On the Parameterized Complexity of Layered Graph Drawing*

V. Dujmović** M. Fellows*** M. Kitching**
G. Liotta† C. McCartin‡ N. Nishimura§ P. Ragde§
F. Rosamond*** S. Whitesides** D. R. Wood¶

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 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 either 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 [War77,Car80,STT81] is a popular paradigm for drawing graphs, and has applications in visualization [DETT99], DNA mapping [WG86], and VLSI layout [Len90]. 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.

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 number of layers an edge may cross, the total number of layers crossed by 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 [EW94b,GJ83], as is the question of whether r or fewer edges can be removed from G so that the remaining graph

* 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, School 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 Quebec-based authors is also supported by a grant from FCAR. Research of D. R. Wood completed while visiting McGill University. Research of G. Liotta supported by CNR and MURST.

** School of Computer Science, McGill University, Canada.
*** School of Electrical Engineering and Computer Science, The University of Newcastle, Newcastle, Australia.
† Dipartimento di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia, Italy.
‡ Institute of Information Science and Technology, Massey University, New Zealand.
§ School of Computer Science, University of Waterloo, Canada.
¶ School of Computer Science and Department of Mathematics and Statistics, McGill University, Montreal, Canada.

has a crossing-free drawing on two layers [TKY77,EW94a]. Both problems remain NP-complete when the permutation of vertices in one of the layers is given [EW94b,EW94a].

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 [DF99] 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 observe that recognizing such graphs is NP-complete when h is not fixed [HR92]). 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 (for vertices preassigned to layers) is stated as an open problem in a recent survey [Mut01b]. 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.

Our approach is markedly different from the traditional three-phase method for producing layered drawings, commonly called the *Sugiyama* algorithm. 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 [DETT99, 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 [STT81]. In this method, for successive pairs of neighboring 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 [BJL01].

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 [HK99,JLM98], and if so, to produce such a drawing with straight line-segments [JL99] (an extension [EFL97] handles edges that span multiple layers). Integer linear programming formulations have been developed for crossing minimization in layered graphs [HK99,JLMO98,JM97] and for 2-layer planariza-

tion [Mut01a,MW98]. The special case of two layers is important for the layer-by-layer sweep approach. Junger and Mutzel [JM97] summarize the many heuristics for 2-layer crossing minimization. The *median heuristic* [EW94b] is a 3-approximation algorithm for 2-layer crossing minimization with one layer fixed. A heuristic that does not follow the layer-by-layer sweep paradigm is the "global sifting" algorithm [MSM99,GSBM01]. Our companion paper [DFH+02,DFH+] 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.

2 Preliminaries

2.1 Graphs and Pathwidth

To describe the graphs considered in this paper, we make use of standard graph-theoretic notation [BM76]. 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 on the vertices in S. In order to structure our dynamic programming algorithms, we make use of the graph-theoretic concepts of *path decomposition* and *pathwidth*.

Definition 1. 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 a t such that $u \in P_t$;
2. for every edge $(u, v) \in E(G)$, there is a t such that both $u, v \in P_t$; and
3. for every $u \in V(G)$, $\{t \mid u \in P_t\} = \{i \mid j \leq i \leq k\}$ for some $1 \leq j \leq k \leq p$.

The *width* of a path decomposition is defined to be $\max\{|P_t| - 1 : 1 \leq t \leq p\}$. The *pathwidth* of a graph G is the minimum width w of any path decomposition of G. Each P_t is called a *bag* of P. It is easily seen that the set of vertices in a bag P_t is a separator of the graph G for $1 < t < p$. Of particular interest are graphs of constant pathwidth, for they have constant-size separators. For fixed w, path decompositions of graphs of pathwidth w can be found in linear time [Bod96,BK96].

Definition 2. A path decomposition $P = P_1, \ldots, P_p$ of a graph G of pathwidth w is a *normalized path decomposition* if

1. $|P_t| = w + 1$ for t odd;
2. $|P_t| = w$ for t even; and
3. $P_{t-1} \cap P_{t+1} = P_t$ for even t.

Given a path decomposition, a normalized path decomposition of the same width (and $\Theta(n)$ bags) can be found in linear time [GNPR00].

2.2 Layered Graph Drawing

An *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 $(u, v) \in E(G)$, $u \in L_i$ and $v \in L_j$ implies $i \neq j$; each vertex on layer L_j, $1 \leq j \leq h$, is positioned at a distinct point in the plane with a Y-coordinate of i and each edge is represented by a straight line-segment. If we further stipulate that for each edge $(u, v) \in E(G)$, $u \in L_i$ and $v \in L_j$ implies $|i - j| = 1$, G is a *proper h-layer drawing*.

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 with at most a internal vertices such that the total number of interior "dummy" vertices on the paths is at most b, and path vertices 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 the number of crossings is the number of pairs of edges that 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 that admits an ((a, b)-stretched) r-planarizable h-layer drawing is said to be an ((a, b)-*stretchable*) r*-planarizable h-layer graph*. We further distinguish between proper and non-proper versions of these graphs, depending on whether or not the graph admits a proper h-layer drawing of the specified type.

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

When h is implicit and constant, we use the shorthand *layered drawing* to replace "h-layer drawing" in the various terms given above.

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 (algorithms for this purpose were presented in other work [BJL01,GKNV93,San96]). 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. The linear ordering can be extended in the obvious fashion to points that are not vertices. 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.

Such a partition and a set of orderings correspond to a layered plane drawing if and only if there do not exist two edges $(u_1, v_1), (u_2, v_2)$ with $u_1 < u_2$ in the same layer and $v_1 > v_2$ in an adjacent layer, as such a pair of edges must cross. When there are no crossings, the edges between two adjacent layers are also linearly ordered, by the obvious extension of the vertex ordering. Given edges e and e' between layers L and L' in a layered plane drawing, the *edge-sequence from e to e'* denotes the set of edges between e and e' in the ordering; if e and e' are the first and last edges between the layers, we refer to *the edge-sequence between L and L'*.

3 Proper h-Layer Plane Drawings

In this section we present an algorithm for recognizing proper h-layer planar graphs; elaborations of this basic algorithm form the basis of our subsequent results. In Section 3.1, after showing that a h-layer planar graph has bounded pathwidth, we give an overview of how dynamic programming can be used to exploit the structure by keeping track of key visibility information. Formal definitions of visibility are presented in Section 3.2, along with foundational lemmas and a way of representing a possible drawing of part of the graph. Section 3.3 considers the problem of reducing the representation in such a way that a fixed-parameter algorithm is possible. Finally, in Section 3.4 the algorithm is presented.

3.1 Bounded Pathwidth and Dynamic Programming

As noted above, the algorithm, which performs dynamic programming on a path decomposition, relies on the fact that an h-layer planar graph has bounded pathwidth, as we now prove.

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

Proof. We first consider a proper h-layer planar graph G. Given an h'-layer plane drawing of G for some $h' \leq h$, we give an algorithm to form a normalized path decomposition of G in which each bag of size h' contains exactly one vertex from each layer. We initialize the path decomposition to consist of the single bag B containing the leftmost vertices on each layer, and repeat the following step until B consists of the rightmost vertices on each layer: find a vertex $v \in B$ such that v is not rightmost in its layer, and for all neighbors w of v, either $w \in B$ or w is to the left of a vertex in B. For u the vertex immediately to the right of v on the same layer, we append the bag $B \setminus \{v\}$ to the path decomposition, followed by $B \cup \{u\} \setminus \{v\}$, and set the current bag $B \leftarrow B \cup \{u\} \setminus \{v\}$.

To prove that the algorithm is correct, it suffices to show that before the last iteration, it is possible to find such a pair v and u; it is straightforward to verify that the resulting normalized path decomposition is valid by checking the three conditions of Definition 1. If v does not exist before the last iteration, then for $B' = \{b_0, b_1, \ldots, b_j\}$ the set of vertices in B that are not rightmost on their layer, enumerated in order of increasing layer number, each b_i has a neighbor to the right of some vertex in B'. Suppose b_0 has a neighbor to the right of some $b_{i_1} \in B'$, and is connected to that neighbor by edge e_1. Then b_{i_1} is connected by edge e_2 to a neighbor to the right of some $b_{i_2} \in B'$, and $i_1 < i_2$, as otherwise e_2 would cross e_1. Continuing this argument, some b_{i_g} (where $g \leq j-1$) must have a neighbor to the right of b_j and be connected to it by e_g. But b_j also has a neighbor to the right of some member of B', and the edge to that neighbor must cross e_g, a contradiction. Thus v exists in B', and u is the vertex immediately to its right on the same layer.

Finally, if G is a non-proper h-layer planar graph, then we can apply the above algorithm to a stretched h-layer drawing of G (and hence a proper h-layer drawing of a graph G' obtained by adding dummy vertices to G) to obtain a path decomposition of G' of width at most $h-1$. To obtain a path decomposition of G, we choose any linear

ordering of the vertices of G, and in each bag of this path decomposition of G', replace a dummy vertex on the edge (u, v) by u, where $u < v$ in the chosen linear ordering. Again, it is straightforward to verify that the properties of a path decomposition hold. □

As stated in Section 2, we can obtain a normalized width-w path decomposition of any graph for which one exists in $O(n)$ time (for fixed w) [Bod96,BK96,GNPR00]. 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 Section 3 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 algorithms of Bodlaender and Kloks [Bod96,BK96], we can test if G has a path decomposition of width at most $h-1$. 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.* [GNPR00]. Let $w < h$ be the width of this path decomposition. (Our algorithm, in fact, works on any path decomposition of fixed width w, and in the generalizations in Section 4, we present modifications of this procedure where the path decomposition has width $w \geq h$.)

Our dynamic programming is structured on the path decomposition; we define $G_t = G[\cup_{s \leq t} P_s]$ and define subproblems for various values of t. In particular, for each bag P_t in turn, we determine all possible ordered assignments of the vertices of P_t to layers and, for each such assignment A (henceforth, an *ordered layer assignment of P_t*), we solve the subproblem of whether G_t is proper h-layer planar with assignment A of the vertices of P_t. The answer to this subproblem is determined by checking the answers for subproblems involving G_{t-1}. To ensure that the time to process each subproblem is a function only of h, we will make use of succinct representations of possible drawings of G_{t-1}. In particular, we rely on the fact that P_{t-1} separates G_{t-1} from the rest of the graph to obtain a representation of size a function of h but not of n.

In order to relate each bag to the rest of the graph, we must represent not only edges between vertices in P_t but also edges with one or more endpoints in $V(G_t) \setminus P_t$. The placement of these edges can affect how the vertices in P_t and their associated edges can be placed in a drawing, as due to planarity, the existence of a particular edge precludes the existence of any edge that could cross it. Representing each edge will require a prohibitively large amount of information as there may be $O(n)$ such edges. Instead, we store permissible locations for new edges as "visibility" information, where if one point (not necessarily a vertex) is visible to another, an edge can be added between without violating the planarity condition. Equivalently, the existence of an edge may render certain points "invisible" to each other. We maintain the visibility information using a *reduced visibility representation* (defined formally later) in which a small number of edges will be represented explicitly and the rest will be "hidden". We will show that given a drawing of G_t, a reduced visibility representation can easily

be constructed, and that the number of possible reduced visibility representations is a function only of h and not of n (thereby permitting an algorithm whose running time is linear in n). We observe that the term "edge" has three distinct uses, which should be clear from context: an edge in a graph, its realization in a drawing, and (if present) its representation in a reduced visibility representation.

Our subproblem then becomes: is there a proper h-layer plane drawing of G_t associated with a given ordered layer assignment of P_t and a given reduced visibility representation of G_t? We answer this question by a dynamic programming recurrence involving G_{t-1} and various pairs of assignments A and visibility representations $\mathcal{R}$. The relationship between G_{t-1} and G_t depends on whether t is odd or even (because the bags P_t form a normalized path decomposition of G). If t is odd, then G_{t-1} has one fewer vertex than G_t, namely the single vertex x in $P_t \setminus P_{t-1}$. Thus, in setting up the recurrence, we can confine ourselves to the pair $(A_{t-1}, \mathcal{R}_{t-1})$ that can be created from a pair $(A_t, \mathcal{R}_t)$ for G_t by deleting x. If t is even, then G_{t-1} is equal to G_t, but the single vertex x in $P_{t-1} \setminus P_t$ is not mentioned in A_t or $\mathcal{R}_t$. In this case, for each subproblem $(A_{t-1}, \mathcal{R}_{t-1})$ for G_{t-1} that yields a YES answer, we store a YES answer for the subproblem $(A_t, \mathcal{R}_t)$, where A_t is formed by removing x from A_{t-1} and $\mathcal{R}_t$ is formed from $\mathcal{R}_{t-1}$ by changing the role of x.

3.2 Visibility Representations

We now proceed to more formally define our notions, and prove several lemmas concerning visibility of vertices. To this end, we add $2h$ artificial boundary vertices, one to the extreme left and one to the extreme right of each layer, and add these to each set P_t. Furthermore, we add edges between each of these leftmost (rightmost) vertices and the leftmost (rightmost) vertices on the layers above and below. The lemmas that follow will later be applied to the graphs G_t and sets of vertices P_t, but we phrase them in terms of an arbitrary graph G and subset S of vertices in the graph. We now define the notion of visibility, which allows us to compute where edges can be added to the graph without introducing crossings.

Definition 3. In a proper h-layer plane drawing, a point x on layer L is *visible* to a point y on an adjacent layer L' (and vice versa) if it is possible to add the line segment between x and y without causing any crossings. Equivalently, if x is visible to y (and hence y is visible to x), then y can *see* x.

The following lemma is a direct consequence of the definition of visibility.

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

1. *if there is an edge (x, y) 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 y' is to the left of y, or x' is to the left of x and y' is to the right of y.* □

We will show that to form a succinct representation of a possible drawing of G_t, it will suffice to represent the visibility information. In order that this information be small in size, we make the key observation that either $G_{t-1} = G_t$ (for t even) or to alter G_{t-1} to form G_t, we will only need to add the single vertex x in $P_t \setminus P_{t-1}$ and edges between x and vertices in P_t (for t odd). As a consequence, it is possible to condense visibility information to focus on the vertices in P_t or, more generally, any special set $S \subseteq V(G)$.

Determining an entry in the dynamic programming solution will entail determining possible drawings of G_t from possible drawings of G_{t-1}. As G_t and G_{t-1} differ in the role of the vertex x in $P_t \setminus P_{t-1}$ or in $P_{t-1} \setminus P_t$, and as the representations focus on vertices in P_t and P_{t-1}, respectively, the critical information includes the positions of the vertices of S on the layers and the presence (or absence) of edges that block visibility. As implied by Lemma 2, the exact placement of edges is unimportant, as is determining which vertices of G are the endpoints of the edges (except for those in S). Moreover, certain edges of the drawing of a graph are unimportant, allowing us to remove them to form a compact representation. We can think of the remaining edges as defining regions (where each region is delimited by two consecutive remaining edges between the same pair of layers); a region is given a color to indicate whether in the original drawing there were (black) or were not (white) any other edges appearing between the remaining edges in the edge-sequence. Each layer L can then be associated with the ordering of the left sides of region boundaries, the right side of the last region, and the subset of S on layer L.

In giving the formal definitions of the parts of the representation, as we present each concept we consider its relation to a drawing. In the following definition of a region, the color is as described above and the shape indicates relationships among endpoints of boundary edges.

Definition 4. A *visibility region* R is a pair $(\mathsf{shape}(R), \mathsf{color}(R))$ where $\mathsf{shape}(R) \in \{\triangle, \triangledown, \square\}$ and $\mathsf{color}(R) \in \{\mathsf{white}, \mathsf{black}\}$. We say that R is *realized at* L_j in a drawing if there exists a pair of edges e and e' between layers L_j and L_{j+1} such that:

1. if $\mathsf{shape}(R) = \triangle$, then e and e' have a common endpoint on L_{j+1};
2. if $\mathsf{shape}(R) = \triangledown$, then e and e' have a common endpoint on L_j;
3. if $\mathsf{shape}(R) = \square$, then e and e' share no endpoints;
4. if $\mathsf{color}(R) = \mathsf{white}$, then e and e' are consecutive edges in the edge-sequence between L_j and L_{j+1}; and
5. if $\mathsf{color}(R) = \mathsf{black}$, then e and e' are not consecutive edges in the edge-sequence between L_j and L_{j+1}.

We say that R is *realized by* e and e', and that e and e' are the *boundaries of* R (or e and e', and by extension, their endpoints, *bound* R).

For convenience, we say that R is *white, black, triangular*, or *square* if $\mathsf{color}(R) = \mathsf{white}$, $\mathsf{color}(R) = \mathsf{black}$, $\mathsf{shape}(R) \neq \square$, or $\mathsf{shape}(R) = \square$, respectively, and that R is an *up-triangle, down-triangle*, or *square* if $\mathsf{shape}(R) = \triangle$, $\mathsf{shape}(R) = \triangledown$, or $\mathsf{shape}(R) = \square$, respectively. At times, we will write $\triangle$, $\blacktriangle$, $\triangledown$, $\blacktriangledown$, $\square$, and $\blacksquare$ as shorthand for $(\triangle, \mathsf{white})$,

$(\triangle, \mathsf{black})$, $(\triangledown, \mathsf{white})$, $(\triangledown, \mathsf{black})$, $(\square, \mathsf{white})$, and $(\square, \mathsf{black})$, respectively. The common endpoint of the boundaries of a triangular region is an *apex*.

As a direct consequence of the definition above, we can extract visibility information from regions.

Lemma 3. *For a visibility region R realized by edges e and e' in the edge-sequence between L_j and L_{j+1},*

1. *if $\mathsf{shape}(R) = \triangle$ and v is the apex of R, then v is the only point on L_{j+1} visible to points between the endpoints of e and e' on L_j;*
2. *if $\mathsf{shape}(R) = \triangledown$ and v is the apex of R, then v is the only point on L_j visible to points between the endpoints of e and e' on L_{j+1}; and*
3. *if $R = \square$, the set of points between and including the endpoints of e and e' on L_{j+1} (L_j) is exactly the set of points visible to all points between the endpoints of e and e' on L_j (L_{j+1}).* □

We will frequently refer to sequences of regions; in a sequence $\sigma = [R_1, \ldots, R_\ell]$ we use $\mathsf{prev}(R_i)$ to denote R_{i-1} (for $1 < i \leq \ell$) and $\mathsf{next}(R_i)$ to denote R_{i+1} (for $1 \leq i < \ell$). We can generalize the notion of regions being realized by edges to apply to a sequence of regions by requiring that consecutive pairs of regions share boundaries.

Definition 5. Given a sequence of regions $\sigma = [R_1, \ldots, R_\ell]$, if each R_i is realized at L_j by edges e_i and e'_i and $e'_i = e_{i+1}$ for all $1 \leq i < \ell$, we say that σ is *realized by* the sequence $[e_1, e_2, \ldots, e_\ell, e'_\ell]$ or that σ is *realized at* L_j.

In order for our representation to indicate the relative positions of sequences of regions with respect to vertices in S as well as other sequences of regions, we need notation that indicates the ordering of endpoints of boundaries that occur on a particular layer. The key observation is that if R is triangular, in the realization the apex of R is an endpoint of the left boundary of the following region in the sequence. Applying this observation to a sequence $\sigma = [R_1, R_2, \ldots, R_\ell]$ where R_1 through $R_{\ell-1}$ are either all up-triangles or all down-triangles, we can view R_ℓ as a representative of all regions in σ on the layer containing the apex. Regions are associated with their left boundaries, and a special symbol is used to denote the right boundary of the rightmost region in the sequence.

Definition 6. For a sequence of visibility regions $\sigma = [R_1, R_2, \ldots, R_\ell]$, the *upward projection of* σ, $\mathsf{UP}(\sigma)$, is the sequence of symbols u_i for each i such that $\mathsf{shape}(R) \neq \triangle$ followed by the special symbol u_{last}. For any i, $\mathsf{up}(R_i)$ is defined as the symbol u_j for the smallest value of j greater than or equal to i such that u_j is in $\mathsf{UP}(\sigma)$ (or u_{last} if no such j exists). The *downward projection of* R $\mathsf{down}(R)$ is defined as above but with $\triangledown$ replacing $\triangle$ and d_{last} replacing u_{last}.

Figure 1 illustrates the realization of a sequence σ of visibility regions as well as its upward and downward projections. In this example, since u_1 and u_5 are consecutive entries in $\mathsf{UP}(\sigma)$, $\mathsf{up}(R_2) = \mathsf{up}(R_3) = \mathsf{up}(R_4) = \mathsf{up}(R_5) = u_5$. Since u_9 represents R_8 in $\mathsf{UP}(\sigma)$, $\mathsf{up}(R_8) = u_9$.

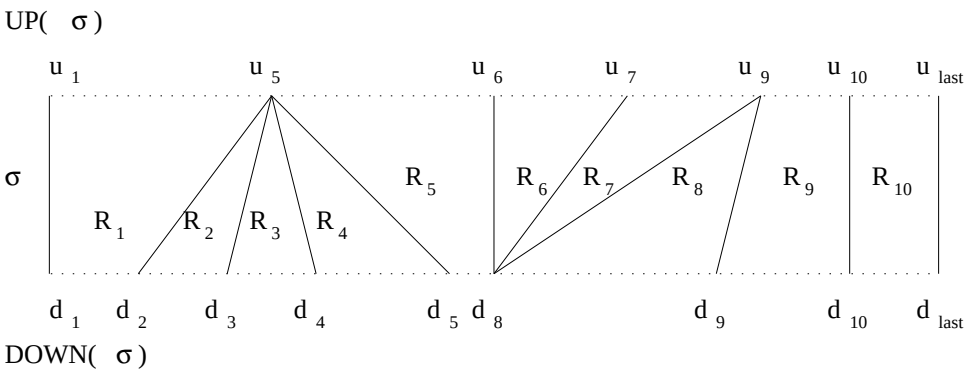

Fig. 1. Upward and downward projections of a sequence of visibility regions.

Definition 7. For a sequence σ of visibility regions realized by the sequence of edges $\mathcal{E} = [e_1, e_2, \ldots, e_\ell, e'_\ell]$ between L_j and L_{j+1}, the *upward (downward) projection of* $\mathcal{E}$, $\mathsf{UP}(\sigma)[\mathcal{E}]$ ($\mathsf{DOWN}(\sigma)[\mathcal{E}]$) is the sequence of vertices formed from $\mathsf{UP}(\sigma)$ ($\mathsf{DOWN}(\sigma)$) by replacing each u_i (d_i) by the endpoint of e_i on L_{j+1} (L_j) and u_{last} (d_{last}) by the endpoint of e'_ℓ on L_{j+1} (L_j).

In our representation we need all vertices in S as well as all edges in $G[S]$ to be represented. The first step is to assign the vertices to layers and to choose orderings among the vertices on each layer.

Definition 8. An *ordered layer assignment* A *of* S is an assignment of the vertices of S to layers such that there is an ordering imposed on the vertices of S in each layer. We use $A[j]$ to denote the sequence of vertices in S on layer L_j in the order imposed by A. A drawing D is *consistent with* A if the vertices of S appear in D on the layers and in the order specified by A.

In the drawing of Figure 2, for $S = \{s_1, \ldots, s_6\}$, $A[1] = [s_5, s_6]$, $A[2] = [s_4]$, $A[3] = [s_3]$, and $A[4] = [s_1, s_2]$.

To represent the ordering of points on a particular layer, we next relate the projections, upward and downward, of sequences of visibility regions onto that layer, both with each other and with vertices in S. We can view the two projections and the vertices in S as three sequences such that elements can be identified (that is, assigned to be the same point) as long as the order within each sequence is preserved, as formalized in the next definition.

Definition 9. Given k sequences $\lambda_1, \ldots, \lambda_k$ of disjoint elements, the sequence λ is formed by a *merge-and-identify* operation if each element of λ is associated with at least one element of some sequence and at most one element of each sequence such that the order within each λ_i is preserved.

As there are many different merge-and-identify operations possible on a set of sequences, specifying the orderings at each layer is necessary.

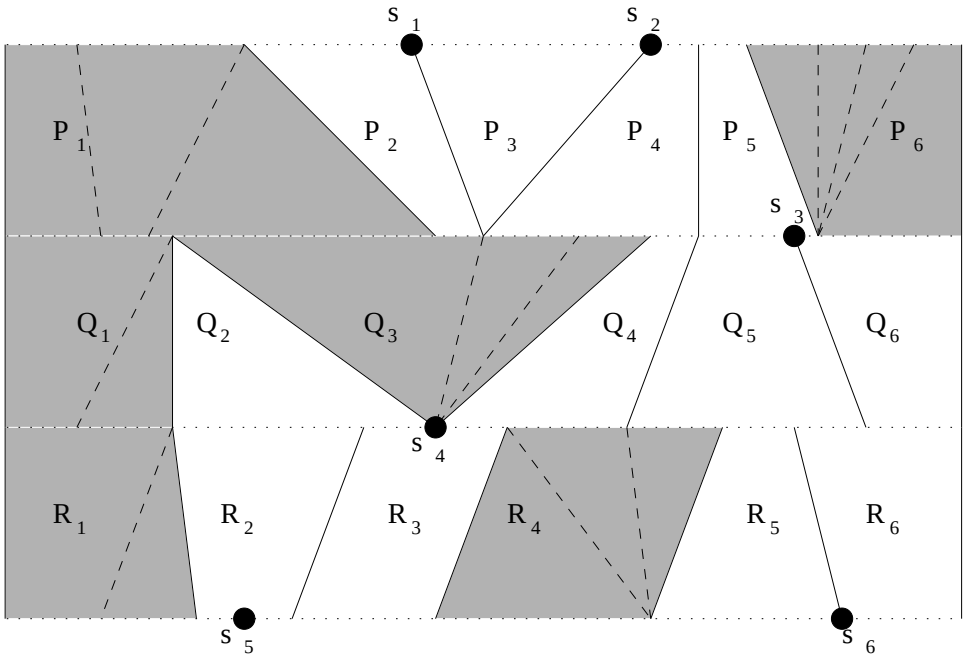

Fig. 2. A visibility representation and an underlying drawing.

Thus, any potential drawing of a graph can be represented in a way that highlights a subset S of the vertices, using the definition below. To show the relationships between $G[S]$ and the rest of the graph, when a vertex s in S is identified with an endpoint of a boundary, it is required that in any realization, that endpoint must map precisely to the vertex s. We can then use this notion to require that each edge in $G[S]$ is represented explicitly.

Definition 10. An *h-layer visibility representation with respect to A and S* consists of:

- $\sigma_1, \sigma_2, \ldots, \sigma_{h-1}$, sequences of visibility regions, and
- $\pi_1, \pi_2, \ldots, \pi_h$, *layer orderings*, where each π_j is formed by a merge-and-identify operation of the sequences $A[j]$, $\mathsf{UP}(\sigma_{j-1})$ (defined to be empty for $j = 1$), and $\mathsf{DOWN}(\sigma_j)$ (defined to be empty for $j = h$),

such that for each edge $(u, v) \in E(G[S])$, u and v are identified, respectively, with the two endpoints of a boundary of a visibility region.

For convenience, if an element v of $\mathsf{UP}(\sigma_{j-1})$ or $\mathsf{DOWN}(\sigma_j)$ is identified with the same element of π_j as an element s of $A[j]$ (and hence an element of S), we say that v is *S-identified* or, more specifically, *s-identified*. If v is not S-identified, we say that v is *S-free*.

If an element of $A[j]$ appears in π_{j-1} or π_j but is not identified with an endpoint of a boundary, we view it as being inside a region, as detailed in the formal definition below.

Definition 11. For any $a \in A[j]$ that is not identified with any element of $\mathsf{UP}(\sigma_{j-1})$ ($\mathsf{DOWN}(\sigma_j)$), if u_i (d_i) is the rightmost element of $\mathsf{UP}(\sigma_{j-1})$ ($\mathsf{DOWN}(\sigma_j)$) to the left of a, element a is said to be *inside* R_i *in* $\mathsf{UP}(\sigma_{j-1})$ ($\mathsf{DOWN}(\sigma_j)$). For a given region R in σ, we say that there is an element *inside* R if there is an element inside R in at least one of $\mathsf{UP}(\sigma)$ and $\mathsf{DOWN}(\sigma)$.

Where appropriate, we may extend the definition to identify a point in a drawing as being *inside* the drawing of a region.

Definition 12. For a graph G, $\mathcal{R} = [\sigma_1, \ldots, \sigma_{h-1}; \pi_1, \ldots, \pi_h]$ an h-layer visibility representation with respect to ordered layer assignment A of $S \subseteq V(G)$, and D an h-layer plane drawing of G, we say that $\mathcal{R}$ *is realized by* D if the following conditions hold:

1. for each σ_j, σ_j is realized at L_j by a sequence of edges $\mathcal{E}_j$ ($1 \leq j \leq h-1$) such that $\mathcal{E}_j$ contains the first and last edges in the edge-sequence between L_j and L_{j+1}, and
2. for each π_j, the sequence of vertices formed by replacing in π_j each element of $\mathsf{UP}(\sigma_{j-1})$ ($\mathsf{DOWN}(\sigma_j)$) by the corresponding vertex in $\mathsf{UP}(\sigma_{j-1})[\mathcal{E}_j]$ ($\mathsf{DOWN}(\sigma_j)[\mathcal{E}_j]$) is consistent with the ordering of vertices in D and for any vertex v replacing a symbol identified with an element $a \in A[j]$, $v = a$ ($1 \leq j \leq h$).

For convenience, we may refer to an element v of a layer ordering as a *vertex*, where the corresponding vertex in the drawing is $D(v)$, the *drawing of* v. Similarly, a boundary e between visibility regions is an *edge* such that the corresponding edge in the drawing is the drawing of e, or $D(e)$; two vertices that are projections of the same edge are *neighbors*.

As a special case, we can immediately convert any h-layer plane drawing D of a graph G into a visibility representation $\mathcal{R}$ (the *trivial representation* of D) by defining the region as bounded by consecutive edges in an edge-sequence between two layers. Each edge in the graph serves as a boundary edge for some visibility region, so each region is bounded by consecutive edges in the edge-sequence. It is then trivial to see that $\mathcal{R}$ is realized by D.

We illustrate the notions described above by means of an example. If we consider the drawing D consisting of all the edges, solid and dashed, in Figure 2, the trivial representation of D has the following sequences of visibility regions:
$\sigma_1 = [\square, \triangle, \square, \square, \triangle, \triangledown, \triangledown, \square, \square]$,
$\sigma_2 = [\square, \triangle, \triangle, \triangledown, \triangledown, \triangledown, \square, \square, \square]$, and
$\sigma_3 = [\square, \square, \triangle, \square, \triangledown, \square, \square, \triangledown, \triangledown, \triangledown, \square]$.

3.3 Reduced Visibility Representations

The size of the trivial representation is a function of the size of the entire graph rather than dependent only on h and the size of S; the key problem is that too much information is being retained. To reduce the size of the representation, we remove unnecessary information by reducing the number of visibility regions. Given a sequence σ of visibility regions, the *meld* of a boundary (or, by extension, the two regions it bounds) results in

the replacement of the two bounded regions by a single black one with shape depending on the shapes of the original regions: the meld of two regions of the same shape results in a black region of the same shape, and the meld of regions of different shapes results in a black square.

When viewed as the melding of a sequence in a visibility representation that is realized by a drawing D, the melding of an edge e can be seen as removing it from the representation. We say that such an edge is a *hidden edge*; any other edge is a *revealed edge*. Similarly, any vertex that is an endpoint of hidden edges only is a *hidden vertex*, and any other vertex is a *revealed vertex*. We need to apply the melding operation with some care, however, as we do not wish any of the vertices of S to be hidden.

In order to preserve visibility information about vertices in S, we restrict the melds that can take place. Ideally we would also like to retain all edges with at least one endpoint in S, but it is not difficult to see that the number of consecutive triangles sharing an S-identified apex could be a function of the size of the graph rather than a function of the size of S. Accordingly, we have to be selective in the choice of edges to protect. We ensure that if a vertex of S is inside or bounds a region, the region is white unless the region is a triangle and S is its apex.

Definition 13. The meld of a boundary e in visibility representation $\mathcal{R}$ is a *legal S-meld* if in the resulting graph each S-identified vertex either:

1. bounds or is inside white regions only, or
2. is the apex of a triangular region R such that for each $R' \in \{\mathsf{prev}(R), \mathsf{next}(R)\}$, either $\mathsf{shape}(R') = \mathsf{shape}(R)$ or $\mathsf{color}(R') = \mathsf{white}$;

the boundary e is then *S-meldable*. Any other meld operation is an *illegal S-meld*, with e being *S-unmeldable*.

Lemma 4. *Given the trivial representation of a drawing D, a set S, and any set of S-meldable edges, any ordering of the melds of those edges will result in the same visibility representation.*

Proof. As it is not difficult to see that the melding of an edge in an edge-sequence between one pair of layers will not have an impact on the representation of another pair of layers, it will suffice to consider the ordering of edges in one edge-sequence. We can view these edges as being in a sequence consistent with their appearance in the edge-sequence. Since any ordering can be achieved by exchanges of adjacent edges, it will suffice to show that for any pair of edges e and e', consecutive in the sequence, the visibility representation $\mathcal{R}_1$ resulting from the meld of e followed by the meld of e' is the same as the representation $\mathcal{R}_2$ resulting from the meld of e followed by the meld of e' (where $\mathcal{R}_1$ and $\mathcal{R}_2$ are both derived from some $\mathcal{R}$ containing e and e' and created from the trivial representation of D by applying zero or more legal S-melds).

We consider the different ways that e and e' may be related, and in each case show that $\mathcal{R}_1 = \mathcal{R}_2$. If there is at least one edge e^* between e and e' in the edge-sequence, the melding of e and e' change disjoint sequences of regions, and hence can occur in either order. If instead e and e' are consecutive in the edge-sequence, they form boundaries of a region. If, for example, that region is square and e and e' each bound another square,

either order of melds will result in the three squares being replaced by a single black square. The result follows from a simple case analysis for each possible shape and color for the three regions bounded by e, e', or both. □

The most succinct representation of a drawing will result from all legal S-melds taking place. For a graph G and a set $S \subseteq V(G)$, a *reduced visibility representation* with respect to ordered layer assignment A for G is a visibility representation with respect to A and S in which no legal S-meld is possible; we denote such a representation as an S-RVR or, without specifying S, simply an RVR. As indicated in the lemma below, we have now achieved the goal of establishing a representation such that the number of possibilities is independent of the size of the graph.

Lemma 5. *For a given ordered layer assignment A and a set S, the number of possible h-layer S-RVR's is at most $(4 \cdot |S|)^{O(h \cdot |S|)}$.*

Proof.

Given a fixed layer assignment, we first determine the number of possible sequences of visibility regions at a particular layer, and then determine the total number of S-RVRs by counting possible merge-and-identify operations.

The size of a particular sequence of visibility regions will depend on the number of illegal S-melds involving regions in the sequence. As an S-meld is illegal only if it would result in a non-apex S-identified vertex v being either inside or bounding a black region, a vertex in S is associated with either one region or a pair of consecutive regions in each of the sequences of visibility regions neighboring the layer on which it resides. Moreover, due to the definition of legal S-melds, there will be at most one region between the regions associated with consecutive S-identified vertices on the same layer. Since the length of each sequence at most $4 \cdot |S|$ and the number of choices for each region is at most six (three shapes multiplied by two colors) the number of choices for one sequence will be at most $6^{4 \cdot |S|}$ or a total of at most $6^{4h \cdot |S|}$ for all layers.

To count the number of layer orderings, we first consider the number of possible results of merge-and-identify operations for a single layer. Ignoring the S-identification in layer orderings for the moment, we observe that each layer ordering requires the merge-and-identify of two sequences each of length at most $4 \cdot |S|$, where after the first element is chosen, at each point there is a choice of (at most) choosing the next element in one sequence as the next in the sequence, choosing the next element in the other sequence as the next in the sequence, or choosing one of these elements as being identified with the current element, for a total of at most four choices for each, or at most $4^{4 \cdot |S|}$. For the S-identification, each member of S is in one of at most $8 \cdot |S|$ locations (at a boundary or in a region), giving a total of $4^{4 \cdot |S|} |S|^{8 \cdot |S|}$ or at most $(4 \cdot |S|)^{8 \cdot |S|}$ choices for a particular layer.

To determine the total number of possible representations, counting over all layers, we obtain the product of the number of sequences of visibility regions and layer orderings, or a total of $6^{4h \cdot |S|} \cdot (4 \cdot |S|)^{8h \cdot |S|}$ which is at most $(4 \cdot |S|)^{O(h \cdot |S|)}$, as claimed. □

If we now consider all legal S-melds in Figure 2, namely those involving dashed edges, we obtain the following S-RVR $\mathcal{R}$, for $S = \{s_1, \ldots, s_6\}$, where identification is

indicated with equal signs:
$\sigma_1 = [\blacksquare, \square, \square, \blacksquare, \square, \square]$,
$\sigma_2 = [\blacksquare, \triangle, \blacktriangledown, \square, \square, \square]$,
$\sigma_3 = [\blacksquare, \square, \triangledown, \square, \square, \blacksquare]$,
$\pi_1 = [d_1, d_2, s_5, d_3, d_4, d_5, d_6 = s_6, d_{\mathsf{last}}]$,
$\pi_2 = [u_1 = d_1, u_2 = d_2, u_3, d_4 = s_4, u_4, d_5, u_5, u_6, d_6, u_{\mathsf{last}} = d_{\mathsf{last}}]$,
$\pi_3 = [u_1 = d_1, u_3, d_2, d_4, u_4, u_5 = d_5, u_6 = s_3, d_6, u_{\mathsf{last}} = d_{\mathsf{last}}]$, and
$\pi_4 = [u_1, u_2, u_3 = s_1, u_4 = s_2, u_5, u_6, u_{\mathsf{last}}]$.

We now establish a few useful properties that have important consequences in the correctness of the algorithm. The following lemma is a simple consequence of the definition of an S-RVR and Lemma 4 above.

Lemma 6. *For any drawing D and set S, there is a unique S-RVR obtained by applying a maximal sequence of legal S-melds to the trivial representation derived from D.* □

For convenience, we use $\mathsf{triv}(D, S)$ to denote the S-RVR defined in Lemma 6.

We use the fact that a visibility representation is realized by a drawing to prove the same property for a related visibility representation. The proof of Lemma 7 is a straightforward case analysis consisting of verifying for each S-meldable edge that the edges and vertices realizing the remaining boundaries and endpoints realize the newly-formed regions in the appropriate way.

Lemma 7. *For any drawing D and set S, if a visibility representation is realized by D, then any legal S-meld will result in a visibility representation that is also realized by D.* □

Lemma 8 follows from Lemma 7 and the observation, made previously, that the trivial representation is realized by D.

Lemma 8. *The* RVR $\mathsf{triv}(D, S)$ *is realized by D.* □

In order to set up a dynamic programming recurrence, we need to demonstrate the relationships between RVRs for G_{t-1} and G_t. As G_{t-1} and G_t differ by a single vertex, we need to consider how the removal or addition of a vertex to S will alter the corresponding RVR.

Suppose t is odd and x is the single vertex in $P_t \setminus P_{t-1}$. Given an RVR $\mathcal{R}_t$ for G_t, we wish to form $\mathcal{R}_{t-1}$ that, upon adding x and incident edges, yields $\mathcal{R}_t$.

We first define the set of regions that may be altered when x is removed. These include not only any regions that x is inside or bounds, but also the regions directly before and after them, as the absence of x, and hence possibly any S-identified vertex in the region formed, may render one or both of the boundaries S-meldable.

Definition 14. The *extended up-neighborhood* (*extended down-neighborhood*) of a vertex $x \in \pi_\ell$ in an S-RVR $\mathcal{R}$ is the subsequence of visibility regions in σ_ℓ ($\sigma_{\ell-1}$) $\mathsf{prev}(R_1), R_1, \ldots, R_m, \mathsf{next}(R_m)$ such that x bounds or is inside each region in the sequence $R_1, \ldots, R_m$, where $\mathsf{prev}(R_1)$ does not exist if R_1 is the first region in σ_ℓ ($\sigma_{\ell-1}$) and $\mathsf{next}(R_m)$ does not exist if R_m is the last region in σ_ℓ ($\sigma_{\ell-1}$).

The *extended neighborhood* of x is a pair of subsequences, consisting of the extended up-neighborhood and the extended down-neighborhood.

In Figure 2, the extended neighborhood of s_4 is $([Q_1, Q_2, Q_3, Q_4, Q_5], [R_2, R_3, R_4])$.

We now consider how the removal of x will change the extended neighborhood. If the extended up-neighborhood is $\sigma = [\mathsf{prev}(R_1), R_1, \ldots, R_m, \mathsf{next}(R_m)]$ for some $m \geq 2$, then the removal of x and all incident edges will leave a white square, as x being S-identified guarantees that R_1 through R_m are either white or only contain edges incident on x. Similarly, if $\sigma = [\mathsf{prev}(R_1), R_1, R_2, \mathsf{next}(R_2)]$, the white region has its shape determined by the shapes of R_1 and R_2, as detailed below.

Definition 15. Given an S-RVR $\mathcal{R} = [\sigma_1, \ldots, \sigma_{h-1}; \pi_1, \ldots, \pi_h]$, a layer ℓ, an element $s \in S$, and an s-identified vertex $x \in \pi_\ell$, the *replacement* $\rho(\sigma)$ of the extended up-neighborhood (extended down-neighborhood) σ of x is defined as follows:

1. if x bounds more than two regions, then for $\sigma = [\mathsf{prev}(R_1), R_1, \ldots, R_m, \mathsf{next}(R_m)]$, $\rho(\sigma)$ is formed by applying all legal $(S - \{s\})$-melds to $[\mathsf{prev}(R_1), \square, \mathsf{next}(R_m)]$,
2. if x bounds exactly two regions, then for $\sigma = [\mathsf{prev}(R_1), R_1, R_2, \mathsf{next}(R_2)]$, $\rho(\sigma)$ is formed by applying all legal $(S - \{s\})$-melds to $[\mathsf{prev}(R_1), R, \mathsf{next}(R_2)]$, where R is the white region formed by the left boundary of R_1 and the right boundary of R_2; and
3. if x bounds no regions, then for $\sigma = [\mathsf{prev}(R), R, \mathsf{next}(R)]$, $\rho(\sigma)$ is formed by applying all legal $(S - \{s\})$-melds to $[\mathsf{prev}(R), R, \mathsf{next}(R)]$.

In our example in Figure 2, the replacement of the extended up-neighborhood of s_4 is $[\blacksquare, \square]$, as the new white square in place of Q_2 through Q_4 can be melded with Q_1 but not Q_5, as the latter region is bounded by s_3, an S-identified vertex. For the extended down-neighborhood, the replacement is $[\square, \blacksquare]$, for similar reasons.

Definition 16. Given an S-RVR $\mathcal{R} = [\sigma_1, \ldots, \sigma_{h-1}; \pi_1, \ldots, \pi_h]$, a layer ℓ, an element $s \in S$, and an s-identified vertex $x \in \pi_\ell$, the $(S - \{s\})$-RVR $\mathcal{R}' = [\sigma'_1, \ldots, \sigma'_{h-1}; \pi'_1, \ldots, \pi'_h]$ *derived from* $\mathcal{R}$ *by removal of* x is formed by first setting $\mathcal{R}'$ to $\mathcal{R}$ and then making the following modifications:

1. remove x from π'_ℓ;
2. replace the extended up-neighborhood of x by its replacement;
3. replace the extended down-neighborhood of x by its replacement; and
4. alter $\pi'_{\ell+1}$ and $\pi'_{\ell-1}$ to reflect the replacements in the obvious manner.

The RVR derived from the removal of s_4 is defined formally as follows:
$\sigma_1 = [\blacksquare, \square, \blacksquare, \square, \square]$,
$\sigma_2 = [\blacksquare, \square, \square]$,
$\sigma_3 = [\blacksquare, \square, \triangledown, \square, \square, \blacksquare]$,
$\pi_1 = [d_1, d_2, s_5, d_{3-4}, d_5, d_6 = s_6, d_{\mathsf{last}}]$,
$\pi_2 = [u_1 = d_{1-4}, u_2, u_{3-4}, d_5, u_5, u_6, d_6, u_{\mathsf{last}} = d_{\mathsf{last}}]$,
$\pi_3 = [u_{1-4} = d_1, d_2, d_4, u_5 = d_5, u_6 = s_3, d_6, u_{\mathsf{last}} = d_{\mathsf{last}}]$, and
$\pi_4 = [u_1, u_2, u_3 = s_1, u_4 = s_2, u_5, u_6, u_{\mathsf{last}}]$.

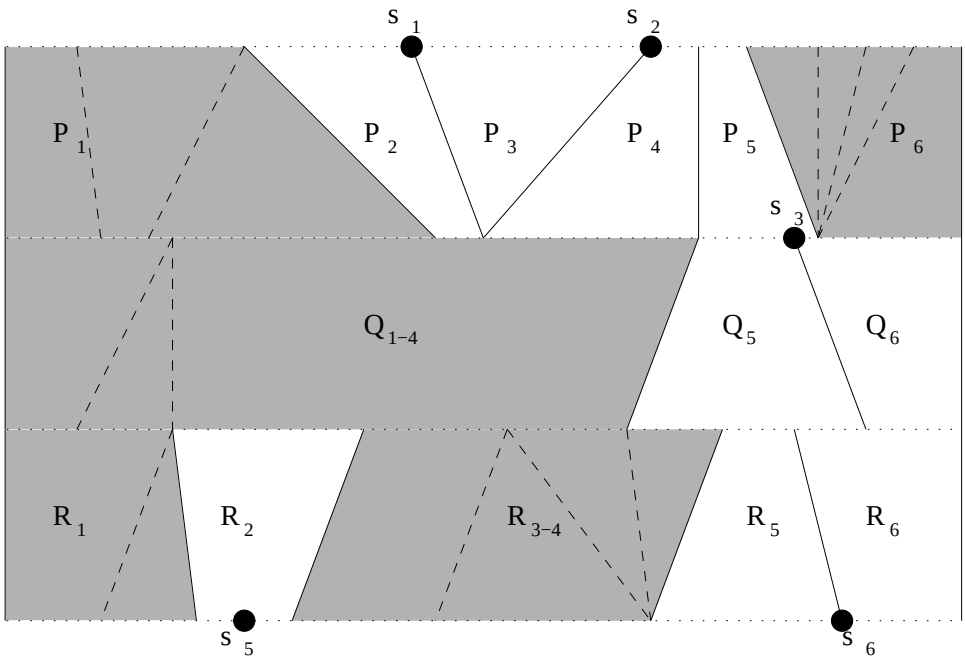

Fig. 3. Removal of s_4 from Figure 2.

Figure 3 illustrates an underlying drawing associated with the RVR; dashed edges are retained to indicate why regions are black.

Now suppose t is even and x is the single vertex in $P_{t-1} \setminus P_t$. Since $P_t \subset P_{t-1}$, we observe that $G_{t-1} = G_t$. We will show that for each P_{t-1}-RVR $\mathcal{R}_{t-1}$ of G_{t-1}, we will obtain the unique P_t-RVR $\mathcal{R}_t$ such that $\mathcal{R}_t$ represents the changes in $\mathcal{R}_{t-1}$ that would result from x being removed from S. The key difference between this situation and the removal of x is that x remains in the representation, perhaps as a hidden vertex and perhaps as a revealed vertex.

We make use of the notions of extended neighborhoods, as in Definition 14, to indicate the subsequences of visibility regions that can be altered by x being removed from S. Unlike in Definition 15, where parts of the extended neighborhoods are replaced by white regions formed by the removal of edges incident on x, here the replacement is by black regions formed by the melding of regions and hiding of edges incident on x.

Definition 17. Given an S-RVR $\mathcal{R} = [\sigma_1, \ldots, \sigma_{h-1}; \pi_1, \ldots, \pi_h]$, a layer ℓ, an element $s \in S$, and an s-identified vertex $x \in \pi_\ell$, the *collapse* $\gamma(\sigma)$ of the extended up-neighborhood (extended down-neighborhood) σ of x is formed by applying to σ all legal $(S - \{s\})$-melds.

Again using the example in Figure 2, the collapse of the extended up-neighbourhood of s_4 is $[\blacksquare, \square]$, the former resulting from the melding of Q_1 through Q_4, as the boundaries are $(S - \{s_4\})$-meldable, and the latter resulting from Q_5, as its boundaries are $(S - \{s_4\})$-unmeldable due to the position of s_3. The collapse of the extended down-neighbourhood of s_4 results in a white square for R_2 and a black square replacing R_3 and R_4.

Definition 18. Given an S-RVR $\mathcal{R} = [\sigma_1, \ldots, \sigma_{h-1}; \pi_1, \ldots, \pi_h]$, a layer ℓ, an element $s \in S$, and an s-identified vertex $x \in \pi_\ell$, the $(S - \{s\})$-RVR $\mathcal{R}' = [\sigma'_1, \ldots, \sigma'_{h-1}; \pi'_1, \ldots, \pi'_h]$ *derived from* $\mathcal{R}$ *by* S*-freeing* x is formed by first setting $\mathcal{R}'$ to $\mathcal{R}$ and then making the following modifications:

1. remove the s-identification of x from π'_ℓ;
2. replace the extended up-neighborhood of x by its collapse;
3. replace the extended down-neighborhood of x by its collapse; and
4. alter $\pi'_{\ell+1}$ and $\pi'_{\ell-1}$ to reflect the replacements in the obvious manner.

In our example, the RVR derived from the S-freeing of s_4 is the same as the RVR derived from the removal of s_4, though in the underlying drawing the edges associated with s_4 would be retained (as dashed edges, using the convention of our illustrations). If instead we compared the removal or S-freeing of s_3, we would see a difference. In the removal of s_3, the replacement of extended down-neighbourhood $[Q_4, Q_5, Q_6]$ would be $[\square, \square]$ since the boundary between Q_4 and Q_5 would be $S - \{s_3\}$-unmeldable (due to the position of s_4) and the boundary between Q_5 and Q_6 would be removed. However, in the S-freeing of s_3, the collapse would be $[\square, \blacksquare]$, since in this case the boundary between Q_5 and Q_6 would be retained.

The lemmas below correlate the alterations made to RVRs with alterations made to drawings. They form the basis of the justification of correctness of the dynamic programming formulation.

Lemma 9. *Suppose x is the single vertex in $P_{t-1} \setminus P_t$ and $\mathcal{R}_{t-1}$ is a P_{t-1}-RVR of G_{t-1} with respect to the ordered layer assignment A_{t-1}, where x appears on layer ℓ in π_ℓ. Furthermore, consider the P_t-RVR $\mathcal{R}_t$ derived from $\mathcal{R}_{t-1}$ by S-freeing x and the ordered layer assignment A_t resulting from removing x from A_{t-1}. Then if an h-layer plane drawing D_{t-1} of G_{t-1} realizes P_{t-1}-RVR $\mathcal{R}_{t-1}$ (with respect to A_{t-1}), D_{t-1} is an h-layer plane drawing of G_t that realizes $\mathcal{R}_t$.*

Proof. Since $G_{t-1} = G_t$ and D_{t-1} is an h-layer plane drawing of G_{t-1}, clearly D_{t-1} is also a plane drawing of G_t.

We observe that since $\mathcal{R}_t$ is formed from $\mathcal{R}_{t-1}$ by a sequence of melds (as indicated in Definition 17), Lemma 7 implies that since D_{t-1} realizes $\mathcal{R}_{t-1}$, D_{t-1} also realizes $\mathcal{R}_t$. This completes the proof of the lemma. □

Lemma 10. *Suppose x is the single vertex in $P_t \setminus P_{t-1}$ and $\mathcal{R}_t$ is a P_t-RVR of G_t with respect to the ordered layer assignment A_t, where x appears on layer ℓ between p_1 and p_2 in π_ℓ. Furthermore, consider any P_{t-1}-RVR $\mathcal{R}_{t-1}$ derived from $\mathcal{R}_t$ by the removal of x and the ordered layer assignment A_{t-1} resulting from removing x from A_t. Then if an h-layer plane drawing D_{t-1} of G_{t-1} realizes P_{t-1}-RVR $\mathcal{R}_{t-1}$, then by adding $D(x)$ between $D(p_1)$ and $D(p_2)$ in D_{t-1} and by adding each edge $(D(x), D(y))$ for y a neighbor of x in P_{t-1}, we form an h-layer plane drawing D_t of G_t that realizes $\mathcal{R}_t$.*

Proof. Since D_{t-1} is planar and D_t differs from D_{t-1} only in the addition of $D(x)$ and edges between $D(x)$ and drawings of the neighbors of x, to prove that D_t is a plane drawing, we need only ensure that for each neighbor y of x in $G[P_t]$ $D(x)$ is visible

to $D(y)$ in D_{t-1}; the proof that D_t realizes $\mathcal{R}_t$ follows from the fact that all edges $(D(x), D(y))$ are then added to form D_t.

We first observe that if the edge $(D(x), D(y))$ exists in D_{t-1}, then since D_t is formed from D_{t-1} by the addition of a vertex and incident edges, the edge will also exist in D_t. Moreover, since D_{t-1} realizes $\mathcal{R}_{t-1}$, if (x, y) is an edge in $\mathcal{R}_{t-1}$, then $(D(x), D(y))$ is an edge in D_{t-1} (and hence in D_t). As $\mathcal{R}_{t-1}$ is derived from $\mathcal{R}_t$ by the removal of x, the only edges $(D(x), D(y))$ that do not exist in D_{t-1} are those for which the edge (x, y) was removed in the formation of $\mathcal{R}_{t-1}$.

To complete the proof, we need to show that if (x, y) was removed in the formation of $\mathcal{R}_{t-1}$, then $D(x)$ is visible to $D(y)$ in D_{t-1}. We consider separately the cases in which x has one or more neighbors y. If x has more than one neighbor, then by Definition 15, all neighbors of x are in a white square, and hence by Lemma 3 are visible to x, as needed. If instead x has a single neighbor, then by Definition 15, y is in a triangular region or a white square, and thereby visible to x, completing the lemma. □

3.4 Fixed-Parameter Recognition Algorithm

In our dynamic programming table, the entry $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ indicates whether or not it is possible to obtain a proper h-layer plane drawing of G_t that realizes the P_t-RVR $\mathcal{R}$ with respect to A. The order of evaluation is by increasing t.

For a bag P_t, ordered layer assignment A of P_t, and P_t-RVR $\mathcal{R}$ with respect to A, a necessary condition for $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ to be YES is that no edges in $G[P_t]$ violate the conditions of a proper h-layer plane drawing. That is, we must ensure that the drawing of $G[P_t]$ consistent with A is itself a proper h-layer plane drawing of $G[P_t]$; there is only one such drawing as each vertex of the graph has its position in D fixed by A.

We now describe how to fill in the dynamic programming table. For $t = 1$, the base case of our dynamic programming recurrence, we consider all possible ordered layer assignments of the vertices in P_1. For any P_1 and A such that the drawing of $G[P_1]$ consistent with A is not a proper h-layer plane drawing, we set $\mathsf{TABLE}\langle 1, A, \mathcal{R}\rangle$ to NO for all values of $\mathcal{R}$. If instead the drawing D of $G[P_1]$ consistent with A is a proper h-layer plane drawing, then for $\mathcal{R} = \mathsf{triv}(D, P_1)$ (or the trivial representation of D, since there are no P_1-meldable edges in $G[P_1]$), we set $\mathsf{TABLE}\langle 1, A, \mathcal{R}\rangle$ to YES, and set $\mathsf{TABLE}\langle 1, A, \mathcal{R}'\rangle$ to NO for each $\mathcal{R}' \neq \mathcal{R}$.

To define the general recurrence, we need to consider two different cases, namely for $|P_t| = w + 1$ (t odd) and $|P_t| = w$ (t even). In the first case, P_{t-1} is missing exactly one vertex that is in P_t, and the algorithm checks the single table entry defined by removing that vertex. In the second case, P_{t-1} has exactly one vertex x that is not in P_t; we consider only an $\mathcal{R}$ that can be derived from $\mathcal{R}'$ by S-freeing x, where $\mathcal{R}'$ is a P_{t-1}-RVR corresponding to a YES-entry for $t - 1$. Both of these tasks can be done in constant time (for fixed h and w). In each case, we first check if the drawing of $G[P_t]$ consistent with A is a proper h-layer plane drawing, and if not, enter NO. Details of the remaining steps are given below.

To compute $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ for odd t, we note that $|P_t| = w+1$ and $|P_{t-1}| = w$. Let x be the single vertex in $P_t \setminus P_{t-1}$. The algorithm computes the ordered layer assignment A' formed by removing x from A and the P_{t-1}-RVR $\mathcal{R}'$ of G_{t-1} derived from $\mathcal{R}$ by

removal of x, and then sets $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ to YES if and only if $\mathsf{TABLE}\langle t-1, A', \mathcal{R}'\rangle$ is YES.

To compute $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ for even t, we note that $|P_t| = w$ and $|P_{t-1}| = w+1$. Let x be the single vertex in $P_{t-1} \setminus P_t$. Note that vertex x is in both G_t and G_{t-1}; the only difference between these two graphs is that $x \notin P_t$ and hence does not appear in the ordered layer assignment A or visibility representation $\mathcal{R}$.

The algorithm determines, for each YES entry of the form $\mathsf{TABLE}\langle t-1, A', \mathcal{R}'\rangle$, the P_t-RVR $\mathcal{R}$ derived from $\mathcal{R}'$ by S-freeing x and the ordered layer assignment A formed by removing x from A', and then sets $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ to YES. Each other table entry of the form $\mathsf{TABLE}\langle t, \cdot, \cdot\rangle$ is set to NO.

Theorem 2. *The entry $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ is YES if and only if $\mathcal{R}$ is an P_t-RVR with respect to A that is realized by some proper h-layer plane drawing of G_t.*

Proof. Our proof is in two parts. We first show by induction on t that if the entry $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ is YES then it is possible to obtain a proper h-layer plane drawing of G_t that realizes $\mathcal{R}$.

Base case:

In the base case of the algorithm, for each possible ordered layer assignment A there is at most one P_1-RVR $\mathcal{R}$ for P_1 such that $\mathsf{TABLE}\langle 1, A, \mathcal{R}\rangle$ is set to YES. For any such A and $\mathcal{R}$, the algorithm sets $\mathsf{TABLE}\langle 1, A, \mathcal{R}\rangle$ to be YES precisely when $\mathcal{R}$ is the trivial representation of D of $G[P_1]$ consistent with A. It follows immediately that D is a proper h-layer plane drawing of $G_1 = G[P_1]$ that realizes $\mathcal{R}$.

Induction step:

To complete the proof, it will suffice to show that if the entry $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ is YES, then there exists an h-layer plane drawing D that realizes $\mathcal{R}$.

Case 1: t is odd ($|P_t| = w+1$)

In the statement of the algorithm, $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ is YES for odd t if and only if $\mathsf{TABLE}\langle t-1, A', \mathcal{R}'\rangle$ is YES for A' the ordered layer assignment resulting from removing $x \in P_t \setminus P_{t-1}$ from A and $\mathcal{R}'$ the P_{t-1}-RVR of G_{t-1} derived from $\mathcal{R}$ by removal of x. We then apply the induction hypothesis to conclude that since $\mathsf{TABLE}\langle t-1, A', \mathcal{R}'\rangle$ is YES, there must exist an h-layer plane drawing D' of G_{t-1} that realizes $\mathcal{R}'$. By Lemma 10, we can then construct a drawing D that realizes $\mathcal{R}$, as needed.

Case 2: t is even ($|P_t| = w$)

For even t, the algorithm sets $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ to YES if and only if $\mathcal{R}$ is derived from $\mathcal{R}'$ by S-freeing x and A is formed from A' by removing x, where $\mathsf{TABLE}\langle t-1, A', \mathcal{R}'\rangle$ is YES. As in the previous case, we apply the induction hypothesis to conclude that there must exist an h-layer plane drawing D' of G_{t-1} that realizes $\mathcal{R}'$. By Lemma 9, D' is an h-layer plane drawing of G_t that realizes $\mathcal{R}$, as needed.

We next show that if there exists an h-layer plane drawing of G_t that realizes $\mathcal{R}$, then the entry $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ is YES.

Base case:

Since $V(G_1) = P_1$, a particular choice of ordered layer assignment A for P_1 completely determines the formation of the RVR. In particular, a h-layer plane drawing D of G_t realizes $\mathcal{R}$ precisely when $\mathcal{R}$ is the trivial representation of D. In this case, by the base case of the algorithm, the associated table entry will be YES.

Induction step:

We now make use of the following induction hypothesis: if there exists an h-layer plane drawing of G_{t-1} with respect to A' that realizes $\mathcal{R}'$, then the entry $\mathsf{TABLE}\langle t-1, A', \mathcal{R}'\rangle$ is YES.

Case 1: t is odd ($|P_t| = w+1$)

Suppose that D is an h-layer plane drawing of G_t that realizes $\mathcal{R}$. In order to show that $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ is YES, by the definition of the algorithm, we need to show that $\mathsf{TABLE}\langle t-1, A', \mathcal{R}'\rangle$ is YES, where x is the single vertex in $P_t \backslash P_{t-1}$, A' is formed by removing x from A, and $\mathcal{R}'$ is the P_{t-1}-RVR of G_{t-1} derived from $\mathcal{R}$ by removal of x. In order to prove that $\mathsf{TABLE}\langle t-1, A', \mathcal{R}'\rangle$ is YES, by the induction hypothesis it suffices to show that there exists an h-layer plane drawing of G_{t-1} with respect to A' that realizes $\mathcal{R}'$.

To complete this case, we will construct a drawing D' and prove that D' is an h-layer plane drawing of G_{t-1} with respect to A' that realizes $\mathcal{R}'$. In particular, we form D' from D by removing $D(x)$ and all incident edges; since G_{t-1} is a subgraph of G_t and D is a drawing of G_t, it is not difficult to see that D' is a h-layer plane drawing of G_{t-1}. The association of edges in D' with boundaries in $\mathcal{R}'$, as required in Definition 12, will follow from the association of edges in D with boundaries in $\mathcal{R}$, as the edges removed from D to form D' are associated with the boundaries removed in the replacement of the extended neighbourhood of x. Finally, by Lemma 7, the subsequent melding will not change the fact that the resulting RVR is realized by D, as needed to complete the proof of this case.

Case 2: t is even ($|P_t| = w$)

We suppose that D is an h-layer plane drawing of G_t that realizes $\mathcal{R}$, and observe that since in this case $G_t = G_{t-1}$, D is also an h-layer plane drawing of G_{t-1}. We let $x \in P_{t-1} \backslash P_t$ and form A' from A by adding x as in D. To complete the proof, we need to show that there exists $\mathcal{R}'$ such that D realizes $\mathcal{R}'$, apply the induction hypothesis to show that $\mathsf{TABLE}\langle t-1, A', \mathcal{R}'\rangle$ is YES, and show that $\mathcal{R}$ is derived from $\mathcal{R}'$ by S-freeing x in order to conclude that $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ is YES, as needed.

We construct $\mathcal{R}'$ by applying all legal P_{t-1}-melds to the trivial representation of D, that is, $\mathcal{R}' = \mathsf{triv}(D, P_{t-1})$. Since by Lemma 8 $\mathcal{R}'$ is realized by D, by the induction hypothesis, $\mathsf{TABLE}\langle t-1, A', \mathcal{R}'\rangle$ is YES. Finally, we show that $\mathcal{R}$ is derived from $\mathcal{R}'$ by S-freeing x by noting that $\mathcal{R} = \mathsf{triv}(D, P_t)$, since the changes due to collapse consist exactly of melding edges that are P_{t-1}-unmeldable but P_t-meldable. □

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

Our extensions follow from the bounded pathwidth of the 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 $h-1$ 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 $h+2r-1$. 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 11. *If G is a k-crossing h-layer graph (r-planarizable h-layer graph), then G has pathwidth at most $h+2k-1$ ($h+2r-1$).* □

Theorem 3. *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 $\mathsf{TABLE}\langle t, A, \mathcal{R}, c\rangle$ indicates whether or not it is possible, by removing at most $r-c$ edges, to obtain a proper h-layer plane drawing of G_t that realizes P_t-RVR $\mathcal{R}$ with respect to ordered layer assignment A. □

Theorem 4. *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 3. 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. We then extend the notion of a visibility representation to include endpoints of red edges in the layer orderings and the notion of legal melds to ensure that in an RVR no red edge is inside or bounds a black region. Moreover, we allow only RVRs in which the number of crossings of red edges is at most k.

In our algorithm, the entry $\mathsf{TABLE}\langle t, A, \mathcal{R}\rangle$ indicates whether or not it is possible to obtain a proper h-layer drawing of G_t that realizes a P_t-RVR $\mathcal{R}$ with respect to ordered layer assignment A such that a subset of edges maps to the set of red edges in $\mathcal{R}$, where the only crossings in the graph involve red edges. We can then obtain a solution if and only if some entry $\mathsf{TABLE}\langle p, *, *\rangle$ is YES.

The modification of an RVR due to the removal or S-freeing of a vertex is similar to the modification used for the h-layer plane drawing algorithm, though colors of associated edges must be taken into account in the total count of red edges and crossings.

The analyses of the various subroutines depends, as before, on the number of layers, and in this case also on the number of red edges. □

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 2.2) has pathwidth at

most $h-1$. 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 5. *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 [Mut01b] 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; as such, in establishing the fixed-parameter tractability of the results, we have not attempted to optimize the running time, nor the counting in Lemma 5. In a companion paper [DFH$^+$02,DFH$^+$] we use other FPT techniques to shed light on the case of 2-layer drawings, obtaining better constants. Later work has examined the role of width and drawings [DMWar,Sud04] and focused on experimental work [SW03] and related problems [DW04].

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 [BM90], and then apply Baker's

approach of dynamic programming on k-outerplanar graphs [Bak94]. 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 [Gro04]. 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.

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

[Bak94] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

[BJL01] C. Buchheim, M. Jünger, and S. Leipert. A fast layout algorithm for k-level graphs. In Marks [Mar01], pages 229–240.

[BK96] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. of Algorithms*, 21:358–402, 1996.

[BM76] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. North-Holland, 1976.

[BM90] D. Bienstock and C. L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmics*, 5:93–109, 1990.

[Bod96] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. on Computing*, 25(6):1305–1317, 1996.

[Car80] M. J. Carpano. Automatic display of hierarchized graphs for computer aided decision analysis. *IEEE Trans. Syst. Man Cybern.*, SMC-10(11):705– 715, 1980.

[DETT99] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Springer, 1999.

[DFH$^+$] 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. To appear.

[DFH$^+$02] 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. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. 9th Internat. Symp. on Graph Drawing (GD '01)*, volume 2265 of *Lecture Notes in Comput. Sci.*, pages 1–15. Springer, 2002.

[DMWar] V. Dujmović, P. Morin, and D. R. Wood. Layout of graphs with bounded tree-width. *SIAM J. Comput.*, to appear.

[DW04] V. Dujmović and S. Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40(1):15–31, 2004.

[EFL97] P. Eades, Q. W. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In S. North, editor, *Proc. Graph Drawing: Symp. on Graph Drawing (GD'96)*, volume 1190 of *Lecture Notes in Comput. Sci.*, pages 113–128. Springer, 1997.

[EW94a] P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoret. Comput. Sci.*, 131(2):361–374, 1994.

[EW94b] P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994.

[GJ83] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM J. Algebraic Discrete Methods*, 4(3):312–316, 1983.

[GKNV93] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Trans. Softw. Eng.*, 19(3):214–230, 1993.

[GNPR00] 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.

[Gro04] M. Grohe. Computing crossing numbers in quadratic time. *J. Comput. System Sci.*, 68(2):285–302, 2004.

[GSBM01] W. Günther, R. Schönfeld, B. Becker, and P. Molitor. k-layer straightline crossing minimization by speeding up sifting. In Marks [Mar01], pages 253–258.

[HK99] P. Healy and A. Kuusik. The vertex-exchange graph and its use in multi-level graph layout. In Kratochvil [Kra99], pages 205–216.

[HR92] L. S. Heath and A. L. Rosenberg. Laying out graphs using queues. *SIAM J. Comput.*, 21(5):927–958, 1992.

[JL99] M. Jünger and S. Leipert. Level planar embedding in linear time. In Kratochvil [Kra99], pages 72–81.

[JLM98] M. Jünger, S. Leipert, and P. Mutzel. Level planarity testing in linear time. In S. Whitesides, editor, *Proc. Graph Drawing: 6th Internat. Symp. (GD'98)*, volume 1547 of *Lecture Notes in Comput. Sci.*, pages 224–237. Springer, 1998.

[JLMO98] 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 Internat. Symp. (GD'97)*, volume 1353 of *Lecture Notes in Comput. Sci.*, pages 13–24. Springer, 1998.

[JM97] 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.

[Kra99] J. Kratochvil, editor. *Proc. Graph Drawing: 7th Internat. Symp. (GD'99)*, volume 1731 of *Lecture Notes in Comput. Sci.* Springer, 1999.

[Len90] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley, 1990.

[Mar01] J. Marks, editor. *Proc. Graph Drawing: 8th Internat. Symp. (GD'00)*, volume 1984 of *Lecture Notes in Comput. Sci.* Springer, 2001.

[MSM99] C. Matuszewski, R. Schönfeld, and P. Molitor. Using sifting for k-layer straightline crossing minimization. In Kratochvil [Kra99], pages 217–224.

[Mut01a] P. Mutzel. An alternative method to crossing minimization on hierarchical graphs. *SIAM J. Optim.*, 11(4):1065–1080, 2001.

[Mut01b] P. Mutzel. Optimization in leveled graphs. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, volume 4, pages 189–196. Kluwer, 2001.

[MW98] P. Mutzel and R. Weiskircher. Two-layer planarization in graph drawing. In K. Y. Chwa and O. H. Ibarra, editors, *Proc. 9th Internat. Symp. on Algorithms and Computation (ISAAC'98)*, volume 1533 of *Lecture Notes in Comput. Sci.*, pages 69–78. Springer, 1998.

[San96] G. Sander. A fast heuristic for hierarchical Manhattan layout. In *Proc. Internat. Symp. on Graph Drawing (GD '95)*, volume 1027 of *Lecture Notes in Comput. Sci.*, pages 447–458. Springer, Berlin, 1996.

[STT81] 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.

[Sud04] Matthew Suderman. Pathwidth and layered drawings of trees. *Internat. J. Comput. Geom. Appl.*, 14(3):203–225, 2004.

[SW03] M. Suderman and S. Whitesides. Experiments with the fixed-parameter approach for two-layer planarization. In G. Liotta, editor, *Proc. Graph Drawing: 11th Internat. Symp. (GD'03)*, volume 2912 of *Lecture Notes in Comput. Sci.*, pages 345–356. Springer, 2003.

[TKY77] 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.

[War77] J. N. Warfield. Crossing theory and hierarchy mapping. *IEEE Trans. Systems Man Cybernet.*, SMC-7(7):505–523, 1977.

[WG86] M. S. Waterman and J. R. Griggs. Interval graphs and maps of DNA. *Bull. Math. Biol.*, 48(2):189–195, 1986.