On the Parameterized Complexity of Layered Graph Drawing*

V. Dujmović[1], M. Fellows[2], M. Hallett[1], M. Kitching[1],
G. Liotta[3], C. McCartin[4], N. Nishimura[5], P. Ragde[5],
F. Rosamond[2], M. Suderman[1], S. Whitesides[1], and D.R. Wood[6]

[1] McGill University, Canada.
[2] University of Victoria, Canada.
[3] Università di Perugia, Italy.
[4] Victoria University of Wellington, New Zealand.
[5] University of Waterloo, Canada.
[6] The University of Sydney, Australia.

Abstract. We consider graph drawings in which vertices are assigned to layers and edges are drawn as straight line-segments between vertices on adjacent layers. We prove that graphs admitting crossing-free h-layer drawings (for fixed h) have bounded pathwidth. We then use a path decomposition as the basis for a linear-time algorithm to decide if a graph has a crossing-free h-layer drawing (for fixed h). This algorithm is extended to solve a large number of related problems, including allowing at most k crossings, or removing at most r edges to leave a crossing-free drawing (for fixed k or r). If the number of crossings or deleted edges is a non-fixed parameter then these problems are NP-complete. For each setting, we can also permit downward drawings of directed graphs and drawings in which edges may span multiple layers, in which case the total span or the maximum span of edges can be minimized. In contrast to the so-called Sugiyama method for layered graph drawing, our algorithms do not assume a preassignment of the vertices to layers.

1 Introduction

Layered graph drawing [28,5,26] is a popular paradigm for drawing graphs, and has applications in visualization [6], in DNA mapping [29], and in VLSI layout [21]. In a layered drawing of a graph, vertices are arranged in horizontal layers, and edges are routed as polygonal lines between distinct layers. For acyclic digraphs, it may be required that edges point downward.

* Research initiated at the International Workshop on Fixed Parameter Tractability in Graph Drawing, Bellairs Research Institute of McGill University, Holetown, Barbados, Feb. 9-16, 2001, organized by S. Whitesides. Contact author: P. Ragde, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 2P9, e-mail `plragde@uwaterloo.ca`. Research of Canada-based authors is supported by NSERC. Research of D. R. Wood supported by ARC and completed while visiting McGill University. Research of G. Liotta supported by CNR and MURST.

F. Meyer auf der Heide (Ed.): ESA 2001, LNCS 2161, pp. 488–499, 2001.

The quality of layered drawings is assessed in terms of criteria to be minimized, such as the number of edge crossings; the number of edges whose removal eliminates all crossings; the number of layers; the maximum *span* of an edge, i.e., the number of layers it crosses; the total span of the edges; and the maximum number of vertices in one layer. Unfortunately, the question of whether a graph G can be drawn in two layers with at most k crossings, where k is part of the input, is NP-complete [11,12], as is the question of whether r or fewer edges can be removed from G so that the remaining graph has a crossing-free drawing on two layers [27,10]. Both problems remain NP-complete when the permutation of vertices in one of the layers is given [11,10].

When, say, the maximum number of allowed crossings is small, an algorithm whose running time is exponential in this parameter but polynomial in the size of the graph may be useful. The theory of parameterized complexity (surveyed in [7]) addresses complexity issues of this nature, in which a problem is specified in terms of one or more parameters. A parameterized problem with input size n and parameter size k is *fixed parameter tractable*, or in the class *FPT*, if there is an algorithm to solve the problem in $f(k) \cdot n^{\alpha}$ time, for some function f and constant α.

In this paper we present fixed parameter tractability results for a variety of layered graph drawing problems. To our knowledge, these problems have not been previously studied from this point of view. In particular, we give a linear time algorithm to decide if a graph has a drawing in h layers (for fixed h) with no crossings, and if so, to produce such a drawing. We then modify this basic algorithm to handle many variations, including the *k-crossings* problem (for fixed k, can G be drawn with at most k crossings?), and the *r-planarization* problem (for fixed r, can G be drawn so that the deletion of at most r edges removes all crossings?). The exact solution of the r-planarization problem for $h \geq 3$ layers is stated as an open problem in a recent survey [23], even with vertices preassigned to layers. Our algorithm can be modified to handle acyclic directed graphs whose edges must be drawn pointing downward. We also consider drawings whose edges are allowed to span multiple layers. In this case, our algorithm can minimize the total span of the edges, or alternatively, minimize the maximum span of an edge.

We do not assume a preassignment of vertices to layers. In this regard, our approach is markedly different from the traditional method for producing layered drawings, commonly called the *Sugiyama* algorithm, which operates in three phases. In the first phase, the graph is *layered*; that is, the vertices are assigned to layers to meet some objective, such as to minimize the number of layers or the number of vertices within a layer [6, Chapter 9.1]. In the second phase the vertices within each layer are permuted to reduce crossings among edges, typically using a layer-by-layer sweep algorithm [26]. In this method, for successive pairs of neighbouring layers, the permutation of one layer is fixed, and a good permutation of the other layer is determined. The third phase of the method assigns coordinates to the vertices [2].

A disadvantage of the Sugiyama approach is that after the vertices have been assigned to layers in the first phase, these layer assignments are not changed

during the crossing minimization process in the second phase. In contrast, our algorithms do not assume a preassignment of vertices to layers. For example, in linear time we can determine whether a graph can be drawn in h layers with at most k edge crossings (for fixed h and k), taking into account all possible assignments of vertices to the layers.

The second phase of the Sugiyama algorithm has received much attention in the graph drawing literature. Notable are polynomial time algorithms to test if a layered graph admits a crossing-free drawing [15,17], and if so, to produce such a drawing with straight line-segments [16], even for edges which span multiple layers [9]. Integer linear programming formulations have been developed for crossing minimization in layered graphs [15,18,19], and for 2-layer planarization [22,24]. The special case of two layers is important for the layer-by-layer sweep approach. Junger and Mutzel [19] summarize the many heuristics for 2-layer crossing minimization. Our companion paper [8] addresses the 2-layer case.

The remainder of this paper is organized as follows. Section 2 gives definitions and discusses pathwidth, a key concept for our algorithms. The overall framework for our algorithms is presented in Section 3, where we consider the problem of producing layered drawings with no crossings. The r-planarization problem, the k-crossings problem, and further variants are considered in Section 4. Section 5 concludes with some open problems. Many proofs are omitted or sketched due to space limitations.

2 Preliminaries

We denote the vertex and edge sets of a graph G by $V(G)$ and $E(G)$, respectively; we use n to denote $|V(G)|$. Unless stated otherwise, the graphs considered are simple and without self-loops. For a subset $S \subseteq V(G)$, we use $G[S]$ to denote the subgraph of G induced by the vertices in S. In order to structure our dynamic programming algorithms, we make use of the well-known graph-theoretic concepts of *path decomposition* and *pathwidth*.

A *path decomposition* P of a graph G is a sequence $P_1, \ldots, P_p$ of subsets of $V(G)$ that satisfies the following three properties: (1) for every $u \in V(G)$, there is an i such that $u \in P_i$; (2) for every edge $uv \in E(G)$, there is an i such that both $u, v \in P_i$; and (3) for all $1 \leq i < j < k \leq p$, $P_i \cap P_k \subseteq P_j$. The *width* of a path decomposition is defined to be $\max\{|P_i| - 1 : 1 \leq i \leq p\}$. The *pathwidth* of a graph G is the minimum width w of a path decomposition of G. Each P_i is called a *bag* of P. It is easily seen that the set of vertices in a bag is a separator of the graph G. For fixed w, path decompositions of graphs of pathwidth w can be found in linear time [4].

A path decomposition $P = P_1, \ldots, P_p$, of a graph G of pathwidth w is a *normalized path decomposition* if (1) $|P_i| = w + 1$ for i odd; (2) $|P_i| = w$ for i even; and (3) $P_{i-1} \cap P_{i+1} = P_i$ for even i. Given a path decomposition, a normalized path decomposition of the same width (and $\Theta(n)$ bags) can be found in linear time [13].

A *proper h-layer drawing* of a (directed or undirected) graph G consists of a partition of the vertices $V(G)$ into h *layers* $L_1, L_2, \ldots L_h$ such that for each edge $uv \in E(G)$, $u \in L_i$ and $v \in L_j$ implies $|i - j| = 1$; vertices in layer L_i, $1 \leq i \leq h$, are positioned at distinct points in the plane with a Y-coordinate of i, and edges are represented by straight line-segments.

Edge crossings in layered drawings do not depend on the actual assignment of X-coordinates to the vertices, and we shall not be concerned with the determination of such assignments. For our purposes, a layered drawing can be represented by the partition of the vertices into layers and linear orderings of the vertices within each layer. In a layered drawing, we say a vertex u is to the *left* of a vertex v and v is to the *right* of u, if u and v are in the same layer and $u < v$ in the corresponding linear ordering.

An (a, b)-*stretched h-layer drawing* of a graph G is a proper h-layer drawing of a graph G' obtained from G by replacing each edge of G by a path of length at most $a + 1$ such that the total number of "dummy" vertices is at most b, and all edges have monotonically increasing or decreasing Y-coordinates. Of course, proper h-layer drawings are $(0, 0)$-stretched. A graph is said to be an (a, b)-*stretchable h-layer graph* if it admits an (a, b)-*stretched* h'-*layer drawing* for some $h' \leq h$.

A layered drawing with at most k crossings is said to be k-*crossing*, where a crossing is counted every time a pair of edges cross. A 0-crossing h-layer drawing is called an h-*layer plane drawing*. A graph is ((a, b)-*stretchable*) h-*layer planar* if it admits an ((a, b)-stretched) plane h'-layer drawing for some $h' \leq h$. A layered drawing in which r edges can be deleted to remove all crossings is said to be r-*planarizable*, and a graph which admits an ((a, b)-stretched) r-planarizable h-layer drawing is said to be an ((a, b)-*stretchable*) r-*planarizable h-layer graph*.

For an acyclic digraph G, an ((a, b)-stretched) h-layer drawing if G is called *downward* if for each edge $(u, v) \in E(G)$, $u \in L_i$ and $v \in L_j$ implies $i > j$.

3 Proper h-Layer Plane Drawings

In this section we present an algorithm for recognizing proper h-layer planar graphs. Our algorithm, which performs dynamic programming on a path decomposition, relies on the fact that an h-layer planar graph has bounded pathwidth.

Lemma 1. *If G is an h-layer planar graph, then G has pathwidth at most h.*

Proof Sketch. First consider a proper h-layer planar graph G. Given an h'-layer plane drawing of G for some $h' \leq h$, we form a normalized path decomposition of G in which each bag of size h' contains exactly one vertex from each layer. Let S be the set of leftmost vertices on each layer, where $s_i \in S$ is the vertex on layer i. Initialize the path decomposition to consist of the single bag S, and repeat the following step until S consists of the rightmost vertices on each layer: find a vertex $v \in S$ such that all neighbours of v are either in S or to the left of vertices in S. Let u be the vertex immediately to the right of v. Append the bag $S \cup \{u\}$, followed by $S \cup \{u\} \setminus \{v\}$ to the path decomposition, and set

the current bag $S \leftarrow S \cup \{u\} \setminus \{v\}$. It is not hard to show that this yields the required decomposition. To handle stretchable h-layer planar graphs, insert dummy nodes, apply the above algorithm, replace dummy vertices on an edge uv by u, and remove all but one of a sequence of duplicate bags. □

As stated in Section 2, we can obtain a normalized width-h path decomposition of any graph for which one exists in time $O(n)$ (for fixed h) [4,13]. Applying this algorithm to an h-layer planar graph will in general not result in a "nice" decomposition like that in Lemma 1 (where bags of size h contain exactly one vertex from each layer), but we can use the fact that each bag is a separator in order to obtain a dynamic programming algorithm. In the remainder of this section we prove the following result.

Theorem 1. *There is an $f(h) \cdot n$ time algorithm that decides whether a given graph G on n vertices is proper h-layer planar, and if so, produces a drawing.*

By applying the algorithm of Bodlaender [4], we can test if G has a path decomposition of width at most h. If G does not have such a path decomposition, by Lemma 1, G is not h-layer planar. Otherwise, let $P = P_1, \ldots, P_p$ be the normalized path decomposition of G given by the algorithm of Gupta *et al.* [13]. Let $w \leq h$ be the width of this path decomposition. (Our algorithm, in fact, works on any path decomposition of fixed width w, and in subsequent sections we present modifications of this procedure where the path decomposition has width $w > h$.)

Our dynamic programming is structured on the path decomposition, where for each bag P_i in turn, we determine all possible assignments of the vertices of P_i to layers and, for each assignment, all possible orderings of the vertices on a particular layer such that $G[\cup_{j \leq i} P_j]$ is proper h-layer planar.

The key to the complexity of the algorithm is the fact that the number of proper h-layer plane drawings of $G[\cup_{j \leq i} P_j]$ can be bounded as a function only of the parameters h and w. In order to ensure this bound, we need a way of representing not only edges between vertices in P_i but also edges with one or more endpoints in $\bigcup_{j<i} P_j \setminus P_i$. Since representing each edge will require a prohibitively large amount of information, we instead label horizontal "intervals" between vertices in P_i on the same layer.

More formally, for a subset S of vertices in a proper h-layer drawing of a graph, we add a set S' of $2h$ boundary vertices to the extreme left and extreme right of each layer, and use the term *interval*, denoted by $\mathsf{int}(u, v)$, to refer to the horizontal line-segment between any pair of consecutive vertices u and v in $S \cup S'$ on the same layer. The set of all intervals for $S \cup S'$ is denoted by $\mathsf{int}(S)$. It is not difficult to see that the number of intervals is linear in $h + |S|$.

A crucial observation is that, for the purposes of deciding where vertices and edges can be inserted into a partial drawing of G, the edges with endpoints in $\mathsf{int}(u, v)$ can be represented by the sets of vertices in the layers above and below that can "see" a particular "subinterval" of $\mathsf{int}(u, v)$.

In a proper h-layer plane drawing, let u and v be a pair of vertices appearing at layer ℓ, $1 < \ell < h$, (where other vertices may appear between u and v on

layer ℓ), T be any subsequence of the vertices at layer $\ell+1$, and B be any subsequence of the vertices at layer $\ell-1$, where the vertices in a particular layer are ordered from left to right. A point x in $\mathsf{int}(u,v)$ is *visible* to a subsequence T' of T and a subsequence B' of B if it is possible to draw line-segments between x and every point in T' and B' without creating any crossings. A *subinterval* I of $\mathsf{int}(u,v)$ is a sequence of points in $\mathsf{int}(u,v)$ visible to the same subsequences T' of T and B' of B. We say I is *visible* to these subsequences, and we label I by the pair (T',B'), its *visibility label with respect to* T *and* B, or more succinctly, its *visibility label* (see Fig. 1). By extension, a sequence of visibility labels for subintervals of $\mathsf{int}(u,v)$, with each consecutive sequence of identical labels replaced by a single copy, forms a *visibility label of* $\mathsf{int}(u,v)$, and for any subset S of vertices in a layered graph, the visibility labels of all subintervals of intervals in $\mathsf{int}(S)$ forms a *visibility label of* $\mathsf{int}(S)$.

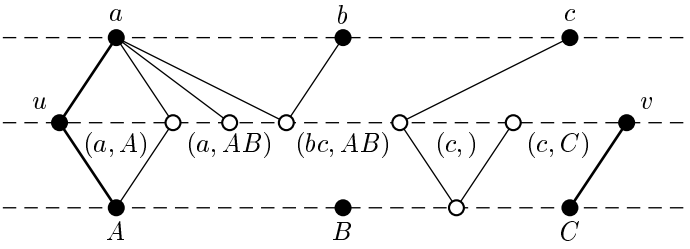

Fig. 1. An example of visibility labelling. Solid vertices are in the set S, and edges with both ends in S are bold. The interval $\mathsf{int}(u,v)$ is divided into five subintervals of differing visibility, each with visibility labels shown. The label of each subinterval indicates how a vertex added to that subinterval could be connected to vertices in S.

This definition can be extended to label intervals on layers 1 (where B is empty) and h (where T is empty). The following lemmas are direct consequences of the definition of visibility.

Lemma 2. *Given vertices x and y on consecutive layers,*

1. *if there is an edge xy then no vertex to the left (resp. right) of x is visible to any vertex to the right (resp. left) of y, and*
2. *if x is not visible to y, then there must exist an edge $x'y'$ such that either x' is to the right of x and and y' is to the left of y, or x' is to the left of x and y' is to the right of y.* □

Lemma 3. *For each visibility label (T',B'), T' is a consecutive subsequence of T and B' is a consecutive subsequence of B.* □

In a proper h-layer plane drawing, the visibility labels satisfy the following lexicographic ordering. For a sequence $X = (x_1, x_2, \ldots, x_{|X|})$ and two consecutive subsequences y_1 and y_2 of members of X, where y_1 contains x_a through x_b and y_2 contains $x_{a'}$ through $x_{b'}$, $y_1 \prec y_2$ if one of the following three cases

holds: $a < a'$, $a = a'$ and $b < b'$, or exactly one of y_1 and y_2 is empty. Similarly, $y_1 \preceq y_2$ if either $y_1 \prec y_2$ or $a = a'$ and $b = b'$.

Based on the above definition and Lemma 2, it is straightforward to prove the following lemma.

Lemma 4. *For any vertices u and v in a proper h-layer plane drawing, the number j of visibility labels of subintervals of $\mathsf{int}(u, v)$ with respect to T and B is $O(|T||B|)$. Moreover, the labels $(T_1, B_1), (T_2, B_2), \ldots, (T_j, B_j)$ satisfy the following conditions:*

1. *for any $1 \leq i < j$, $T_i \preceq T_{i+1}$, $B_i \preceq B_{i+1}$;*
2. *for any $1 \leq i < j$, either $T_i \prec T_{i+1}$, $B_i \prec B_{i+1}$, or both hold; and*
3. *for any w to the right of v, if (T_j, B_j) is the last label for $\mathsf{int}(u, v)$ and (T'_1, B'_1) is the first label for $\mathsf{int}(v, w)$, then $T_j \preceq T'_1$ and $B_j \preceq B'_1$.* □

We define an *ordered layer assignment* A of P_i to be an assignment of the vertices in P_i to layers such that there is an ordering imposed on the vertices of P_i in each layer. The drawing of $G[P_i]$ obtained by placing vertices according to A and including all edges in $E(G[P_i])$ is called the *drawing of $G[P_i]$ induced by A*. As a consequence of Lemma 3, the number of ordered layer assignments and visibility labels for these assignments are functions of h and w.

In our dynamic programming table, the entry $\mathsf{TABLE}\langle i, A, L\rangle$ indicates whether or not it is possible to obtain a proper h-layer plane drawing of $G[\cup_{j \leq i} P_j]$ with ordered layer assignment A of P_i and visibility label L of $\mathsf{int}(P_i)$. The notation $\mathsf{TABLE}\langle i, A, *\rangle$ is used to denote entries for i, A, and any L. The order of evaluation is by increasing i. For a bag P_i, ordered layer assignment A of P_i, and visibility label L, a necessary condition for $\mathsf{TABLE}\langle i, A, *\rangle$ to be YES is that no edges in $G[P_i]$ violate the conditions of a proper h-layer plane drawing or of part 1 of Lemma 2. In this case, we call P_i, A, and L *consistent*.

For $i = 1$, the base case of our dynamic programming recurrence, we consider all possible ordered layer assignments of the vertices in P_1. If P_1, A, and L are inconsistent, we set $\mathsf{TABLE}\langle 1, A, L\rangle$ to NO. In general, a labeling L represents visibility due to edges not only in $G[P_i]$ but also with endpoints in $G[\cup_{j<i} P_j]$; since for $i = 1$ the latter graph is empty, we impose the additional constraint that L corresponds to exactly the edges in $G[P_i]$. That is, for a given P_1 and A, there is exactly one labeling L such that P_1, A, and L are consistent and such that part 2 of Lemma 2 is satisfied for edges in $G[P_1]$; for this L, we set $\mathsf{TABLE}\langle 1, A, L\rangle$ to YES, and set $\mathsf{TABLE}\langle 1, A, L'\rangle$ to NO for each $L' \neq L$.

To define the general recurrence, we need to consider two different cases, namely for $|P_i| = w + 1$ (i odd) and $|P_i| = w$ (i even). In the first case, P_{i-1} is missing exactly one vertex that is in P_i, and the algorithm checks the single table entry defined by removing that vertex. In the second case, P_{i-1} has exactly one vertex x that is not in P_i, and we consider all possible placements of x. Both of these tasks can be done in constant time (for fixed h and w). In each case, we first check if P_i, A, and L are consistent, and if not, enter NO. Details of the remaining steps are given below.

To compute $\mathsf{TABLE}\langle i, A, L\rangle$ for odd i, we note that $|P_i| = w+1$ and $|P_{i-1}| = w$. Let x be the single vertex in $P_i \setminus P_{i-1}$. The algorithm computes A' and

L', the ordered layer assignment and visibility label of $\mathsf{int}(P_{i-1})$ that would result by removing x from a drawing corresponding to A and L, and then sets $\mathsf{TABLE}\langle i, A, L\rangle \leftarrow \mathsf{TABLE}\langle i-1, A', L'\rangle$.

To compute A', we simply remove x from A. The computation of L' is only slightly more complicated. Observe that for ℓ, the layer containing x, the only visibility labels that will be altered are those in layers $\ell+1$, ℓ, and $\ell-1$. Part 3 of Lemma 4 shows that for t and t', the vertices appearing immediately before and after x in layer ℓ in A, there is an order imposed on the last label of $\mathsf{int}(t, x)$ and the first label of $\mathsf{int}(x, t')$. Since parts 1 and 2 establish orderings within the intervals, the sequence formed by concatenating these labels and removing x forms a labeling of $\mathsf{int}(t, t')$ that satisfies Lemma 4, parts 1 and 2.

Next, for each visibility label at layer $\ell+1$, the algorithm removes x from all B_j's, and then removes any adjacent duplicate pairs of subinterval labels that result. Explicit adjustments will only occur when x appears as an endpoint of a subinterval. The T_j's at layer $\ell-1$ are adjusted in the same fashion.

To compute $\mathsf{TABLE}\langle i, A, L\rangle$ for even i, we note that $|P_i| = w$ and $|P_{i-1}| = w+1$. Let x be the single vertex in $P_{i-1} \setminus P_i$. The algorithm computes the set $\mathcal{S}$ of pairs (A', L') representing the ordered layer assignment and visibility label of $\mathsf{int}(P_{i-1})$ that would result from adding x to a drawing corresponding to A and L in all possible subintervals I, and then sets $\mathsf{TABLE}\langle i, A, L\rangle$ to YES if and only if $\mathsf{TABLE}\langle i-1, A', L'\rangle$ is YES for some $(A', L') \in \mathcal{S}$.

To add the new vertex x to a particular subinterval I, we first add x to A in the appropriate position, along with all its edges. If the neighbours of x in P_i do not form a subset of the visibility label of I, $\mathsf{TABLE}\langle i, A, L\rangle$ is set to NO.

If instead the neighbours form a subset of the visibility label of I, to form L', define $m_t = \min\{i' | t_{i'} x \text{ is an edge}\}$, $M_t = \max\{i' | t_{i'} x \text{ is an edge}\}$, $m_b = \min\{i' | x, b_{i'} \text{ is an edge}\}$, and $M_b = \max\{i' | x b_{i'} \text{ is an edge}\}$, where $t_1, \ldots t_{|T|}$ is the sequence of vertices of P_{i-1} at layer $\ell+1$ and $b_1, \ldots, b_{|B|}$ is the sequence of vertices of P_{i-1} at layer $\ell-1$.

Split I into two identical subintervals, one on each side of x. Then, for all visibility labels of subintervals to the left of x, remove t_i for all $i > m_t$, and remove b_j for all $j > m_b$. Similarly, for all visibility labels to the right of x, remove t_i for all $i < M_t$ and remove b_j for all $j < M_b$.

A straightforward induction on i proves the following statement and therefore the correctness of the algorithm: the entry $\mathsf{TABLE}\langle i, A, L\rangle$ is YES if and only if it is possible to obtain a proper h-layer plane drawing of $G[\cup_{j \le i} P_j]$ with ordered layer assignment A of P_i and visibility label L of $\mathsf{int}(P_i)$.

Obviously, G has a proper h-layer plane drawing if and only if some entry $\mathsf{TABLE}\langle p, *, *\rangle$ is YES. To determine the complexity of the algorithm, we recall that the number of bags is in $O(n)$ and thus the number of table entries is $g(h, w) \cdot n$ for some function g. Each table entry can be computed in time $e(h, w)$ for some function e, and $w = h$. Thus the total running time is $f(h) \cdot n$. (The precise nature of the functions of the parameters is discussed in Section 5.) An actual drawing can be obtained by tracing back through the table in standard dynamic programming fashion. This concludes the proof of Theorem 1.

4 Edge Removals, Crossings, and Other Variants

Consider a proper r-planarizable h-layer graph. The h-layer planar subgraph obtained by removing the appropriate r edges has a path decomposition of width at most h by Lemma 1. By placing both endpoints of each of the r removed edges in each bag of the path decomposition, we form a path decomposition of the original graph of width at most $h+2r$. In a proper k-crossing h-layer drawing we can delete at most k edges to remove all crossings. By the same argument as above we obtain the following lemma.

Lemma 5. *If G is a k-crossing h-layer graph (r-planarizable h-layer graph), then G has pathwidth at most $h + 2k$ ($h + 2r$).* □

Theorem 2. *There is a $f(h, r) \cdot n$ time algorithm to determine whether a given graph on n vertices is a proper r-planarizable h-layer graph, and if so, produces a drawing.*

Proof Sketch. The main change to the dynamic programming algorithm described in Section 3 is an additional dimension to the table representing an edge removal budget of size at most r. The entry $\mathsf{TABLE}\langle i, A, L, c\rangle$ indicates whether or not it is possible to obtain a proper h-layer plane drawing of $G[\cup_{j\leq i} P_j]$ with ordered layer assignment A of P_i and visibility label L of $\mathsf{int}(P_i)$ by removing at most $r - c$ edges. □

Theorem 3. *There is a $f(h, k) \cdot n$ time algorithm to determine whether a given graph on n vertices is a proper k-crossing h-layer graph, and if so, produces a drawing.*

Proof Sketch. We proceed in a fashion similar to Theorem 2. We modify the graph representation of the previous section to include two different types of edges, *black* edges not involved in crossings, and up to $2k$ *red* edges which may be involved in crossings. Since we can obtain more than k crossings using $2k$ edges, we also need to keep a budget of crossings. In our algorithm, the entry $\mathsf{TABLE}\langle i, A, L, R, c\rangle$ indicates whether or not it is possible to obtain a proper h-layer drawing of $G[\cup_{j\leq i} P_j]$ with ordered layer assignment A of P_i, visibility label L of $\mathsf{int}(P_i)$ such that a subset of edges map to red edges in the set R, the only crossings in the graph involve red edges, and the total number of crossings is $k - c$. □

We now describe how the algorithms above can be modified to take into account the directions of edges and stretch. For a downward drawing of a digraph, the directions of edges can easily be verified in the consistency check. Suppose a graph G is stretchable h-layer planar. By Lemma 1, the graph G' (defined in Section 2) has pathwidth at most h. Since graphs of bounded pathwidth are closed under edge contraction, G also has bounded pathwidth. To handle stretch in the dynamic programming, we consider placements not only of new vertices but also of dummy vertices. The total number of possibilities to consider at any step of the dynamic programming is still a function only of h and w (the bag size). The bound on the total number of dummy vertices, if used, need not be a parameter, though the running time is multiplied by this bound.

Theorem 4. *For each of the following classes of graphs, there are FPT algorithms that decide whether or not an input graph belongs to the class, and if so, produces an appropriate drawing, with the parameters as listed:*

1. *h-layer planar, k-crossing or r-planarizable graphs, (h, ∞)-stretched, (a, ∞)-stretched, or (a, b)-stretched, with parameters h, k, and r;*
2. *radial graphs (drawn on h concentric circles), with k crossings or r edges removed, (h, ∞)-stretched, (a, ∞)-stretched, or (a, b)-stretched, with parameters h, k, and r;*
3. *digraph versions of the above classes such that the drawings are downward;*
4. *multigraph versions of the above classes, where edges can be drawn as curves; and*
5. *versions of any of the above classes of graphs where some vertices have been preassigned to layers and some vertices must respect a given partial order.*

Proof Sketch. These classes have bounded pathwidth, and the basic dynamic programming scheme can be modified to deal with them. □

5 Conclusions and Open Problems

Mutzel [23] writes "The ultimate goal is to solve these [layered graph drawing] problems not levelwise but in one step". In this paper we employ bounded pathwidth techniques to solve many layered graph drawing problems in one step *and* without the preassignment of vertices to layers.

A straightforward estimation of the constants involved in our linear-time algorithms shows that if $s = h + 2k + 2r$ is the sum of the parameters, then the dynamic programming can be completed in time $s^{cs}n$ for some small c. However, the cost of finding the path decomposition on which to perform the dynamic programming dominates this; it is $2^{32s^3}n$. Hence our algorithms should be considered a theoretical start to finding more practical FPT results. In a companion paper [8] we use other FPT techniques to shed light on the case of 2-layer drawings, obtaining better constants.

Improving the general result might involve finding more efficient ways to compute the path decomposition (perhaps with a modest increase in the width) for the classes of graphs under consideration.

Another approach to laying out proper h-layer planar graphs is to use the observation that such graphs are k-outerplanar, where $k = \lceil \frac{1}{2}(h+1) \rceil$. One could use an algorithm to find such an embedding in $O(k^3n^2)$ time [3], and then apply Baker's approach of dynamic programming on k-outerplanar graphs [1]. However, this approach depends heavily on planarity, and so does not appear to be amenable to allowing crossings or edge deletions.

If we relax the requirement of using h layers, recent work gives an $f(k) \cdot n^2$ algorithm for recognizing graphs that can be embedded in the plane with at most k crossings [14]. A very similar approach would work for deleting r edges to leave a graph planar. Unfortunately, the approach relies on deep structure theorems from the Robertson-Seymour graph minors project, and so is even

more impractical. Nevertheless, since the maximum planar subgraph problem is of considerable interest to the graph drawing community, this should provide additional incentive to consider FPT approaches.

If we relax the requirement of planarity, we ask only if r edges can be deleted from a DAG to leave its height at most h. This is easily solved in time $O(((h+1)(r+1))^r + n)$; find a longest directed path (which cannot have length more than $(h+1)(r+1)$), and recursively search on each of the graphs formed by deleting one edge from this path to see if it requires only $r-1$ deletions.

There is a linear-time test for h-layer planar graphs when the assignment of vertices to layers is specified [17]. Is recognizing h-layer planar graphs without such a specification NP-complete, if h is not fixed?

Acknowledgements. We would like to thank the Bellairs Research Institute of McGill University for hosting the International Workshop on Fixed Parameter Tractability in Graph Drawing, which made this collaboration possible.

References

1. B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.
2. C. Buchheim, M. Jünger, and S. Leipert. A fast layout algorithm for k-level graphs. In J. Marks, editor, *Proc. Graph Drawing: 8th International Symposium (GD'00)*, volume 1984 of *Lecture Notes in Comput. Sci.*, pages 229–240. Springer, 2001.
3. D. Bienstock and C. L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmics*, 5:93–109, 1990.
4. H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.
5. M. J. Carpano. Automatic display of hierarchized graphs for computer aided decision analysis. *IEEE Trans. Syst. Man Cybern.*, SMC-10(11):705– 715, 1980.
6. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.
7. R. G. Downey and M. R. Fellows. *Parameterized complexity.* Springer, 1999.
8. V. Dujmović, M. Fellows, M. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. Rosemand, M. Suderman, S. Whitesides, and D. R. Wood. A fixed-parameter approach to two-layer planarization. Submitted.
9. P. Eades, Q. W. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In North [25], pages 113–128.
10. P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoret. Comput. Sci.*, 131(2):361–374, 1994.
11. P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994.
12. M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM J. Algebraic Discrete Methods*, 4(3):312–316, 1983.
13. A. Gupta, N. Nishimura, A. Proskurowski, and P. Ragde. Embeddings of k-connected graphs of pathwidth k. In M. M. Halldorsson, editor, *Proc. 7th Scandinavian Workshop on Algorithm Theory (SWAT'00)*, volume 1851 of *Lecture Notes in Comput. Sci.*, pages 111–124. Springer, 2000.

14. M. Grohe. Computing crossing numbers in quadratic time. In *Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC'01)*, 2001. To appear.
15. P. Healy and A. Kuusik. The vertex-exchange graph and its use in multi-level graph layout. In Kratochvil [20], pages 205–216.
16. M. Jünger and S. Leipert. Level planar embedding in linear time. In Kratochvil [20], pages 72–81.
17. M. Jünger, S. Leipert, and P. Mutzel. Level planarity testing in linear time. In S. Whitesides, editor, *Proc. Graph Drawing: 6th International Symposium (GD'98)*, volume 1547 of *Lecture Notes in Comput. Sci.*, pages 224–237. Springer, 1998.
18. M. Jünger, E. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing minimization problem. In G. Di Battista, editor, *Proc. Graph Drawing: 5th International Symposium (GD'97)*, volume 1353 of *Lecture Notes in Comput. Sci.*, pages 13–24. Springer, 1998.
19. M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: performance of exact and heuristic algorithms. *J. Graph Algorithms Appl.*, 1(1):1–25, 1997.
20. J. Kratochvil, editor. *Proc. Graph Drawing: 7th International Symposium (GD'99)*, volume 1731 of *Lecture Notes in Comput. Sci.* Springer, 1999.
21. T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley, 1990.
22. P. Mutzel. An alternative method to crossing minimization on hierarchical graphs. In North [25], pages 318–333. To appear in *SIAM Journal on Optimization.*
23. P. Mutzel. Optimization in leveled graphs. In P. M. Pardalos and C. A. Floudas, editors, *Encyclopedia of Optimization.* Kluwer, 2001. To appear.
24. P. Mutzel and R. Weiskircher. Two-layer planarization in graph drawing. In K. Y. Chwa and O. H. Ibarra, editors, *Proc. 9th International Symposium on Algorithms and Computation (ISAAC'98)*, volume 1533 of *Lecture Notes in Comput. Sci.*, pages 69–78. Springer, 1998.
25. S. North, editor. *Proc. Graph Drawing: Symposium on Graph Drawing (GD'96)*, volume 1190 of *Lecture Notes in Comput. Sci.* Springer, 1997.
26. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Systems Man Cybernet.*, 11(2):109–125, 1981.
27. N. Tomii, Y. Kambayashi, and S. Yajima. On planarization algorithms of 2-level graphs. *Papers of tech. group on elect. comp., IECEJ*, EC77-38:1–12, 1977.
28. J. N. Warfield. Crossing theory and hierarchy mapping. *IEEE Trans. Systems Man Cybernet.*, SMC-7(7):505–523, 1977.
29. M. S. Waterman and J. R. Griggs. Interval graphs and maps of DNA. *Bull. Math. Biol.*, 48(2):189–195, 1986.