A Branch-and-Cut Approach to the Directed Acyclic Graph Layering Problem

Patrick Healy and Nikola S. Nikolov

CSIS Department, University of Limerick, Limerick, Republic of Ireland
{patrick.healy,nikola.nikolov}@ul.ie

Abstract. We consider the problem of layering Directed Acyclic Graphs, an $\mathcal{NP}$-hard problem. We show that some useful variants of the problem are also $\mathcal{NP}$-hard. We provide an Integer Linear Programming formulation of a generalization of the standard problem and discuss how a branch-and-bound algorithm could be improved upon with cutting planes. We then describe a separation algorithm for two classes of valid inequalities that we have identified – one of which is facet-defining – and discuss their efficacy.

1 Introduction

Drawing graphs in a hierarchical manner is a well-established style of presentation of relationships between entities, and in many disciplines it is the expected format. The most widely used method of drawing graphs hierarchically is, perhaps, the Sugiyama method [6]. Conceptually, the entities (nodes) should be drawn on a series of levels such that no two nodes that are related should be situated on the same layer and every entity relation (represented by a directed edge) should respect the direction of the relation by pointing in a uniform direction. To achieve this, the three main steps of the method are a) partitioning the node set into edgewise independent subsets so that all inter-partition edges can point the same direction; b) ordering the vertices of each subset in order to minimize edge crossings; and, c) determining a location on the level assigned to each vertex that maximizes clarity of the drawing.

In this paper we focus on the first step. This is called the layering problem since the objective is to assign each node of the diagram to a (usually horizontal) layer such that all of the directions of the edges point in the same direction and thus preclude assigning to the same layer a pair of vertices that are related *viz.* connected by an edge. Currently three algorithms, Longest Path, Coffman-Graham and Gansner et al.'s, predominate for layering a graph. Healy and Nikolov [4] performed an experimental study of these three algorithms. Although the algorithms are fundamentally incomparable since they layer according to different objectives, *Gans* was found to produce the best layerings judged by a variety of measurements in this study.

It is our contention that the algorithms currently in use are inadequate because of the simplified model that they assume, and evidence we have gathered

M.T. Goodrich and S.G. Kobourov (Eds.): GD 2002, LNCS 2528, pp. 98–109, 2002.

on the algorithms' performance supports this view [4]. In the same paper we proposed a polyhedral approach to the layering problem that reflected more of the problem's complexities and we defined the graph layering polytope. We also proposed a system `ULair` which used the ILP solver of CPLEX 7.0. `ULair` in that context was used as a reference point for the other algorithms' performance and its running time was not a central focus. In this paper we present a new branch-and-cut version of `ULair` which contains a cutting plane algorithm for generating cuts based on valid inequalities of the graph layering polytope.

The next section describes the layering problem formally. In Section 3 we introduce an Integer Linear Programming model and discuss capacity-related facets of the graph layering polytope and a cutting-plane algorithm. Section 4 describes the experiments that we conducted to evaluate the performance of the branch-and-cut version of `ULair`. We make some concluding remarks in Section 5.

2 The Layering Problem

Consider a directed acyclic graph (DAG) $G = (V, E)$ and a function $w : V \to \mathbb{Q}^+$ which represents the widths of the smallest enclosing rectangles of the nodes of G. We *layer* a graph as follows.

Partition the node set V of G into $\chi \geq 1$ layers $V_1, V_2, \ldots, V_\chi$, such that if $(u, v) \in E$ with $u \in V_j$ and $v \in V_i$ then $i < j$. Then for each edge $e = (u, v) \in E$ with $u \in V_j$ and $v \in V_i$ add so-called *dummy nodes* d_e^{i+1}, d_e^{i+2}, $\ldots$, d_e^{j-1} to layers V_{i+1}, V_{i+2}, $\ldots$, V_{j-1} respectively, replacing edge (u, v) by the directed path $(u, d_e^{j-1}, \ldots, d_e^{i+1}, v)$. The dummy nodes are normally not presented in the final drawing; their introduction is required by the algorithms applied at the next phases of the Sugiyama's algorithmic framework. We assign width $w_e^d \in \mathbb{Q}$ to all the dummy nodes contributed by edge e by extending the node width function w over the dummy nodes. We also consider w_e^d as the width of edge e.

Such a partitioning of the nodes of G is known as a *layering* of G and we denote it by $\mathfrak{L}(G)$. A DAG with a layering is called a *layered digraph*. An algorithm that partitions the DAG node set into layers is called a *layering algorithm*. Once we have separated the nodes of G into layers the width and the height of the final drawing can be approximated by the *width* and the *height* of the layering. The height of a layering is the number of layers. We define the layering width as follows.

Definition 1. *The width of layer* V_i *is* $\mathfrak{w}_i = \sum_{v \in V_i} w(v)$ *and the width of layering* $\mathfrak{L}(G)$ *is* $\mathfrak{w} = \max\{\mathfrak{w}_i : i = 1, \ldots, \chi\}$.

Usually, the width of a layering is defined as the maximum number of nodes in a layer [3,1]. Our definition generalizes this by allowing variable node and edge widths but if all the real nodes have unit width, and the edges (i.e. the dummy nodes) have zero width, then our definition for layering width is identical to the conventional one.

We call the problem of finding a layering of a DAG with bounded sum of edge spans and bounded dimensions *DAG layering*. Considered as a decision problem, DAG layering can be formulated as follows.

DAG Layering
Instance: A DAG $G(V, E)$ with a width function $w : V \cup E \to \mathbb{Q}^+$, and three numbers $H, S \in \mathbb{N}$, $W \in \mathbb{Q}^+$.
Question: Is there a layering of G with sum of edge spans at most S, height at most H and width at most W?

DAG layering is $\mathcal{NP}$-complete since the simpler decision problem when edge spans are ignored is $\mathcal{NP}$-complete [3]. There is no known exact algorithm for solving DAG layering. If the widths of edges – and thus, the dummy nodes they generate – are significant then DAG layering becomes even more difficult [2]. The three layering algorithms referred to earlier solve DAG layering approximately by considering the DAG layering constraints only partially.

DAG layering remains $\mathcal{NP}$-complete when the relative order of two *independent nodes* (i.e. nodes without a direct path between them) is given. We consider the following two problems – to which partial solutions will be used later in our cutting-plane generation algorithm.

Same-Layer DAG Layering.
Instance: A DAG $G(V, E)$ with a width function $w : V \cup E \to \mathbb{Q}^+$, three numbers $H, S \in \mathbb{N}$, $W \in \mathbb{Q}^+$, and two independent nodes $u, v \in V$.
Question: Is there a layering of G on at most H layers, with total sum of edge spans at most S, of width at most W, and with node u placed in the same layer as node v?

Theorem 1. *Same-layer DAG layering is $\mathcal{NP}$-complete.*[1]

Above DAG Layering.
Instance: A DAG $G(V, E)$ with a width function $w : V \cup E \to \mathbb{Q}^+$, and three numbers $H, S \in \mathbb{N}$, $W \in \mathbb{Q}^+$, and two independent nodes $u, v \in V$.
Question: Is there a layering of G on at most H layers, with total sum of edge spans at most S, of width at most W, and with node u placed in a layer above node v, i.e. if $u \in V_j$ and $v \in V_i$ then $i < j$?

Theorem 2. *Above DAG layering is $\mathcal{NP}$-complete.*

3 An ILP Model of the DAG Layering Problem

A starting point of our polyhedral approach to DAG layering as an optimization problem is the following ILP model – which we call *WHS-Layering* (W for *width*, H for *height* and S for *spans*).

[1] Proofs of theorems not included here are available upon request.

3.1 The ILP Model *WHS-Layering*

Consider a DAG $G = (V, E)$ and let x be the incidence vector of a subset of $V \times \{1, \ldots, H\}$ then *WHS-Layering* is the following ILP model.

$$\min \sum_{(u,v)\in E} \left(\sum_{k=\varphi(u)}^{\rho(u)} k x_{uk} - \sum_{k=\varphi(v)}^{\rho(v)} k x_{vk} \right) \tag{1}$$

Subject to

$$\sum_{k=\varphi(v)}^{\rho(v)} x_{vk} = 1 \qquad \forall v \in V \tag{2}$$

$$\sum_{i=\varphi(u)}^{k} x_{ui} + \sum_{i=k}^{\rho(v)} x_{vi} \leq 1 \qquad \forall k \in L(u) \cap L(v),\ \forall (u,v) \in E \tag{3}$$

$$\sum_{v\in V_k^*} w_v x_{vk} + \mathcal{D}_k \leq W \quad \forall k = 1, \ldots, H \tag{4}$$

$$\sum_{v\in V_k^*} x_{vk} \geq 1 \qquad \forall k = 1, \ldots, \pi(G) \tag{5}$$

all x_{vk} are binaries

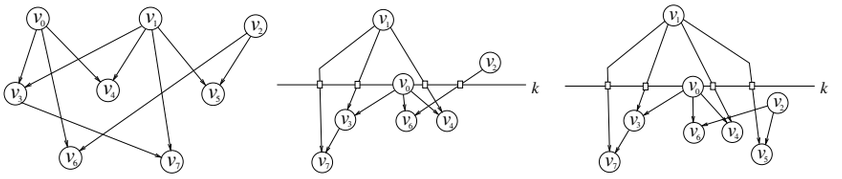
In this formulation W and H are upper bounds on the width and height of the layering respectively; $\pi(G)$ is the number of nodes in the longest directed path in G; $\varphi(v)$ and $\rho(v)$ are respectively the lowest and the highest layer where node v can be placed in; $L(v) = \{\varphi(v), \ldots, \rho(v)\}$ and $V_k^* = \{v \in V : \varphi(v) \leq k \leq \rho(v)\}$. The objective minimises the sum of edge spans, i.e. the number of dummy nodes. Equalities (2) force each node to be placed in exactly one layer; we call them *assignment constraints*. Inequalities (3) force each edge to point downwards; thus we call them *direction constraints*. Inequalities (5), called *fixing constraints*, introduce the additional requirement of having at least one node in the first $\pi(G)$ layers. This reduces the number of identical layerings (but shifted vertically) if the height of the solution is less than the upper bound H.

Inequalities (4), called *capacity constraints*, restrict the width of each layer (including the dummy nodes) to be less than or equal to W: the first term on the left hand side represents the contribution of the real nodes to the width of layer V_k while $\mathcal{D}_k$ represents the contribution of the dummy nodes. We set

$$\mathcal{D}_k = \sum_{e=(u,v)\in E} w_e^d \left(\sum_{l>k}^{\rho(u)} x_{ul} - \sum_{l\geq k}^{\rho(v)} x_{vl} \right)$$

where w_e^d is the width of the dummy nodes along edge e. The difference of the two sums in the parentheses is 1 if edge $e = (u, v)$ spans layer V_k and 0 otherwise.

We refer to the constraint polytope of *WHS-Layering* as the *DAG layering polytope*. To study its properties we relax it by allowing layerings of subgraphs



(a) A random drawing of the DAG G_2

(b) Node v_2 placed above layer V_k

(c) Node v_2 placed below layer V_k

Fig. 1. Node v_1 placed above node v_0 in a layering of the DAG G_2

of the DAG G. We do this by replacing equalities (2) by "less-than-or-equal" inequalities. In the sections below when we refer to the DAG layering polytope we mean its relaxed version.

3.2 Generation of Valid Inequalities Related to the Layering Width

The upper bound W imposed on the width of the layering gives rise to a set of *capacity-related* valid inequalities for the DAG layering polytope. In previous work we identified strong relative ordering valid inequalities based on pairs of independent nodes, one of which had to be placed above the other; we called these strong RO inequalities [4]. In this section we show how, on the basis of heuristic algorithms which solve Same-layer DAG layering and Above DAG layering, we can strengthen them.

Before we proceed we show an example, which gives an insight on our approach to finding capacity-related valid inequalities of the DAG Layering polytope.

Example 1. Let G_2 be the DAG depicted in Figure 1(a). Consider the problem of layering G_2 of width at most 4 assuming all the nodes (including the dummy nodes) have a unit width. The drawings in Figure 1(b) and (c) show how placing node v_1 in a layer above node v_0 will affect the width of the layering. Let v_0 be placed in layer V_k. Then its successors v_3, v_4, v_6 and v_7 must be placed in the layers below V_k. Thus the edges connecting v_1 to them will cause three dummy nodes in layer v_k (the dummy nodes are depicted by small rectangles in Figure 1(b) and (c)). Since node v_2 is neither a predecessor nor a successor of v_0 and v_1, placing v_1 above v_0 does not determine the relative position of v_2 to layer V_k. There are three different cases which we should consider. The first case is when v_2 is placed in layer V_k. Then layer V_k will have width at least 5, which makes this case impossible. Second, let v_2 be placed above layer V_k. This case is depicted in Figure 1(b). Then edge (v_2, v_6) will contribute another dummy node to layer V_k and this way making this case also impossible. The last alternative is to have node v_2 placed below layer V_k. Then its successor v_5 must be also below layer V_k. This case is depicted in Figure 1(c). Then edge (v_1, v_5) will contribute

a dummy node to layer V_k, which shows that this case is not possible either. Thus v_1 cannot be placed above node v_0 for $W \leq 4$.

Consider two independent nodes u and v of a DAG $G = (V, E)$ and assume we have a heuristic procedure $P_{SAME-LAYER}(u, v, S, H, W)$ for solving Same-layer DAG layering, which takes a decision by considering only the width of the layer where nodes u and v are placed and the widths of its neighboring layers above and below. If $P_{SAME-LAYER}$ detects a layer of width greater than W it returns a negative answer and otherwise it returns a positive answer; clearly it will fail to detect some bad layerings due to its shortsightedness. If $P_{SAME-LAYER}$ returns a negative answer we say that it detects the pattern depicted in Figure 2(a) as an infeasible pattern in a layering of G.

Similarly, assume we have a heuristic procedure $P_{ABOVE}(u, v, \gamma, S, H, W)$ for solving Above DAG layering for a given gap, γ, between the layers of u and v. We consider three cases of gaps between layers which correspond to three patterns which P_{ABOVE} may detect as infeasible in a layering of G. The three cases are: $\gamma = 1$ when u and v are in adjacent layers (illustrated in Figure 2(b)); $\gamma = 2$ when there is exactly one layer between the layers of u and v (illustrated in Figure 2(d)); and $\gamma = 3$ when there are two or more layers between the layers of u and v (illustrated in Figure 2(h)). P_{ABOVE} decides by considering only the widths of the two layers where nodes u and v are placed respectively and the widths of their adjacent layers. In all cases P_{ABOVE} considers the two adjacent layers above and below the layer of v and the two adjacent layers above and below the layer of u. Thus, the number of layers whose widths are considered are 4 ($\gamma = 1$), 5 ($\gamma = 2$), or 6 ($\gamma = 3$). We call the four patterns considered by $P_{SAME-LAYER}$ and P_{ABOVE} *basic infeasible patterns*.

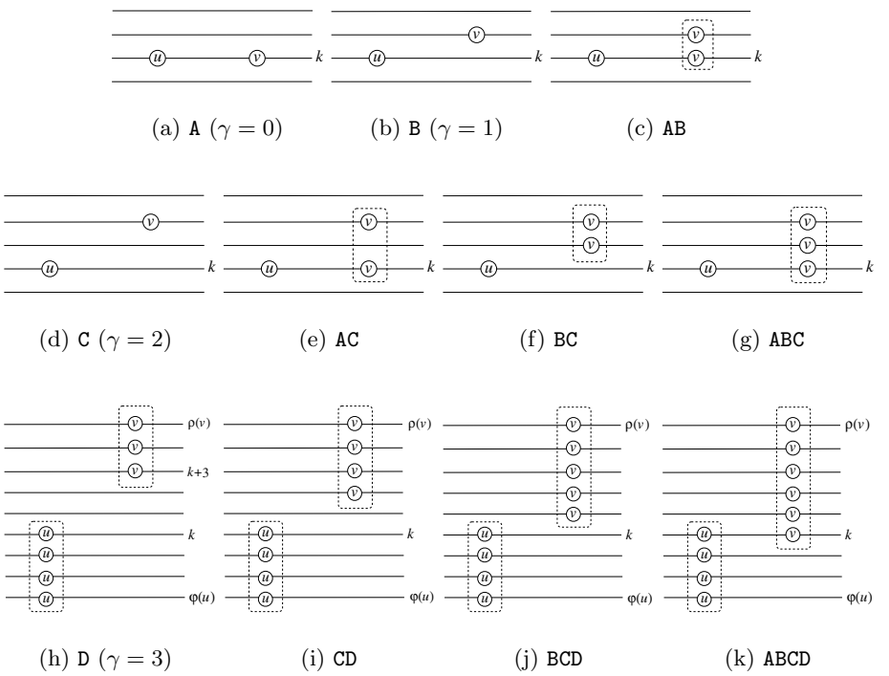
If more than one of the four infeasible basic patterns are detected simultaneously then that gives rise to combined infeasible patterns. Of the 15 possible combinations we consider the 11 illustrated in Figure 2 which correspond to generic types of valid inequalities for the DAG layering polytope. The first seven infeasible combined patterns give rise to the *fixed-gap (FG) layering inequalities*

$$x_{uk} + \sum_{i=0}^{2} \alpha_i x_{vk+i} \leq 1 \tag{6}$$

with $k \in L(u)$, $\alpha_i \in \{0, 1\}$, $\sum_{i=0}^{2} \alpha_i \geq 1$, and $k + i \in L(v)$ if $\alpha_i = 1$. Similarly, the last four infeasible combined patterns give rise to the valid inequalities

$$\sum_{i=\varphi(u)}^{k} x_{ui} + \sum_{i=k+\gamma}^{\rho(v)} x_{vi} \leq 1 \tag{7}$$

with $k \in L(u)$, $\gamma \in \{0, 1, 2, 3\}$, and $k + \gamma \in L(v)$. These are the inequalities that generalize the strong RO inequalities [4] and we call them simply *RO inequalities* in the remainder of this paper.



(a) `A` ($\gamma = 0$) (b) `B` ($\gamma = 1$) (c) `AB`

(d) `C` ($\gamma = 2$) (e) `AC` (f) `BC` (g) `ABC`

(h) `D` ($\gamma = 3$) (i) `CD` (j) `BCD` (k) `ABCD`

Fig. 2. The 11 infeasible patterns as a result of combining the four basic infeasible patterns `Pattern A`, `Pattern B`, `Pattern C` and `Pattern D`. Nodes enclosed in a dashed rectangle represent alternative positions of the same node.

Theorem 3. *Let: $G = (V, E)$ be a DAG with a width function $w : V \cup E \to \mathbb{Q}^+$; $H, W \geq 0$; u and v be two independent nodes; and the sum of the widths of any two nodes be not larger than W. Then the RO inequality* (7) *is facet-defining for the DAG layering polytope.*

Proof. As background call $\mathcal{L}_G^H$ the derived graph with node set $V_{\mathcal{L}} \subseteq V \times \{1, \ldots, H\}$ with unique nodes corresponding to each layer that $q \in V$ could be placed and edge set $E_{\mathcal{L}}$ where $(p', q') \in E_{\mathcal{L}})$, $p', q' \in V_{\mathcal{L}}$ and $(p, q) \in E$, where p' (respectively q') corresponds to the placement of p (q) on some level $1, \ldots, H$.

The dimension of the DAG Layering polytope is $|V_{\mathcal{L}}|$ [5].

Let $\mathcal{L}_G^H = (V_{\mathcal{L}}, E_{\mathcal{L}})$. We construct $|V_{\mathcal{L}}|$ subsets of $V_{\mathcal{L}}$ which partially represent G, induce subgraphs of the LDAG $\mathcal{L}_G^H$ of width at most W, and whose incidence vectors satisfy (7) with equality and are linearly independent. We do this in five steps by choosing $|V_{\mathcal{L}}|$ subsets of $V_{\mathcal{L}}$ each corresponding to a different node $\lambda \in V_{\mathcal{L}}$.

Step 1. For each node λ_{ui} with $\varphi(u) \leq i \leq k$ we choose the set $\{\lambda_{ui}\}$, and for node λ_{vj} with $k + \gamma \leq j \leq \rho(v)$ we choose the set $\{\lambda_{vj}\}$.

Step 2. If node $z \in V$ is incident neither to u nor to v and is different from both u and v, then we can always choose the set $\{\lambda_{u\varphi(u)}, \lambda_{zi}\}$ for each $\varphi(z) \leq i \leq \rho(z)$.

Step 3. Let z be either u or v and let λ_{zi} be a node that has not been considered in a previous step. If $z = u$ then we choose the set $\{\lambda_{zi}, \lambda_{v\rho(v)}\}$; and if $z = v$ then we choose the set $\{\lambda_{zi}, \lambda_{u\varphi(u)}\}$.

Step 4. Now consider node λ_{zi} with $z \notin \{u, v\}$ and z incident to exactly one of nodes u and v. If there is an edge between z and u then we choose the set $\{\lambda_{zi}, \lambda_{v\rho(v)}\}$. Otherwise if there is an edge between z and v then we choose the set $\{\lambda_{zi}, \lambda_{u\varphi(u)}\}$.

Step 5. Finally let λ_{zi} is a node with z incident to both u and v. The two cases are a) both u and v are predecessors of z, or b) both u and v are successors of z. We need to choose sets ensuring that they will not induce a subgraph of the LDAG $\mathcal{L}_G^H$ with an edge that does not point downwards. In case a) since $\rho(v) > \rho(z)$, we can always choose the set $\{\lambda_{zi}, \lambda_{v\rho(v)}\}$; in case b) since $\varphi(u) < \varphi(z)$, we can always choose the set $\{\lambda_{zi}, \lambda_{u\varphi(u)}\}$.

For each node λ_{ui} with $\varphi(u) \leq i \leq k$ and each node λ_{vj} with $k+\gamma \leq j \leq \rho(v)$ at Step 1 we have chosen a single one-element subset of $V_{\mathcal{L}}$ containing it. Then at Steps 2-5 we have chosen a two-element subset for each other node of $V_{\mathcal{L}}$ with a second element one of the nodes already placed in a one-element subset at Step 1. Clearly the incidence vectors of the chosen sets are linearly independent and satisfy (7) with equality. Moreover, we have chosen them to represent G partially and induce subgraphs of $\mathcal{L}_G^H$ of width at most W, given that the sum of the widths of any two nodes does not exceed W.

The branch-and-cut algorithm for solving DAG layering which we propose consists of solving *WHS-Layering* in a branch-and-bound framework employing a cutting-plane algorithm at each node of the branch-and-bound tree for generating violated fixed-gap layering and RO inequalities. The cutting-plane algorithm consists of two parts: first, as a preprocessing step of the branch-and-cut algorithm we compute a list, `c-list`, of triples of two independent nodes u and v and the identifier of a pattern which is infeasible for u and v and any choice of k; then at each branch-and-bound node we simply scan `c-list` generating cuts based on violated fixed-gap layering and RO inequalities for a specific value of k. The computational time needed for building `c-list` is $O(|V|^2)$ and so is the size of `c-list` in the worst case.

The main part of the preprocessing step for building `c-list` is a boolean function `IsInfeasible` which combines implementations of both P_{ABOVE} and $P_{SAME-LAYER}$ discussed above. `IsInfeasible` takes as input an ordered pair of independent nodes (u, v) and a number $\gamma \in \{0, 1, 2, 3\}$ of a basic pattern. It returns **true** if it detects that basic pattern γ for nodes u and v leads to a layering of width greater than the upper bound W. Otherwise it returns **false**.

In order to give a fast answer `IsInfeasible` considers only the widths of the layers where nodes u and v are placed and their adjacent layers as discussed above. `IsInfeasible` simply scans all DAG edges and all DAG nodes. For each edge whose both endpoints are related to (or coincide with) either u

or v `IsInfeasible` adds the width of the dummy nodes along it to the widths of the considered layers. Similarly for each node which is either u or v or a node related to any of u and v `IsInfeasible` adds their contribution to the widths of the considered layers. `IsInfeasible` works as a heuristic, i.e. it may not detect that a particular pattern would lead to a too wide pattern and return **false**. This is due to the fact that both problems Same-layer DAG layering and Above DAG layering are $\mathcal{NP}$-complete.

an infeasible pattern for the node pair (u, v) and `IsInfeasible` returns **true**. Otherwise `IsInfeasible` returns **false**.

4 Experimental Procedures and Results

The layering algorithm we have developed, `ULair`[2], follows the general framework of an ABACUS-based branch-and-cut algorithm to which we have added the cutting-plane algorithm discussed in the previous section. For computational evaluation of `ULair` we used a machine with an 800 MHz Intel Pentium III CPU. The LP solver required by ABACUS was the one of CPLEX 7.0. In all tests we assume that all real and dummy nodes have unit width. All the running times we report are in centiseconds (the time unit adopted by ABACUS).

We applied `ULair` to the 1277 AT&T DAGs[3]. To each test DAG we first applied Gansner et al.'s layering algorithm to compute a layering with minimum number of dummy nodes and stored its width and height as W_{Gans} (taking the dummy nodes into account) and H_{Gans} respectively. Then for each test DAG we ran `ULair` several times, each time with a different pair of upper bounds on the width and the height. Each pair of upper bounds had the form $(f_W W_{Gans}, f_H H_{Gans})$ with

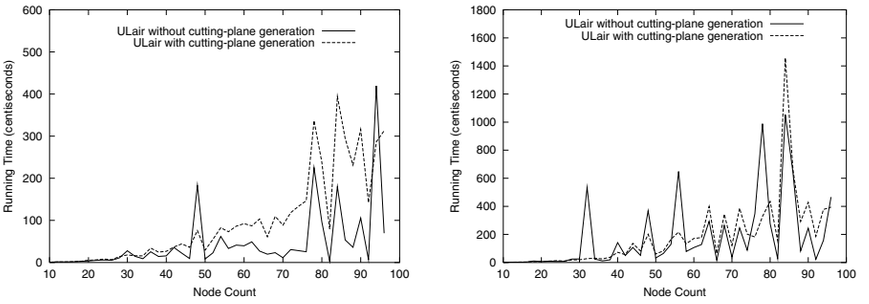
$$f_W, f_H \in \{0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4\}.$$

We observed that the running time of `ULair` is too long for some test DAGs and combinations of upper bounds, and we took 158 (12.37%) "hard-to-layer" DAGs out of the Test Set leaving 1119 DAGs. In the remainder of this section we present results of applying `ULair` to these 1119 AT&T DAGs.

We experimented with disabling the cutting-plane generation in `ULair`. By doing this, we observed that for the majority of DAGs it took the same computational time (or slightly better) to solve DAG layering, but there were a few instances which could not be optimized in reasonable time. The number of such "bad" instances increases when the width bound becomes tighter and the height bound becomes larger. We have collected complete results for $f_H = 1.0$ and $f_H = 1.1$ and partial results for $f_H = 1.2$ when cutting-plane generation is disabled. When $f_H \geq 1.2$ the running times for optimizing the "bad" instances becomes too long and much too resource-consuming, which shows that

[2] In what follows, mention of `ULair` refers to the present branch-and-cut system, as opposed to a previous version of the system that relied on CPLEX's mixed integer programming solver.

[3] We obtained the AT&T DAG set from http://www.graphdrawing.org.



(a) $W = 0.9W_{Gans}$, $H = 1.1H_{Gans}$ (b) $W = 0.7W_{Gans}$, $H = 1.2H_{Gans}$

Fig. 3. Average running times of `ULair` with and without cutting-plane generation.

the generation of cutting planes in such cases is important for completing the computation in reasonable time. Two typical cases of how `ULair` runs with and without generation of cutting planes are illustrated in Figure 3. It can be seen how the cutting-plane generation eliminates some peaks in the average running time, which are present when the cutting-plane generation is disabled. We excluded from consideration one test DAG (attg.56.5) in Figure 3(a) and one test DAG (attg.94.0) in Figure 3(b), because the running times for layering them without generation of cutting planes were too large and did not fit nicely in the plots.

The table in Figure 4 compares the running times of `ULair` with and without generation of cutting planes to the running times of the mixed integer programming solver (MIP) of CPLEX 7.0 for five DAGs which are hard to layer without generation of cutting planes. In order to build this table we constructed the

AT&T attg.	f_H/f_W	Running time (centisec.)			Cutting planes
		no cuts	MIP	cuts	
23.37	1.2 / 0.7	45	11	15	A:1, AB:1, BC:3, ABC:4
	1.2 / 0.8	330	9	17	A:6, AB:1, C:1, BC:4, ABC:9
29.2	1.2 / 0.7	235	174	29	AC:1, ABCD:17
	1.2 / 0.8	172	186	115	A:3, BC:5, ABC:15, BCD:7
32.1	1.2 / 0.7	16,919	85	24	AB:1, ABC:8, ABCD:7
	1.2 / 0.8	524	78	137	A:1, AB:9, AC:2, BC:2, ABC:1, ABCD:5
56.5	1.2 / 0.7	9,852	60,000+	945	ABCD:94
	1.2 / 0.8	60,000+	60,000+	945	ABCD:95
94.0	1.2 / 0.7	38,478	4,237	585	ABCD:157
	1.2 / 0.8	9,009	3,882	631	ABCD:203

Fig. 4. Some DAGs which are hard to layer without cutting-plane generation.

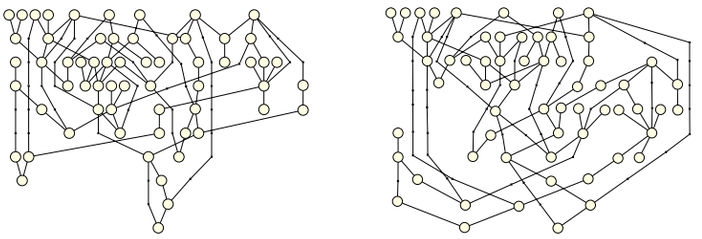
WHS-Layering ILP model for each of the five DAGs and ran MIP for solving it, accepting its default settings. The first column contains the number of the DAG; the second column, f_H/f_W, contains the values of f_H and f_W respectively. The next three columns represent the CPU time for solving DAG layering (eventually reporting that there is no feasible solution) in centiseconds. Running times over 60,000 centiseconds (10 min) are presented as 60,000+. Column *no cuts* contains the running times of `ULair` when the generation of cutting planes is disabled. The next column, *MIP*, displays the running times of CPLEX's MIP. Column *cuts* contains the running times of `ULair` when our cutting-plane strategy is applied. With a few insignificant exceptions the running times in column *cuts* are the best, which is a further evidence that our cutting-plane generation strategy helps for optimizing "hard" instances of DAG layering. The last column in Table 4, *Cutting planes*, contains for each infeasible pattern the number of generated cutting planes by `ULair` (when the cutting-plane generation is enabled). For example "AC:1, ABCD:17" means that one cutting plane corresponding to infeasible `Pattern AC` was generated and 17 cutting planes corresponding to infeasible `Pattern ABCD` were generated.

The average number of cutting planes tends to be bigger for bigger values of f_W and fixed f_H when there is no optimal solution (on average up to 80 cutting planes per DAG with node count between 70 and 100), while in the case that an optimal solution was found average numbers of cutting planes greater than or equal to 2 appear only for tight bounds on the width and large height bounds. This suggests to us that our infeasible solution detection and cutting plane generation algorithms are in need of further development. It might be fruitful to consider triples of nodes instead of pairs as we currently do. While this implies a larger list of candidates to consider, the running time of checking a triple involves only a constant number of pairwise infeasibility checks.

In Figure 5 we compare Gansner's layering on a 62-node, 79-edge DAG to `ULair`'s layering with dimension bounds equal to 0.7 and 1.0 times Gansner's width and height, respectively. Gansner's layering has 39 dummy nodes (depicted as tiny rectangles along the edges), while `ULair`'s layering has 7 more dummy nodes, caused by tightening the width bound. After the layering phase, we applied exact crossing minimization and at the end manually adjusted the final position of the nodes and the final shape of the edges. Gansner's layering resulted in 33 edge crossings, while `ULair` resulted into 30 edge crossings. Both drawings are drawn over the same drawing area. Because `ULair`'s layering has a smaller width, this allows larger distances between the nodes and as a result, `ULair`'s clearly has a better distribution of the nodes over the drawing area.

5 Conclusions

We have developed and implemented a branch-and-cut layering algorithm, `ULair`, which computes high quality layerings from aesthetic point of view. On the basis of our computational experience we conclude that `ULair` runs reasonably fast for DAGs having up to 100 nodes and it can be applied successfully when the quality



(a) (b)

Fig. 5. (a) Gansner's layering: height 10, width 23, 39 dummy nodes; (b) `ULair`'s layering: height 10, width 16, 46 dummy nodes

of a drawing is more important than fast computation. `ULair` can also provide a good initial layering for further manual adjustments and local improvements. With the current cutting-plane generation algorithm that considers only pairs of independent nodes, we have identified a number of DAGs where `ULair` performs faster than the mixed integer programming solver of CPLEX 7.0. By extending the cutting-plane generation algorithm, we believe that `ULair` can be made faster still. The performance of `ULair` can be further improved by employing tighter upper bounds on the sum of edge spans in bounded dimensions.

References

1. O. Bastert and C. Matuszewski. Layered drawings of digraphs. In M. Kaufman and D. Wagner, editors, *Drawing Graphs: Method and Models*, volume 2025 of *LNCS*, pages 87–120. Springer-Verlag, 2000.
2. J. Branke, S. Leppert, M. Middendorf, and P. Eades. Width-restriced layering of acyclic digraphs with consideration of dummy nodes. *Information Processing Letters*, 81(2):59–63, January 2002.
3. P. Eades and K. Sugiyama. How to draw a directed graph. *Journal of Information Processing*, 13(4):424–437, 1990.
4. P. Healy and N. S. Nikolov. How to layer a directed acyclic graph. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing: Proceedings of 9th International Symposium, GD 2001*, volume 2265 of *LNCS*, pages 16–30. Springer-Verlag, 2002.
5. P. Healy and N. S. Nikolov. Facets of the directed acyclic graph layering polytope. In Luděk Kučera, editor, *Graph Theoretical Concepts in Computer Science: WG2002*, LNCS. Springer-Verlag, (To appear.).
6. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transaction on Systems, Man, and Cybernetics*, 11(2):109–125, February 1981.