algorithm CB-Draw($G = (V, E)$, $S_c = \{C_1, C_2, ..., C_k\}$, $H'' = (V, A'')$)
1. **For** $i = 1$ to k do
2. $X(C_i) = 2i$
3. **For** any $v \in V$
4. $(X(v), Y(v)) = (X(C_v), T(v))$
5. **For** any $e = (u, v) \in A''$
6. **If** the straight line drawing of e does not intersect a vertex different from u, v:
7. Draw e as a straight line
8. **Else**:
9. **If** $X(u) < X(v)$:
10. $X(b_e) = X(u) + 1$
11. **Else**:

12. $X(b_e) = X(u) - 1$
13. $Y(b_e) = Y(v) - 1$
14. Draw e with one bend at point $(X(b_e), Y(b_e))$

We define the *jumping number* of a path in a channel based drawing as the number of times it "jumps" from a channel to another channel, or, in other words, the number of cross edges it contains. Intuitively the lower the jumping number is the higher is the readability of a path in the drawing. Figure 7 shows two paths on the same channel based drawing connecting the same two vertices, but having a different jumping number. The path in (a) has jumping number 3, while the path in (b) has jumping number 1. Clearly, the path of Part (b) is more readable.

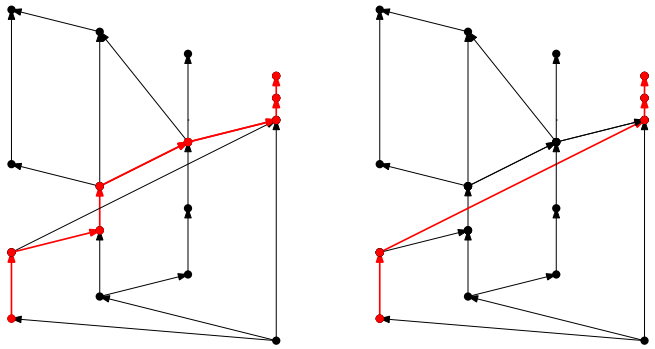


Figure 7: Two paths on the same channel based drawing connecting the same two vertices, but having a different jumping number. In (a) the jumping number is 3, in (b) the jumping number is 1.

In the rest of the section we introduce a "special" transitive closure, called compressed transitive closure, which is based on the concept of channel decomposition. This transitive closure is obtained from an ordinary transitive closure by removing some (usually many) of its transitive edges. Then, we will define a graph Q, based on the compressed transitive closure, that will let us obtain more readable drawings. Two vertices are *comparable* if there is a path connecting them. An interesting property of Q and its channel based drawing is that every pair of comparable vertices are connected by a (very readable) path having jumping number either 1 or 0.

The Compressed Transitive Closure Graph

Fix one channel decomposition $S_c = \{C_1, ..., C_k\}$; the concepts introduced in the following will depend on it. Let L_v be a list of vertices associated with a vertex $v \in V$ so that the following two properties hold: L_v contains at most one vertex of any decomposition channel; a vertex w is reached from v in G if and only

if L_v contains a vertex w' such that w and w' are in the same decomposition channel and w' precedes w in the order of that decomposition channel. The *compressed transitive closure* of G is the set of all the lists L_v. In [18] it is shown how to compute the compressed transitive closure of a graph in $O(km)$ time. Next we show how we can store it in $O(kn)$ space and that it contains the complete reachability information of G. For a given channel decomposition $S_c = \{C_1, ..., C_k\}$, the compressed transitive closure of a DAG is unique.

We define the *compressed transitive closure graph (CTC graph)* $Q = (V, I)$ such that $(u, v) \in I$ if and only if u is the highest vertex in the order of its channel such that $v \in L_u$. Notice that an edge of Q may not exist in the original graph G, as is the case in the ordinary transitive closure graph G^* of G. Furthermore, an edge of G may not be included in Q, while G^* contains all the edges of G. Please notice that Q has the same reachability properties (i.e., the same transitive closure) as G, since it is computed directly from the compressed transitive closure of G. We denote by *channel edge* an edge of Q connecting two vertices of the same channel, else it is a *cross edge*, similar to the definition of the previous section. Notice that the concepts introduced in this section depend on the given channel decomposition $S_p = \{C_1, ..., C_k\}$ and that, given a channel decomposition, the compressed transitive closure of a DAG is unique.

Figure 8 shows two examples pointing out the difference between G^* and Q. Part (a) shows a vertex u adjacent to two vertices of channel C_i in G^*, v_i^j and $w_i^{j'}$. It is adjacent only to the vertex v_i^j in the graph Q, since $j < j'$. Part (b) shows a graph G^* and the respective graph Q.

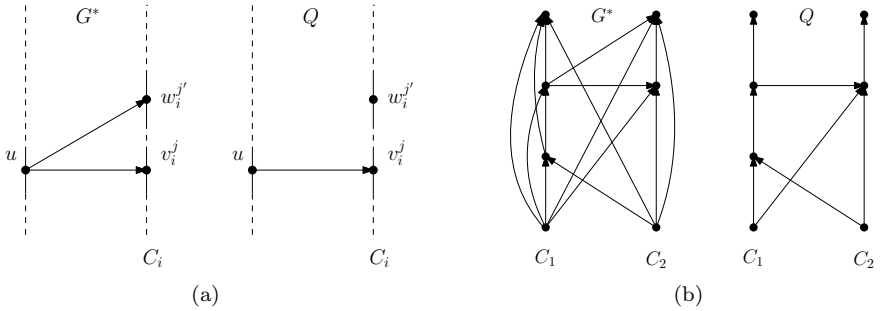


Figure 8: Two examples pointing out the difference between G^* and Q.

Let u_i^j be a vertex. The list L_u contains by definition the vertex v_i^{j+1}, since it is the lowest vertex in channel C_i reachable from u. Hence we have the following property:

Lemma 3 *$(u, v) \in I$ for any u_i^j, v_i^{j+1}.*

Lemma 3 implies that the channel decomposition S_c of G is a path decomposition of Q, so a channel based drawing of Q is essentially also a path drawing and hence we can compute it using Algorithm CB-Draw or Algorithm PB-Draw since in this case the two algorithms produce the same result.

Figure 9 shows an example of a channel based drawing of G computed by Algorithm CB-Draw using Q as an input is shown: Part (a) shows the original graph G drawn as computed by Tom Sawyer Perspectives that uses the Sugiyama framework. A channel decomposition of this graph is $S_c = \{C_1, C_2, C_3, C_4\}$, where: $C_1 = \{0, 2, 3, 7, 8, 12, 15, 16, 19\}$; $C_2 = \{1, 4, 9, 17\}$; $C_3 = \{5, 10, 13, 18\}$; $C_4 = \{6, 11, 14\}$. Part (b) shows the drawing of Q as computed by Algorithm CB-Draw. The dashed edges are edges that do not exist in G. Some channel edges are dashed, since a channel may not be a path of G.

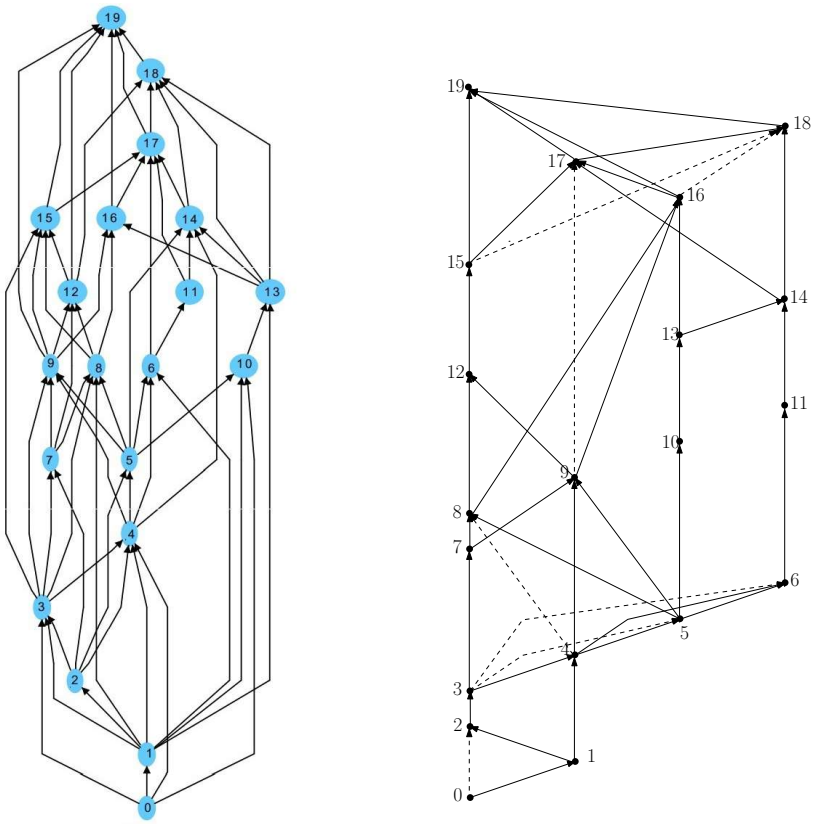


Figure 9: (a) Drawing of a DAG G computed by Tom Sawyer Perspectives (b) channel based drawing of Q computed by Algorithm CB-Draw.

There is one list L_v for every vertex v and every list contains $O(k)$ elements. Since every element of a list L_v corresponds to (at most) one edge of Q we have that Q contains $O(kn)$ edges. Hence we have the following:

Theorem 3 *Let $S_c = \{C_1, ..., C_k\}$ be a channel decomposition of a DAG G and let Q be the corresponding transitive closure graph. Graph Q has $O(kn)$ edges.*

The above theorem implies that the number of edges of Q is linear if k is a constant. Also, it requires only $O(kn)$ space to be stored.

Theorem 4 *Let $S_c = \{C_1, ..., C_k\}$ be a channel decomposition of a DAG G and let Q be the corresponding transitive closure graph. Two vertices of Q are comparable if and only if there exists a path connecting them having jumping number 0 or 1.*

Proof: Let u and v be two vertices. We want to prove that u reaches v in Q if and only if there exist a path connecting them having jumping number 0 or 1. Suppose that u and v are in the same channel C_i. In this case there exists a path having jumping number 0 from u to v as a consequence of Lemma 3. Suppose that u and v are in two different channels C_i and C_j. If u reaches v, by definition of Q, there must be a vertex $u' \in C_i$ that is a successor of u in C_i and a vertex $v' \in C_j$ which is predecessor of v in C_j, such that $(u', v') \in I$. Let P_1 and P_2 be the paths having jumping number 0 from u to u' and from v' to v. The path $P = P_1 + (u', v') + P_2$ is a path having jumping number 1 connecting u to v. □

A channel based drawing of Q is a very useful instrument to visualize the reachability properties of G. Indeed, if we want to check if a vertex reaches another vertex in Q (and consequently in G) we just need to check if there exists a path P having jumping number 0 or 1. Finding P in Γ is very easy since it is drawn on a vertical line or in two vertical lines connected by a cross edge. Moreover since Q has an almost-linear number of edges ($O(kn)$) by Theorem 3 it makes Q easier to visualize and so it gives us a clear way to visualize the reachability properties of G. The price we have to pay is that we do not visualize many edges of the original graph G. These edges can be visualized on demand by moving the mouse over a given query vertex. A full implementation of this technique will be presented in the near future.

5 Constrained Minimum Size Channels Decomposition

In this section we consider that a user may want to use specific "user-defined" channels as part of the full channel decomposition. Let $S^* = \{C_1, ..., C_h\}$ be a set of (user-defined) disjoint channels of G. We will present an algorithm to compute a channel decomposition S_c^* with the minimum number of channels among all the channel decompositions of G containing S^*.

First we present Algorithm Remove-Channel (RC) which takes as input a graph G and a channel C_i and produces as output a graph $G' = (V', E')$ such that $V' = V - C_i$. We will prove that it computes E' such that the reachability relation between any pair of vertices of V' is the same in G and G'. Firstly G' is initialized with G (Step 1). Then, for every vertex of the channel C_i, starting from its source and proceeding in order, we add a transitive edge connecting all its adjacent incoming vertices to all its adjacent outgoing vertices (Step 2-4)

and then we remove it (Step 6).

Algorithm Remove-Channel(G, C_i):
1. $G' = (V', E') = G$:
2. **For** any $j = 1, ..., |C_i|$:
3. **For** any pair of edges (v, u_i^j) and (u_i^j, w) of E'
4. $E'.add((v, w))$
5. $G'.remove(u_i^j)$
6. *output*: G'

Lemma 4 *Vertex w is reachable from v in G' if and only if w is reachable from v in G.*

Proof: Let $P = (V_P, E_P) \in G$ be a path from v to w and $q = |C_i \cap V_P|$. We proceed by induction on q. *Base case:* If $q = 0$ then $P \in G'$. Suppose $q = 1$. Let $x, y, z \in V_p$ be three vertices so that $z \in C_i$ and $(x, z), (z, y) \in E_P$. The path $P' = (V_P - z, E_P - (x, z) - (z, y) + (x, y))$ is a path from v to w so that $P' \in G'$. *Inductive case:* Suppose that the lemma is true for $q = N$ and suppose $q = N + 1$. Let $P_N \in P$ be a path from v to a vertex $z \in C_i$ containing N vertices of C_i and $P_1 = P - P_N + z$. Let G'_N be the graph computed during Algorithm Remove-Channel after the removal of the vertex precedent to z in C_i. By the inductive hypothesis there exists $P'_N \in G'_N$ connecting v and z. The path $P'_N \cup P_1$ connects v and w and contains on exactly 1 vertex of C_i, which is z. We can prove as we proved the base case $q = 1$ that in this case exists $P' \in G'$ connecting v and w. □

Let $G = (V, E)$ be a DAG, let b be its width and let Minimum-Channels-Selection (MCS) be an algorithm that, given a DAG G, computes a channel decomposition of size b in $O(t)$ time (where, $t = n^3$ if we use the algorithm described in [18] and $t = bn^2$ if we use the algorithm described in [6]).

Now we define a constrained version of Algorithm MCS, called Constrained-Minimum-Channels-Selection (CMCS). Algorithm CMCS computes a channel partition S_c^* with the minimum number of channels among all the channel partitions of G containing S^*. Firstly it removes all the channels of S^* using Algorithm RC (Steps 1-2). Then it computes the minimum size channel decomposition S' of the resulting graph using Algorithm MCS (Step 3) and finally it computes S_c^* merging S' and S_c^* (Step 4).

Algorithm Constrained-Minimum-Channels-Selection(G, S^*):
1. for any $C_i \in S^*$
2. $G' =$ Remove-Channel(G,C_i)
3. $S'_c = MCS(G')$
4. $S_c^* = S'_c + S^*$
5. *output*: S_c^*

Figure 10 depicts an illustration of Algorithm CMCS. In this figure consecutive vertices of a same channel are connected by dotted edges for a better

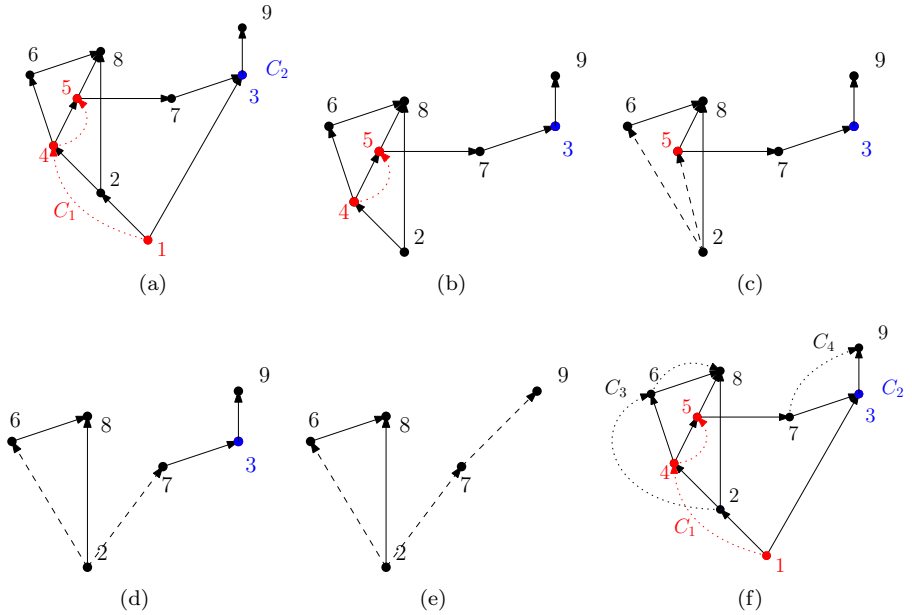


Figure 10: Illustration of Algorithm Constrained-Minimum-Channels-Selection.

visualization of the channels. Part (a) shows the input DAG G and user-defined channels $S^* = \{C_1, C_2\}$, where $C_1 = (1, 4, 5)$ (red) and $C_2 = (3)$ (blue). Channel C_1 is not a path, since vertices 1 and 2 are not adjacent. Channel C_2 is composed by a single vertex. The three vertices of C_1 and the vertex of C_2 are removed by Algorithm RC during the Steps 1-2 of Algorithm $CMCS$. In (b), (c), and (d) we show graph G' as computed during the steps of Algorithm Remove-Channels. The dashed edges are the edges added by Algorithm RC. Part (e) shows graph G' after the removal of S^* is completed. A channel decomposition of minimum size of G' is $S' = \{C_3, C_4\}$, where $C_3 = (2, 6, 8)$ and $C_4 = (7, 9)$. S' is computed by Step 3. Part (f) shows $S_c^* = \{C_1, C_2, C_3, C_4\}$, the channel decomposition output of Algorithm $CMCS$, computed by Step 4.

Theorem 5 *Let G be a DAG and S^* be a set of vertex-disjoined channels of G. $S_c^* = MCS(G, S^*)$ is a channel decomposition having the minimum number of channels among all channel decompositions of G containing set S^*.*

Proof: Suppose that there exists a channel decomposition S_x of G containing S^* and so that $|S_x| < |S_c|$. Let $S'_x = S_x - S^*$. It is straightforward to see that by Lemma 4 S'_x is a channel decomposition of G'. Hence, we have $|S'_x| = |S_x| - |S^*| < |S'_c| = |S_c| - |S^*|$ which is a contradiction, since S'_c is minimum because it is computed using Algorithm MCS. □

6 Comparisons and Conclusions

We presented a new framework for computing hierarchical drawings of digraphs. Because of the different nature of our framework we would expect that it produces results that are superior to the results produced by the Sugiyama framework with respect to the number of crossings, number of bends, area of the drawing and visual clarity of the existing paths and reachability. The hierarchical drawings produced by the Sugiyama framework have (a) many crossings (a bound is not possible to be computed), (b) the total number of bends can be relatively large and it depends heavily on the number of dummy vertices introduced, (c) the area is large because the width of the drawing is negatively influenced by the number of dummy vertices, (d) the number of bends per edge is also influenced by the number of dummy vertices on it (although the last phase tries to straighten the edges by aligning its segments, at the expense of the area, of course), (e) most problems and subproblems of each phase are NP-hard, and many of the heuristics are very time consuming, and (f) the reachability information in the graph is not easy to detect from the drawing.

We would expect the results produced by our framework to be superior to the results produced by the Sugiyama framework, based on the following theoretical observations: (a) the number of edge crossings is expected to be lower due to the grouping of edges into paths/channels and the omission of the corresponding transitive edges, (b) the total number of bends is low since we introduce at most one bend for some (not all) cross edges, (c) the area is precisely bounded by a rectangle of height $n - 1$ and width $O(k)$, where k is typically a small fraction of n, (d) the reachability and path information is easily visible in our drawings since any path is deduced by following at most one cross edge (which might have at most one bend), (e) the vertices in each channel are vertically aligned and there is a path from each vertex in the channel to all other vertices in the channel that are at higher y-coordinates, (f) all our algorithms run in polynomial time, and finally, (g) the flexibility of our framework allows a user to decide to have their specified paths/channels, thus allowing for user-defined paths/channels to be drawn vertically aligned. The height of our drawings can be further reduced by performing a one-dimensional compaction.

The only drawback of the drawings produced by our framework is the fact that it does not draw all the edges of the graph, which might be important for some applications. This might be considered a disadvantage by some users. However, in medium to large graphs, the drawing of all edges significantly clutters the drawing. Hence, the omission of some edges might be considered as an advantage by some other users since it offers drawings that are not cluttered by the edges. In any case, we offer the remedy to visualize all the edges incident to a vertex interactively when the mouse is placed on top of a vertex.

We believe that the above theoretical comparison is convincing of the power of the new framework. Additionally, as discussed before, the framework is not a single precise algorithm and hence, it is not possible to present experimental results here. However, in the future we plan to customize the framework in

order to produce precise algorithms, according to different criteria, and perform experiments. The flexibility of this framework is of paramount importance as illustrated in the figures of the introduction. Namely, if we look at the alternative hierarchical drawings presented in Figures 1 and 2, we realize that there is an interesting interplay between the number of bends, crossings, clarity, reachability comprehension, and edge bundling. We plan to investigate the interplay of various options in the future. Furthermore, we plan to perform user studies in order to verify that the users will benefit from the aforementioned properties by showing higher understanding and ease of use of the new drawing framework. Additionally, we need to comprehend human understanding issues related to the removal of some transitive edges (removing clutter from the drawing) and increasing reachability comprehension. It would be interesting to find specific topological orderings and/or sophisticated layer assignment techniques that will reduce the height, the number of crossings and the number of bends of the computed drawing. Another direction could be the development of techniques to compute the path decomposition that minimizes the number of cross edges or the sum of the length of the cross edges. Finally, it would be interesting to find various techniques for path/channel decomposition, whose aim would be to reduce the number of crossings and bends.

Acknowledgement: The authors acknowledge the help of Panagiotis Lionakis who provided some drawings that are produced by his preliminary implementation work.

References

[1] Tom Sawyer Software. URL: www.tomsawyer.com.

[2] yWorks. URL: www.yworks.com.

[3] M. J. Bannister, D. A. Brown, and D. Eppstein. Confluent orthogonal drawings of syntax diagrams. In E. Di Giacomo and A. Lubiw, editors, *Graph Drawing and Network Visualization - 23rd International Symposium, GD 2015, Los Angeles, CA, USA, September 24-26, 2015, Revised Selected Papers*, Lecture Notes in Computer Science, pages 260–271, 2015. doi:10.1007/978-3-319-27261-0_22.

[4] U. Brandes and B. Köpf. Fast and simple horizontal coordinate assignment. In *Graph Drawing, 9th International Symposium, GD 2001 Vienna, Austria, September 23-26, 2001, Revised Papers*, pages 31–44, 2001. doi:10.1007/3-540-45848-4_3.

[5] C. Buchheim, M. Jünger, and S. Leipert. A fast layout algorithm for k-level graphs. In *Graph Drawing, 8th International Symposium, GD 2000, Colonial Williamsburg, VA, USA, September 20-23, 2000, Proceedings*, pages 229–240, 2000. doi:10.1007/3-540-44541-2_22.

[6] Y. Chen and Y. Chen. On the dag decomposition. *British Journal of Mathematics and Computer Science*, 2014. 10(6): 1-27, 2015, Article no.BJMCS.19380, ISSN: 2231-0851. URL: https://www.researchgate.net/publication/285591312_Pre-Publication_Draft_2015_BJMCS_19380.

[7] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, and P. Mutzel. The open graph drawing framework (OGDF). In *Handbook on Graph Drawing and Visualization.*, pages 543–569. 2013.

[8] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.

[9] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Ann. Math. Ser.*, 251:161 – 166, 01 1950. doi:10.1007/978-0-8176-4842-8_10.

[10] P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994. doi:10.1007/BF01187020.

[11] M. Eiglsperger, M. Siebenhaller, and M. Kaufmann. An efficient implementation of Sugiyama's algorithm for layered graph drawing. In J. Pach, editor, *Graph Drawing*, pages 155–166, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/978-3-540-31843-9_17.

[12] M. Fröhlich and M. Werner. Demonstration of the interactive graph-visualization system *da Vinci*. In *Graph Drawing, DIMACS International Workshop, GD '94, Princeton, New Jersey, USA, October 10-12, 1994, Proceedings*, pages 266–269, 1994. doi:10.1007/3-540-58950-3_379.

[13] E. R. Gansner, E. Koutsofios, S. C. North, and K. Vo. A technique for drawing directed graphs. *IEEE Trans. Software Eng.*, 19(3):214–230, 1993. doi:10.1109/32.221135.

[14] E. R. Gansner, E. E. Koutsofios, and S. C. North. Drawing graphs with dot. 2015. URL: https://www.graphviz.org/pdf/dotguide.pdf.

[15] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Softw., Pract. Exper.*, 30(11):1203–1233, 2000. doi:10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.0.CO;2-N.

[16] M. Himsolt. Graphlet: design and implementation of a graph editor. *Softw., Pract. Exper.*, 30(11):1303–1324, 2000. doi:10.1002/1097-024X(200009)30:11<1303::AID-SPE341>3.0.CO;2-3.

[17] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

[18] H. V. Jagadish. A compression technique to materialize transitive closure. *ACM Trans. Database Syst.*, 15(4):558–598, 1990. doi:10.1145/99935.99944.

[19] M. Jünger, P. Mutzel, and C. Spisla. A flow formulation for horizontal coordinate assignment with prescribed width. In *Graph Drawing and Network Visualization - 26th International Symposium, GD 2018, Barcelona, Spain, September 26-28, 2018, Proceedings*, pages 187–199, 2018. doi:10.1007/978-3-030-04414-5_13.

[20] M. Kaufmann and D. Wagner. Drawing graphs: Methods and models. *LNCS vol. 2025*, 2001. doi:10.1007/3-540-44969-8.

[21] E. M. Kornaropoulos and I. G. Tollis. Algorithms and bounds for overloaded orthogonal drawings. *Journal of Graph Algorithms and Applications*, 20(2):217–246, 2016. doi:10.7155/jgaa.00391.

[22] A. Kuosmanen, T. Paavilainen, T. Gagie, R. Chikhi, A. I. Tomescu, and V. Mäkinen. Using minimum path cover to boost dynamic programming on dags: Co-linear chaining extended. In *Research in Computational Molecular Biology - 22nd Annual International Conference, RECOMB 2018, Paris, France, April 21-24, 2018, Proceedings*, pages 105–121, 2018. doi:10.1007/978-3-319-89929-9_7.

[23] L. Li, W. Hua, and X. Zhou. HD-GDD: high dimensional graph dominance drawing approach for reachability query. *World Wide Web*, 20(4):677–696, 2017. doi:10.1007/s11280-016-0407-z.

[24] N. S. Nikolov and P. Healy. *Hierarchical Drawing Algorithms, in Handbook of Graph Drawing and Visualization, ed. Roberto Tamassia.* CRC Press, 2014. pp. 409-453.

[25] J. B. Orlin. Max flows in O(nm) time, or better. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774, 2013. doi:10.1145/2488608.2488705.

[26] F. N. Paulisch and W. F. Tichy. EDGE: an extendible graph editor. *Softw., Pract. Exper.*, 20(S1):S1, 1990. doi:10.1002/spe.4380201307.

[27] S. Pupyrev, L. Nachmanson, and M. Kaufmann. Improving layered graph layouts with edge bundling. In U. Brandes and S. Cornelsen, editors, *Graph Drawing - 18th International Symposium, GD 2010, Konstanz, Germany, September 21-24, 2010. Revised Selected Papers*, Lecture Notes in Computer Science, pages 329–340, 2010. doi:10.1007/978-3-642-18469-7_30.

[28] C. Schnorr. An algorithm for transitive closure with linear expected time. *SIAM J. Comput.*, 7(2):127–133, 1978. doi:10.1137/0207011.

[29] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Systems, Man, and Cybernetics*, 11(2):109–125, 1981. doi:10.1109/TSMC.1981.4308636.