Alcom-FT Technical Report Series
ALCOMFT-TR-03-68

Derivation of algorithms for cutwidth and related graph layout parameters*

Hans L. Bodlaender† Michael R. Fellows‡ Dimitrios M. Thilikos§

October 16, 2003

Abstract

In this paper, we investigate algorithms for some related graph parameters. Each of these asks for a linear ordering of the vertices of the graph (or can be formulated as such), and constructive linear time algorithms for the fixed parameter versions of the problems have been published for several of these. Examples are cutwidth, pathwidth, and directed or weighted variants of these. However, these algorithms have complicated technical details. This paper attempts to present ideas in these algorithms in a different more easily accessible manner, by showing that the algorithms can be obtained by a stepwise modification of a trivial hypothetical non-deterministic algorithm.

The methodology is applied to rederive known results for the cutwidth and the pathwidth problem, and obtain similar new results for variants of these problems, like directed and weighted variants of cutwidth and modified cutwidth.

1 Introduction

The notion of pathwidth (and the related notion of treewidth) has been applied successfully for constructing algorithms for several problems. One such application area is for problems where linear orderings of the vertices of a given graph are to be found, with a specific parameter of the ordering to be optimised. In this paper, we are interested in a number

*The research of the first and third author was partially supported by EC contract IST-1999-14186: Project ALCOM-FT (Algorithms and Complexity - Future Technologies. The research of the third author was supported by the Spanish CICYT project TIC2000-1970-CE and the Ministry of Education and Culture of Spain (Resolución 31/7/00 – BOE 16/8/00).

†Institute of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands. Email: hansb@cs.uu.nl

‡School of Electrical Engineering and Computer Science, University of Newcastle, Callaghan, NSW 2308, Australia

§Departament de Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya. Campus Nord Mòdul C6. c/ Jordi Girona Salgado 1-3, 08034, Barcelona, Spain

of related notions which appear to allow the same algorithmic approach for solving them. The central problem in the exposition is the CUTWIDTH problem (see Section 2 for the definition). While CUTWIDTH is an NP-complete problem [17], we are interested in the *fixed parameter* variant of it: for fixed k, we ask for an algorithm that given a graph G, decides if the cutwidth of G is at most k, and if so, gives a linear ordering of G with cutwidth at most k. This fixed parameter variant of the problem is known to be linear time solvable (with the constant factor depending exponentially on k) [9, 14, 20]. Such a linear time algorithm can be of the following form: first a path decomposition of bounded pathwidth is found (if the pathwidth of G is more than k, we know that the cutwidth of G is more than k), and then a dynamic programming algorithm is run that uses this path decomposition. Unfortunately, the technical details of this dynamic programming algorithm are rather detailed and complex. Other problems that have a similar algorithmic solution are the pathwidth problem itself (see [7, 4]), and variants on weighted or directed graphs, including directed vertex separation number [2]. See also [1].

In this paper, we attempt to present the central ideas in these algorithms in a different, more easily accessible manner, by showing that the algorithms can be obtained by a stepwise modification of a trivial hypothetical non-deterministic algorithm. Thus, while our resulting algorithms will not be much different from solutions given in the literature, the reader may understand the underlying principles and the correctness of the algorithms much easier.

In addition, we use the methodology for deriving a few new results: we give constructively for each fixed k, linear time algorithms, that, given a graph G, or directed acyclic graph G, decides for some parameters related to cutwidth (e.g., 'directed modified cutwidth', or weighted variants) is at most k, and if so, gives the corresponding linear ordering or topological sort of G. Ingredients of the techniques displayed in this paper appeared in the early 1990's independently in work of Abrahamson and Fellows [1], Lagergren and Arnborg [15], and Bodlaender and Kloks [7]. In [6], a relation between decision and construction versions of algorithms running on path decomposition with an eye to finite state automata was established. More background and more references can be found in [10].

2 Definitions

In this section, we give a number of definitions, used in this paper. The notion of pathwidth was introduced by Robertson and Seymour [18].

Definition 1 *A* path decomposition *of a graph $G = (V, E)$ is a sequence of subsets of vertices $(X_1, X_2, \ldots, X_r)$, such that*

- $\bigcup_{1 \le i \le r} X_i = V$.
- *for all edges $\{v, w\} \in E$, there exists an i, $1 \le i \le r$, with $v \in X_i$ and $w \in X_i$.*
- *for all $i, j, k \in I$: if $i \le j \le k$, then $X_i \cap X_k \subseteq X_j$.*

The width *of a path decomposition* $(X_1, X_2, \ldots, X_r)$ *is* $\max_{1 \le i \le r} |X_i| - 1$. *The pathwidth of a graph* G *is the minimum width over all possible path decompositions of* G.

For directed graphs, these definitions are trivially modified. The definition of a path decomposition of a directed graph $G = (V, A)$ only differs in the second condition: now, we require that for every arc $(v, w) \in A$, there is an i, $1 \le i \le r$ with $v, w \in X_i$. Width and pathwidth are now as for undirected graphs. I.e., the pathwidth of a directed graph G equals the pathwidth of the undirected graph obtained from G by ignoring directions of edges.

Definition 2 *A linear ordering of a graph* $G = (V, E)$ *is a bijective function* $f : V \to \{1, 2, \ldots, |V|\}$. *An edge* $\{v, w\} \in E$ *is said to* cross vertex $x \in V$ *in linear ordering* f, *if* $f(v) < f(x) < f(w)$. *Edge* $\{v, w\} \in E$ *is said to* cross gap i, *if* $f(v) \le i < f(w)$.

For an example of the notion of crossing, see Figure 1.

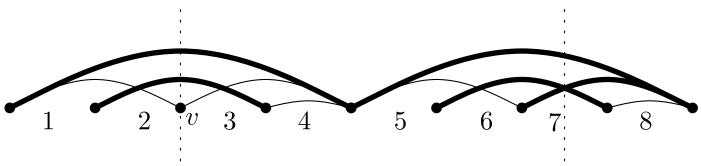

Figure 1: Two edges cross vertex v; three edges cross gap 7.

Definition 3 *Let* $G = (V, E)$ *be a graph, and let* $f : V \to \{1, 2, \ldots, n\}$ *be a linear ordering of* G, $n = |V|$.

1. *For* $1 \le i \le n$, *we denote the number of edges that crosses gap* i *as* $n^f(i) = |\{\{v, w\} \in E \mid f(v) \le i < f(w)\}|$.
2. *The* cutwidth *of* f *is* $\max_{1 \le i \le n} n^f(i)$.
3. *For* $1 \le i \le n$, *we denote the number of edges that cross vertex* $f^{-1}(i)$ *by* $m^f(i) = |\{(u, v) \in E \mid f(u) < i < f(v)\}|$.
4. *The* modified cutwidth *of* f *is* $\max_{1 \le i \le n} m^f(i)$.
5. *The* vertex separation number *of* f *is* $\max_{1 \le i \le n} |\{u \in V \mid \exists v \in V : (u, v) \in E \ \wedge\ f(v) \ge i\ \wedge f(u) < i\}|$.

The cutwidth, modified cutwidth, vertex separation number of a graph G *is the minimum cutwidth, modified cutwidth, vertex separation number over all possible linear orderings of* G.

In Figure 1, we have $n^f(f(v)) = 2$ (two edges cross v), and $m^f(7) = 3$ (three edges cross gap 7.) The cutwidth of the given ordering is three, and the modified cutwidth of this ordering is two.

The pathwidth of a graph is at most its cutwidth, and at most one larger than its modified cutwidth. The vertex separation number of a graph equals its pathwidth [13]. See [5] for an overview of related notions and results.

In this paper, we will also consider a linear ordering of $G = (V, E)$ as a string in V^*, i.e., a string where every element of V appears exactly once. We say that a string t can be obtained by inserting a symbol v into a string s, if s can be written as $s = s_1s_2$, and t can be written as $t = t_1vt_2$, with s_1, s_2 substrings of s and t_1, t_2 substrings of t.

We say a path decomposition $(X_1, \ldots, X_r)$ is *nice*, if $|X_1| = 1$, and for all i, $1 < i \leq r$, there is a v such that $X_i = X_{i-1} \cup \{v\}$ (i is called a *introduce* node, inserting v), or $X_i = X_{i-1} - \{v\}$ (i is called a *forget* node).

It is not hard to see (see e.g. [5], that a path decomposition of width k can be transformed to a nice path decomposition of width k in linear time.

A *terminal graph* is a triple (V, E, X), with (V, E) a graph, and X an ordered set of distinguished vertices from V, called the *terminals*. A terminal graph with k terminals is also called a k-terminal graph. Given two k-terminal graphs G and H, $G \oplus H$ is defined as the graph, obtained by taking the disjoint union of G and H, then identifying the i'th terminal of G with the i'th terminal of H for all i, $1 \leq i \leq k$, and then dropping parallel edges. See Figure 2.

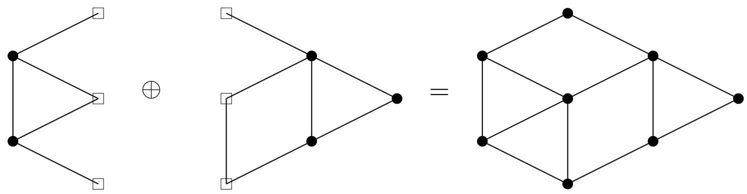

Figure 2: The $\oplus$ operation.

Suppose we have a path decomposition $(X_1, \ldots, X_r)$ of $G = (V, E)$. To each i, $1 \leq i \leq r$, we can associate the terminal graph $G_i = (V_i, E_i, X_i)$, with $V_i = \bigcup_{1 \leq j \leq i} X_j$, and $E_i = |\{\{v, w\} \in E \mid v, w \in V_i\}$.

If we have a nice path decomposition $(X_1, \ldots, X_r)$ of $G = (V, E)$, then we can easily build a $k+1$-coloring $\ell : V \to \{1, \ldots, k+1\}$ of G, such that for all $v, w \in V$, if there is an i with $v, w \in X_i$, then $\ell(v) \neq \ell(w)$. (Go through the path decomposition from left to right, and at each introduce node i, color the inserted vertex $v \in X_i - X_{i-1}$ different from the other vertices in X_i.) For each v, we call $\ell(v)$ the *label* of v.

We now introduce a notation for the possible operations, that given a graph G_{i-1}, build the next graph G_i. If i is an introduce node, suppose the inserted vertex $v \in X_i - X_{i-1}$ has label l', and $S \subseteq \{1, \ldots, k+1\} - \{l'\}$ is the set of the labels of those vertices in $X_{i-1} = X_i - \{v\}$ that are adjacent to v. Now, we can write an introduce operation as $I(r, S)$, meaning an insertion of a new terminal that has number $r \in \{1, \ldots, k+1\}$, with this terminal adjacent to the terminals with numbers in $S \subset \{1, \ldots, k+1\}$. (Note that the introduce operation should fulfil certain criteria; e.g., we may not insert a vertex with a number already given to another present terminal, and for all $s \in S$, there must be a number with that terminal.)

The forget operation can be written as $F(r)$, meaning that the terminal with number r becomes a non-terminal. We denote the terminal with number r as t_r.

Let Σ be a finite alphabet. Σ^* is the set of all (possibly empty) strings with symbols in Σ. The concatenation of strings s and t is denoted st. A string $s \in \Sigma^*$ is a substring of a string $t \in \Sigma^*$, if there are $t', t'' \in \Sigma^*$, with $t = t'st''$. A string s is a subsequence of a string $t = t_1 t_2 \ldots t_r \in \Sigma^*$, if there are indices $1 \leq \alpha(1) < \alpha(2) < \cdots < \alpha(q) \leq r$ with $s = t_{\alpha(1)} \cdots t_{\alpha(q)}$.

Let Σ, Σ_0 be finite alphabets, and let s be a string in $(\Sigma \cup \Sigma_0)^*$. The string $s|_\Sigma$ is the maximal subsequence of s that belongs to Σ^*, i.e., $s|_\Sigma$ is obtained from s by removing all symbols in Σ_0.

If f_1 is a linear order of $G = (V_G, E_G)$, G is a subgraph of $H = (V_H, E_H)$, and f is a linear order of H, we say that f *extends* f_1, if $f_1^{-1}(1), f_1^{-1}(2), \ldots, f_1^{-1}(|V_G|)$ is a subsequence of $f^{-1}(1), f^{-1}(2), \ldots, f^{-1}(|V_H|)$, i.e., f can be obtained from f_1 by inserting the vertices of $V_H - V_G$. Equivalently, we have for all $v, w \in V_G$: $f(v) < f(w) \Leftrightarrow f_1(v) < f_1(w)$.

To a sequence of vertices $v_1, \ldots, v_n$ (or a linear order f), we associate the set of $n+1$ gaps: a gap is the location between two successive vertices, or the location before the first, or after the last vertex.

For a linear order f of $G = (V, E)$ and a subset $W \subseteq V$, we consider the linear order $f|_W$ of $G[W]$, where for all $v, w \in W$, $f(v) < f(w)$, if and only if $f|_W(v) < f|_W(w)$, i.e., $f|_W$ is obtained in the natural way from f by dropping all vertices not in W from the sequence. For $v, w \in V$, we write $f[v, w]$ as the sequence $f|_W$ with $W = \{x \mid f(v) \leq f(x) \leq f(w)\}$, i.e., we take the substring that starts at v and ends at w.

We assume that the reader is familiar with the classic notions of deterministic and non-deterministic automata, and the following result:

Theorem 4 *For every non-deterministic finite automaton A, there is a deterministic finite state automaton B that accepts the same set of strings.*

We will use the acronyms NDFSA and DFSA for non-deterministic finite state automaton and deterministic finite state automaton.

3 An algorithm for cutwidth

We first present our ideas in detail for the notion of *cutwidth*. In later sections, we will show how the same ideas can be applied to other problems. We will derive the following result, already shown in [19].

Theorem 5 *For each k, there is a linear time algorithm, that given a graph G, decides if the cutwidth of G is at most k, and if so, finds a linear ordering of G with cutwidth at most k.*

We will derive the algorithm by first giving a naive non-deterministic algorithm for the problem, and then modifying it step by step, until we have the desired algorithm.

In all cases, we start with finding a nice path decomposition of G of width at most k. If such a path decomposition does not exist, then we know that the cutwidth of G is more than k, and we can stop.

3.1 A non-deterministic algorithm

Consider the following non-deterministic algorithm that finds a linear order of G. The algorithm builds the order by inserting the vertices one by one, i.e., in each step we have a linear order of a subgraph of G.

1. Start with an empty sequence.
2. Now, go through the nice path decomposition from left to right. If we deal with the ith node of the path decomposition, then
   (a) If the ith node is an introduce node of vertex v, then we insert v non-deterministically at some gap in the sequence such that the resulting sequence has cutwidth at most k. (If there is no such gap, the algorithm halts and rejects.)
   (b) If the ith node is a forget node of vertex v, then we do nothing.
3. If all nodes of the path decompositions have been handled, then we output the resulting linear order.

Clearly, this trivial non-deterministic algorithm solves the cutwidth problem. Of course, we must do more in order to turn this algorithm into an efficient deterministic one.

3.2 A non-deterministic algorithm that counts edges

To determine whether a vertex can be inserted at some place in the sequence without violating the cutwidth condition, the algorithm above needs to consult the graph G. Instead, we can keep the information about the number of edges that cross gaps in the sequence. Now, we use sequences in $\mathbf{N}(V, \mathbf{N})^*$, i.e., sequences that have alternatingly an integer in $\mathbf{N}$, and a vertex in V, starting and ending with an integer. The following algorithm does not need to look at G; only when a vertex is inserted, it has to be known to what other vertices it is adjacent. The sequence gives the permutation (ordering) of the vertices, and in addition, between every two vertices the number of edges that cross that position; i.e., if we have sequence $n_0^f, v_1, n_1^f, v_2, \ldots, n_{n-1}^f, v_n, n_n^f$, where f is the linear order that is built by the algorithm. In addition, we maintain that each vertex appears at most once in the sequence, and if we have dealt with node i, then each vertex in V_i appears exactly once in the sequence. In the remainder, we will often drop the superscript f.

We use the same algorithm as in the previous paragraph, but now start with the sequence 0, and we have to detail how an insertion of a vertex v takes place now. This still is fairly obvious:

1. Non-deterministically, a number n_j in the sequence is chosen.
2. n_j is replaced by n_j, v, n_j.
3. For every terminal x with $\{v, x\} \in E$, add one to every number in the sequence between x and v (regardless of the mutual order of v and x).
4. If we obtain a number that is $k + 1$ or larger, we halt and reject.

3.3 A non-deterministic decision algorithm

Suppose now for a moment, that we only want to output whether the cutwidth is at most k, without having to output a corresponding linear ordering. Then, the following version of the algorithm actually also works fine.

In this case, we can actually forget the names of vertices, denoting terminals with their number (i.e., as t_r for $1 \leq r \leq k+1$), and denoting non-terminal vertex as a non-labelled vertex, denoted as $-$. Thus, for $L = \{-, t_1, \ldots, t_k\}$, we have sequences in $\{0, 1, \ldots, k\}$, $(L, \{0, 1, \ldots, k\})^*$.

1. Start with the sequence 0.
2. Now, go through the nice path decomposition from left to right. If we deal with the ith node of the path decomposition, then
   (a) If the ith node is an introduce node of the form $I(r, S)$, then we insert t_r non-deterministically at some gap in the sequence such that the resulting sequence has cutwidth at most k. (If there is no such gap, the algorithm halts and rejects.)

   (b) If the ith node is a forget node of the form $F(r)$, then we replace t_r in the sequence by a symbol $-$.
3. If all nodes of the path decompositions have been handled, then we output *yes*.

When inserting t_r, we again choose a number n_j, replace it by n_j, t_r, n_j, and then for all $s \in S$, we add one to each number in the sequence that is between t_s and t_r. If any such number would become larger than k, then the cutwidth of the sequence would be more than k, so such an insertion cannot be chosen by the non-deterministic algorithm.

Note that at this point, we used the structure of path decomposition, and the fact that each set X_i contains at most $k + 1$ vertices.

3.4 A non-deterministic algorithm that also can construct the sequence

If we let in the above algorithm, alongside with building the sequence, also record in which gap the new vertex was inserted (in a data structure, separate from the sequence), then the non-deterministic algorithm also can construct the sequence after one has been found. In particular, we can maintain a list that denotes in which gap each vertex is inserted; in step 2a of the algorithm of Section 3.3, the gap where the vertex is inserted is put on the list; upon termination, the linear ordering corresponding to this run of the algorithm can be constructed directly (in linear time) from this list.

3.5 A non-deterministic finite state automaton

In this step, we use the crucial observation that turns the method into a linear time algorithm.

First, we give an example. Suppose we have a substring $3 - 5 - 7$ in the sequence. Then one can note that when the non-deterministic algorithm succeeds in completing the sequence, it can also do so by not inserting any vertex on the gap of the 5: everything it inserts there can also be inserted at the 3. Thus, the 5 can be actually be forgotten from the sequence. This will be proved in a more general form below.

Lemma 6 *Let $G_i = (V_i, E_i, X)$, $i = 1, 2$ be r-terminal graphs. Let f be a linear order of $G_1 = (V_1, E_1, X)$ of cutwidth at most k, and let f' be a linear order of $G_1 \oplus G_2$ of cutwidth at most k such that f' extends f. Suppose we have for $1 \leq j_1 < j_2 \leq |V_1|$:*

- $X \cap \{f^{-1}(j_1 + 1), f^{-1}(j_1 + 2), \ldots, f^{-1}(j_2 - 1), f^{-1}(j_2)\} = \emptyset$.
- $n^f(j_1) = \min_{j_1 \leq j \leq j_2} n^f(j)$.
- $n^f(j_2) = \max_{j_1 \leq j \leq j_2} n^f(j)$.

Let f'' be the linear order of $G_1 \oplus G_2$ that is obtained from f' by replacing the substring $f'[f^{-1}(j_1), f^{-1}(j_2)]$ by the substring $f^{-1}(j_1) \cdot (f'[f^{-1}(j_1), f^{-1}(j_2)])|_{V_2 - X} \cdot f'[f^{-1}(j_1 + 1), f^{-1}(j_2)]$. Then the cutwidth of f'' is at most k.

Proof: The change in the linear ordering is graphically depicted in Figure 3.

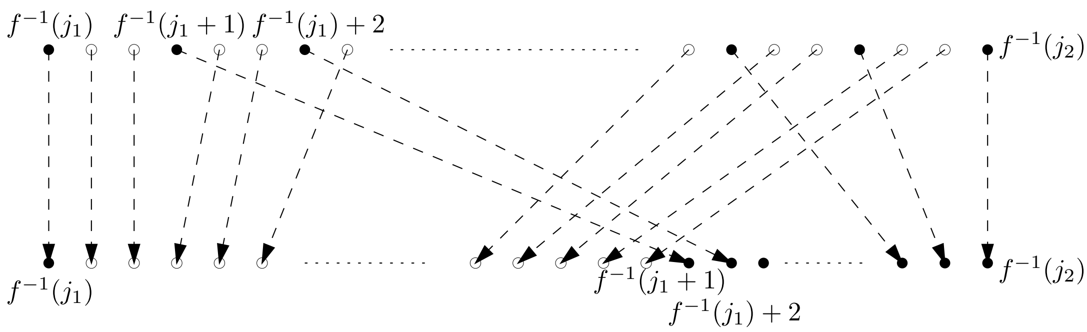

Figure 3: A graphical depiction of the change going from f' to f''

First, we note that for each gap that is not within the replaced substring, the set and hence the number of edges that cross that gap stays the same, so this number is at most k. The same is true for the gap directly after $f^{-1}(j_1)$, and the gap directly before $f^{-1}(j_2)$.

Consider now a gap in the sequence of f'' that is directly after some vertex $v' \in V_2$ (depicted by an 'open' circle in Figure 3. We compare the number of edges crossing this gap by the number of edges crossing the gap after v' in the sequence f'. See Figure 4. We consider three types of edges:

- Edges between vertices in V_2. As the mutual ordering of the vertices in V_2 has not changed, the same edges between vertices in V_2 cross the gap in both gaps.

- Edges between a vertex $v_1 \in V_1$ and a vertex $v_2 \in V_2$. In this case, we must have, by definition of $\oplus$, that $v_1 \in X \subseteq V_2$ or $v_w \in X \subseteq V_1$, and hence this edge either is of the first or third type of edge considered here.

- Edges between two vertices in V_1. Some edges cross the considered gap in sequence f'' but not the considered gap in f' (shown in bold in Figure 4, and other edges this is just the opposite (shown as a dotted line in Figure 4. Note that we have exactly $n^f(j_1)$ edges of this type that cross the gap in f'', and $n^f(j)$ edges of this type that cross the gap in f', for some j, $j_1 \leq j \leq j_2$. (In Figure 4, the last vertex in V_1 before the gap in f' is $f^{-1}(j_1+1)$, and hence this number is $n^f(j_1+1)$.) As $n^f(j_1) = \min_{j_1 \leq j \leq j_2} n^f(j)$, the number of edges of this type crossing the gap in f'' is at most the number of edges of this type crossing the gap in f'.

We conclude that the number of edges crossing the gap after v' in f'' is at most the number of edges crossing the gap after v' in f', hence is at most k.

The last type of gap we consider is a gap in the sequence part $f'[f^{-1}(j_1+1), f^{-1}(j_2)]$ in f'', i.e., after a vertex $v'' \in V_1$, as illustrated in Figure 5 (second part). This gap will be compared with the gap after $f^{-1}(j_2)$ in f'. We distinguish the following types of edges crossing the considered gap in f'':

- Edges between vertices in V_2. Again, the same edges will cross both gaps.

- Edges between a vertex in $v_1 \in V_1$ and a vertex in $v_2 \in V_2$. Again, $v_1 \in X$ or $v_2 \in X$, and hence each such edge is of the first or third type of edges.

- Edges between two vertices in V_1. There are $n^f(j_2)$ such edges for the gap in f', and $n^f(j)$ such edges for the gap in f'', for some j, $j_1 \leq j \leq j_2$. As we have for each such j, $n^f(j_2) \geq n^f(j)$, the number of edges of this type crossing the gap in f'' is at most the number of edges of this type crossing the gap in f'.

We can conclude that the number of edges crossing the gap after v'' in f'' is at most this number for the sequence f', hence again at most k. Thus, it follows that the cutwidth of f'' is at most k. $\square$

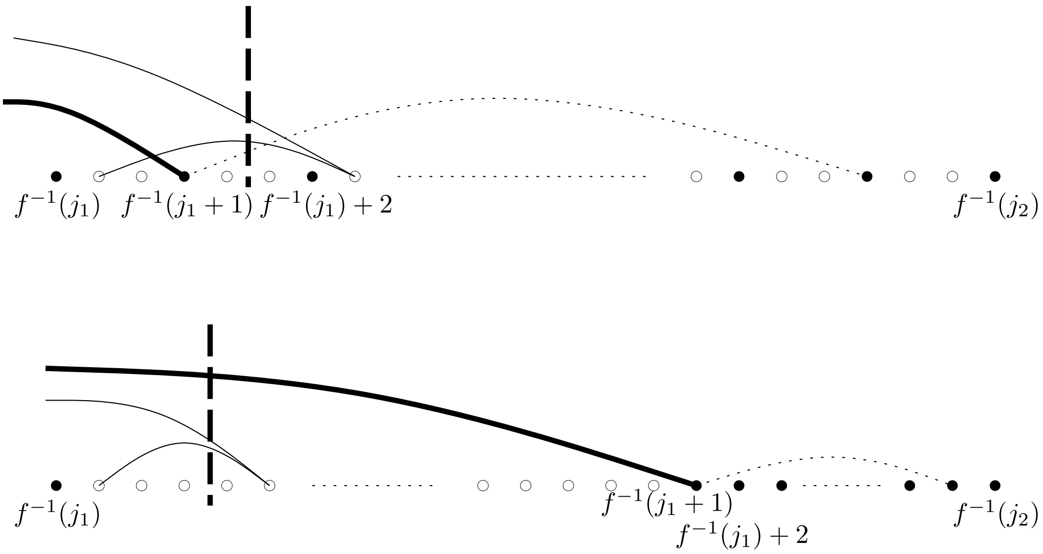

Figure 4: Illustration with the proof of Lemma 6

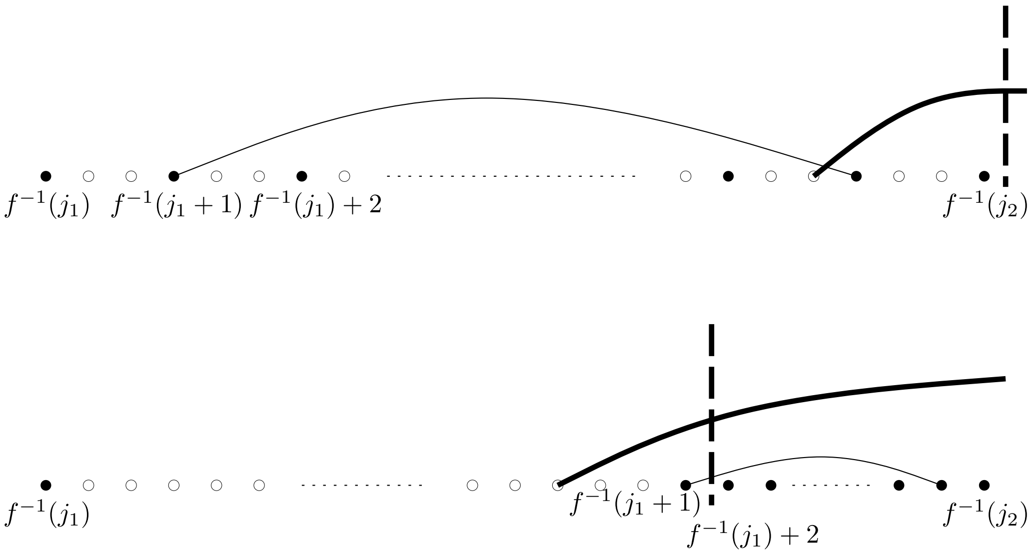

Figure 5: Illustration with the proof of Lemma 6

A similar lemma can be proved for the case that $n^f(j_1) = \max_{j_1 \leq j \leq j_2} n^f(j)$ and $n^f(j_2) = \min_{j_1 \leq j \leq j_2} n^f(j)$, and all other conditions are as in the lemma. The main reason why this lemma is interesting is the following corollary.

Corollary 7 *Consider the non-deterministic decision algorithm. Suppose at some point, a sequence $s_1 \cdot s_2 \cdot s_3$, with $s_2 = n_1 - n_2 - \cdots - n_q$. s_2 does not contain a character of the form t_r. Suppose it holds that $n_1 = \min\{n_1, \ldots, n_q\}$ and $n_q = \max\{n_1, \ldots, n_q\}$, or that $n_1 = \max\{n_1, \ldots, n_q\}$ and $n_q = \min\{n_1, \ldots, n_q\}$. Then, if there is an extension of the sequence that corresponds to a linear order of G of cutwidth at most k, there is such an extension that does not insert any vertex on the gaps corresponding to the numbers $n_2, \ldots, n_{q-1}$ in substring s_2.*

This gives an obvious modification to the non-deterministic algorithm: when choosing where to insert a vertex, forbid to insert a vertex on any of the gaps $n_2, \ldots, n_{q-1}$, as indicated in Corollary 7. But then we may note that when we do not insert at the gaps corresponding to these numbers $n_2, \ldots, n_{q-1}$, we can actually forget these numbers: any insertion of a vertex that would increase the number of edges that crosses a gap corresponding to such a number also will cross the gap with value $\max_{n_1, n_q}$, and thus we can drop the numbers $n_2, \ldots, n_{q-1}$ from the sequence.

The discussion leads to observing that the following non-deterministic algorithm indeed also correctly decides whether the cutwidth is at most k. The insertion of a vertex is still done as in Section 3.2; the main difference in the algorithm below is in the compression operation.

1. Start with the sequence 0.
2. Now, go through the nice path decomposition from left to right. If we deal with the ith node of the path decomposition, then
   (a) If the ith node is an introduce node of the form $I(r, S)$, then we insert t_r non-deterministically at some gap in the sequence such that the resulting sequence has cutwidth at most k. If there is no such gap, the algorithm halts and rejects.
   (b) If the ith node is a forget node of the form $F(r)$, then we replace t_r in the sequence by a symbol $-$.
   (c) In both cases, check if the sequence has a substring of the form $n_1 - n_2 - \cdots - n_q$ with $\{n_{j_1}, n_{j_2}\} = \{\min_{j_1 \leq j \leq j_2} n_j, \max_{j_1 \leq j \leq j_2} n_j\}$. If so, replace the substring $n_1 - n_2 - \cdots - n_q$ with the substring $n_1 - n_q$. Repeat this step until such a replacement is no longer possible. (We call this a compression operation.)
3. If all nodes of the path decompositions have been handled, then we output yes.

A sequence of integers that does not have a substring of length at least three $n_1 \ldots n_q$, with $n_1 = \min_{1 \leq i \leq q} n_i$, and $n_q = \max_{1 \leq i \leq q} n_i$, or $n_1 = \max_{1 \leq i \leq q} n_i$, and $n_q = \min_{1 \leq i \leq q} n_i$ is called a *typical sequence*. Note that every sequence formed by the algorithm has between every two successive terminals always a typical sequence (with "$-$"-symbols between each pair of successive integers.) In [7, Lemma 3.5], it is shown that there are at most $\frac{8}{3}2^{2k}$ typical sequences of the integers in $\{0, 1, \ldots, k\}$. This means that the number of possible sequences that can be formed in the algorithm actually is bounded by a function of k (we have a constant number of terminals, and between each pair of terminals a string from some constant size set); i.e., is bounded by a constant if we assume that k is a constant. In [19, Lemma 3.2], it is shown that this number of sequences is at most $k!(\frac{8}{3}2^{2k})^{k+1}$.

This implies that the algorithm can actually be viewed as a NDFSA. The input to the automaton is a string that describes the nice path decompositions; with symbols from the finite alphabet $\{I(r,S) \mid 1 \leq r \leq k+1,\ S \subseteq \{1, \ldots, k+1\}\} \cup \{F(r) \mid 1 \leq r \leq k+1\}$, i.e., the input string is the sequence of successive introduce and forget operations that give the nice path decomposition. The states of the automaton are the different strings that can be formed during the process: as there is a finite number of different such strings, the number of states is finite. The possible next states are determined by the symbol (the type of node we deal with in the path decomposition), possibly a non-deterministic choice (where to insert the new vertex) and the old state (the sequence to which the vertex is inserted or the sequence where the forgotten node is replaced by an $-$).

We now have arrived at an algorithm that actually is just a NDFSA (in some disguise). As for every NDFSA, there is an equivalent DFSA, and the latter one corresponds to a linear time algorithm solving the decision problem, we directly have a proof of the existence of a linear time decision algorithm for the cutwidth$-k$ problem, k fixed. This will be made explicit below; we will also show how to construct the corresponding sequences.

3.6 Constructive non deterministic finite state automaton

In this step, we give a version of the NDFSA of the previous step that also is *constructive*, i.e., it can construct the linear ordering of cutwidth at most k, if it exists.

To this end, we augment the algorithm of the previous step by maintaining besides the compressed sequence, denoted here as s, also a linear ordering of the vertices inserted so far, here denoted as π. π is implemented as a linked list, with in addition a record for every gap that still appears in s. s is also a linked list, with separate records for the vertices appearing in s, and the gaps (i.e., integer numbers); with for each element in the list an additional pointer to the corresponding element in the data structure for π. An example is given in Figure 6.

1. Start with s to be the sequence 0, and π a list with one record; with a pointer from the record in s to the record in π.

2. Now, go through the nice path decomposition from left to right. If we deal with the ith node of the path decomposition, then

   (a) If the ith node is an introduce node of the form $I(r,S)$, then we insert t_r non-deterministically at some gap in the sequence such that the resulting sequence has cutwidth at most k. In addition, the data structure is used to also insert t_r

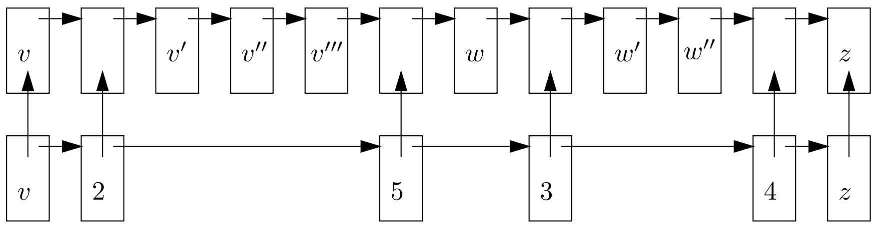

Figure 6: Example of part of data structure. v and z are active vertices

in the sequence π, on 'the same place'. If there is no such gap, the algorithm halts and rejects.

(b) If the ith node is a forget node of the form $F(r)$, then we replace t_r in the sequence by a symbol $-$.

(c) In both cases, check if the sequence has a substring of the form $n_1 - n_2 - \cdots - n_q$ with $\{n_{j_1}, n_{j_2}\} = \{\min_{j_1 \leq j \leq j_2} n_j, \max_{j_1 \leq j \leq j_2} n_j\}$. If so, replace the substring $n_1 - n_2 - \cdots - n_q$ with the substring $n_1 - n_q$. Repeat this step until such a replacement is no longer possible. (We call this a compression operation.)

3. If all nodes of the path decompositions have been handled, then we output π.

From Figure 7 we see how an insertion can be done. The figure shows the fragment of the data structure after insertion of a vertex y, where y is assumed to be adjacent to terminals v and z. By following the pointer from the gap in s in which t_r is inserted, we find the corresponding gap in π, and hence s keeps being a representation of π.

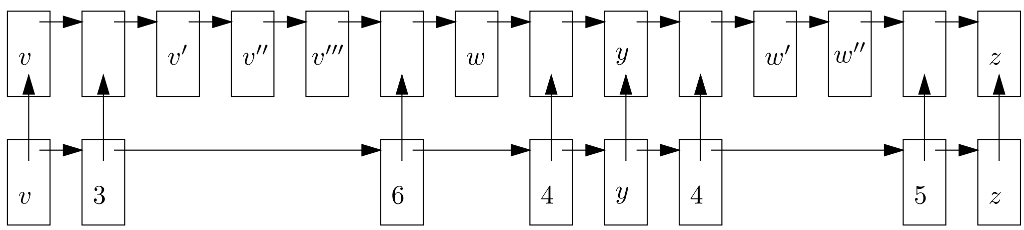

Figure 7: The data structure after insertion of a vertex y; y is assumed to be only adjacent to v and z.

One can easily see that each node of the path decomposition can be handled with a constant number of operations, so we still have a non-deterministic linear time algorithm. In the next section, we will turn to a deterministic algorithm, but the construction of the linear ordering for the deterministic algorithm is there postponed to a later section (Section 3.8).

3.7 A deterministic decision algorithm

As is known for finite state automata, NDFSA's recognise the same set of languages as DFSA's. We can employ the (actually simple) tabulation technique here too, and arrive at our deterministic algorithm. Thus, we take the algorithm of Section 3.5, and make it deterministic, using this technique.

1. Start with a set of sequences A_0 that initially contains one sequence 0.
2. Now, go through the nice path decomposition from left to right. If we deal with the ith node of the path decomposition, then
   (a) Set $A_i = \emptyset$ and $B_i = \emptyset$.
   (b) If the ith node is an introduce node of the form $I(r, S)$, then perform the following step for every sequence $s \in A_{i-1}$:

      For every gap in the sequence s, look to the sequence s' obtained by inserting t_r in s in that gap. If s' has cutwidth at most k and $s' \notin B_i$, then insert s' in B_i.

   (c) If the ith node is a forget node of the form $F(r)$, then for every sequence $s \in A_{i-1}$, let s' be the sequence obtained by replacing t_r in s by a symbol $-$. If $s' \notin B_i$, then insert s' in B_i.
   (d) In both cases, perform the following step for every $s \in B_i$:

      Check if s has a substring of the form $n_1 - n_2 - \cdots - n_q$ with $\{n_{j_1}, n_{j_2}\} = \{\min_{j_1 \le j \le j_2} n_j, \max_{j_1 \le j \le j_2} n_j\}$. If so, replace the substring $n_1 - n_2 - \cdots - n_q$ with the substring $n_1 - n_q$. Repeat this step until such a replacement is no longer possible. Let s' be the resulting sequence. If $s' \notin A_i$, then insert s' in A_i.

3. If all r nodes of the path decompositions have been handled, then we output yes, if and only if $A_r \neq \emptyset$.

What we did above is just tabulating all possible sequences the non-deterministic algorithm can attain at its steps, thus arriving at an equivalent deterministic algorithm.

We mentioned earlier that the number of possible sequences is bounded by a function of k ($k!(\frac{8}{3}2^{2k})^{k+1}$ by [19]); thus, if k is fixed, each set S is of constant size. Hence, the algorithm above uses linear time.

In the next final step we see how this deterministic decision algorithm can be turned into a constructive one, i.e., one that also constructs a linear ordering of cutwidth at most k if there is one.

3.8 A deterministic decision algorithm that also can construct the sequence

To turn the deterministic decision algorithm of the previous step into a deterministic algorithm that also can construct a linear ordering of cutwidth at most k, and still uses linear time, we employ the standard tabulation technique, often used to turn dynamic programming decision algorithms into constructive algorithms, and the data structure for the non-deterministic constructive algorithm. Basically, we first decide whether the cutwidth of the input graph is at most k, then determine the sequence of choices that should be made by the non-deterministic algorithm to lead to acceptance, and then run the 'non-deterministic constructive algorithm', but with these choices.

Thus, our algorithm has three phases. The first phase is similar to the deterministic decision algorithm, but does a little more bookkeeping. We maintain sets of triples; the first element is as in the previous algorithm, while the second is either a sequence or the default $-$ symbol, and the third an integer (denoting a gap where a vertex is inserted) or the default $-$ symbol (used in case of a forget operation). The meaning of the second element of the triple is as follows. The $-$ symbol is only used for the first empty sequence at the start of the algorithm. In all other cases, the second element is the sequence before the introduce or forget operation. So, e.g., in case of a node of type $I(r, S)$, a triple (s, s', j) means that if we insert t_r in s' at the jth gap, and then compress, we obtain s.

Below, we use $*$ to denote any possible value.

1. Start with a set A_0 that initially contains one triple $< 0, -, - >$.
2. Now, go through the nice path decomposition from left to right. If we deal with the ith node of the path decomposition, then
   (a) If the ith node is an introduce node of the form $I(r, S)$, then perform the following step for every element of the form $< s, *, * > \in A_{i-1}$:

      For every gap in the sequence s, look to the sequence s' obtained by inserting t_r in s in that gap. Suppose this is the j'th gap in s. If s' has cutwidth at most k, then insert the triple $< s', s, j >$ in the set B_i.

   (b) If the ith node is a forget node of the form $F(r)$, then for every element of the form $< s, *, * > \in A_{i-1}$, let s' be the sequence obtained by replacing t_r in s by a symbol $-$. Insert $< s, s'', - >$ in B_i.
   (c) In both cases, perform the following step for every $< s, s'', j > \in B_i$:

      Check if s has a substring of the form $n_1 - n_2 - \cdots - n_q$ with $\{n_{j_1}, n_{j_2}\} = \{\min_{j_1 \leq j \leq j_2} n_j, \max_{j_1 \leq j \leq j_2} n_j\}$. If so, replace the substring $n_1 - n_2 - \cdots - n_q$ with the substring $n_1 - n_q$. Repeat this step until such a replacement is no longer possible. Let s' be the resulting sequence. If there is no triple of the form $< s', *, * >$ in A_i, then insert $< s', s'', j >$ in A_i.

3. If $A_r = \emptyset$, then halt: the cutwidth of G is more than k.
4. Select a triple $q_r =< s_r, s_{r-1}, \alpha >$ from A_r.
5. For $i = r - 1$ downto 1 do (i.e., go through the nice path decomposition from right to left): select a triple of the form $q_i =< s_i, s', \alpha >$ from A_i, and set $s_= s'$.
6. Set $i = 0$, let $s = s_0$ be the sequence 0, and π a list with one record with a pointer from the record in s_0 to the record in π.
7. Now, go through the nice path decomposition from left to right. If we deal with the ith node of the path decomposition, then
   (a) If the ith node is an introduce node of the form $I(r, S)$, then suppose $q_i =< s_i, s_{i-1}, j_i >$. Then, we insert t_r at gap j in sequence $s = s_{i-1}$. In addition, the data structure is used to also insert t_r in the sequence π, on 'the same place'.
   (b) If the ith node is a forget node of the form $F(r)$, then we replace t_r in the sequence by a symbol $-$.
   (c) In both cases, do the compression operation. Now, $s = s_0$.
8. If all nodes of the path decompositions have been handled, then we output π.

This final step consisted mainly of an implementation of the standard technique of transforming a dynamic programming algorithm that decides upon a problem to one that also constructs the corresponding solution if it exists. The additional work, compared with the algorithm of the previous step is clearly linear in the number of nodes of the tree decomposition. Thus, we have obtained Theorem 5.

4 Other problems

In this section, we show that the technique can be applied to several other problems. At one hand, we give variants of the cutwidth problem and discuss how the method of the previous section can be modified to handle these as well; at the other hand, for some problems we give a transformation that enables to solve the problem with help of an algorithm for another problem.

4.1 Directed cutwidth

In the directed cutwidth problem, we have a directed acyclic graph $G = (V, A)$, and look for a topological sort (i.e., a linear order f such that for all arcs $(v, w) \in A$: $f(v) < f(w)$ with minimum cutwidth, where the cutwidth is defined as usual. One can observe that basically the same algorithm can be used for this problem as for the undirected case: when inserting a vertex, only those insertions can be done that have no arcs directed in the wrong way. To be precise, consider the linear orders f, f' and f'' from Lemma 6. A simple case analysis shows that if for every arc $(v, w) \in A$: $f(v) < f(w)$ and $f'(v) < f'(w)$, then we also have that for every arc $(v, w) \in A$: $f''(v) < f''(w)$. Hence, Corollary 7 holds also for the directed cutwidth problem, and thus we can derive, exactly as for cutwidth the following result.

Theorem 8 *For each k, there is a linear time algorithm, that given a directed acyclic graph G, decides if the directed cutwidth of G is at most k, and if so, finds a topological sort of G with cutwidth at most k.*

4.2 Mixed graphs

Also, one can solve the cutwidth problem for mixed graphs: given a mixed graph $G = (V, E, A)$, where E is the set of undirected arcs, and A is the set of arcs, we look for a linear order such that for every arc $(v, w) \in A$: $f(v) < f(w)$, of minimum cutwidth. Again, this problem can be solved with the same method.

4.3 Weighted graphs

Suppose the edges (or arcs) have (small) integer weights (in $\{0, 1, \ldots, k\}$). The cutwidth is now modified accordingly, by adding the weights of edges across cuts. When given a path decomposition of the input graph, the method described in the previous section still works as we can again prove variants of Lemma 6 and Corollary 7 for the case of weighted graphs. Note that now, it may be that G has pathwidth more than k, in cases where there are several edges of weight 0, and thus we assume we are given a path decomposition of G of bounded width. (This assumption is not necessary when all weights are at least one, as in that case, any positive instance must have bounded pathwidth.)

Theorem 9 *For each fixed k, l, there exists a linear time algorithm, that given a graph G whose edges have integer weights in $\{0, 1, \ldots, k\}$ with a path decomposition of G of width at most l, decides if the weighted cutwidth of G is at most k, and if so, finds a linear ordering of G of weighted cutwidth at most k.*

For the directed weighted variant of the problem, we similarly get the following result.

Theorem 10 *For each fixed k, l, there exists a linear time algorithm, that given a directed acyclic graph G whose arcs have integer weights in $\{0, 1, \ldots, k\}$ with a path decomposition of G of width at most l, decides if the weighted directed cutwidth of G is at most k, and if so, finds a topological sort of G of weighted directed cutwidth at most k.*

4.4 Pathwidth

The pathwidth problem can also be solved in this way. There are several methods to solve the pathwidth problem. Instead of solving it directly, as is done in [7], we now instead model it as a weighted cutwidth problem on a directed graph, showing how the pathwidth problem can be translated to weighted directed cutwidth.

However, a direct derivation of an algorithm for pathwidth, similar to our derivation of the cutwidth problem is also possible - however, it seems that such an algorithm would be essentially the same as the algorithm obtained by the modification given here.

Let $G = (V, E)$ be an undirected graph. The directed weighted graph G^M is defined as follows:

- G^M has vertices $V^M = \{b_v \mid v \in V\} \cup \{e_v \mid v \in V\}$.
- G^M has two types of arcs. For every $v \in V$, we have an arc (b_v, e_v) of weight 1. For every $\{v, w\} \in E$, we have two arcs (b_v, e_w) and (b_w, e_v), both of weight 0.

Lemma 11 *Let G be an undirected graph. Then G has pathwidth at most k, if and only if G^M has weighted directed cutwidth at most $k + 1$.*

Proof: Suppose $(X_1, \ldots, X_r)$ is a nice path decomposition of G. Now, let f be the linear order of G^M, defined by $f(b_v) = i$, if X_i is an introduce node that introduces v, and $f(e_v) = i$, if X_i is a forget node that forgets v.

We claim that f is a topological sort of G of cutwidth at most $k+1$. As for every v, the node that introduces v is before the node that forgets v, we have that $f(b_v) < f(e_v)$. Also, for every edge $\{v, w\} \in E$, there must be a node that contains both v and w. Hence, we cannot forget v before w is introduced, and vice versa, so $f(b_v) < f(e_w)$ and $f(b_w) < f(e_v)$.

The cutwidth of f is at most $k + 1$. Consider an i. Consider the arcs (b_v, e_v) with $f(b_v) \leq i < f(e_v)$ – the other arcs have zero weight. For each such arc, we have that $v \in X_i$, hence there are at $|X_i| \leq k + 1$ such arcs.

The reverse is similar. □

Lemma 12 *Let G be a graph of pathwidth at most k. Then, G^M has pathwidth at most $2k + 1$.*

Proof: Replace in a path decomposition of G every vertex v in each bag by the vertices b_v and e_v. □

These two lemmas show.

Theorem 13 (Bodlaender, Kloks [7]) *For each fixed k, l, there exists a linear time algorithm, that given a graph G with a path decomposition of G of width at most l, decides if the pathwidth of G is at most k, and if so, finds a path decomposition of G of width at most k.*

Proof: First build a path decomposition of G^M of width at most $2l+1$ (Lemma 12. Then, check if G^M has weighted directed cutwidth at most $k + 1$. (See Lemma 11.) This check can be done in the same way as the cutwidth problem (see the discussions in the previous sections.) □

An algorithm that does not make use anymore of the notion of cutwidth can be also derived as a final step. We will not elaborate this, as, while the details are not difficult, they would require significant space.

This step can be used as a subroutine in the algorithm given in [3] to obtain a linear time algorithm to determine whether the pathwidth of a graph is at most k and construct a corresponding path decomposition if it exists, for constant k.

4.5 Directed vertex separation number

The directed vertex separation number problem is the following: suppose we are given a directed acyclic graph $G = (V, A)$, find a topological sort of G with minimum vertex separation number, where the vertex separation number of ordering $f : V \to \{1, \ldots, |V|\}$ equals

$$\max_{v \in V} |\{w \in V \mid f(w) < f(v) \wedge \exists\{w, x\} \in A : f(x) \geq f(v)\}|$$

Kinnersley [13] showed that for undirected graphs, the vertex separation number equals the pathwidth, hence this notion can be seen as a directed variant of pathwidth.

In [2], Bodlaender, Gustedt and Telle gave the result that this problem is linear time solvable. Now, we give a simpler proof of that fact, similar to the proof of the previous section.

Actually, this can be handled almost identical to the translation of pathwidth to directed cutwidth. Let $G = (V, A)$ be a directed acyclic graph. The weighted directed acyclic graph $G^D = (V^D, A^D)$ is defined as follows:

- G^D has vertices $V^D = \{b_v \mid v \in V\} \cup \{e_v \mid v \in V\}$.
- G^D has two types of arcs. For every $v \in V$, we take an arc (b_v, e_v) of weight 1. For every arc $(v, w) \in A$, we take arcs (b_v, b_w) and (b_w, e_v) of weight 0.

Lemma 14 *Let G be a directed acyclic graph. G has directed vertex separation number at most k, if and only if G^D has weighted directed cutwidth at most $k + 1$.*

Proof: First, suppose we have a topological sort g of G^D of weighted directed cutwidth at most $k + 1$. Then, take the ordering f of G with $f(v) < f(w) \Leftrightarrow g(v) < g(w)$. As for every $(v, w) \in A$, $(b_v, b_w) \in A^D$ hence $b_v < b_w$, this is a topological sort. We claim that this sort has directed vertex separation number at most k. Consider, for a vertex v, the set $J_v = \{w \in V \mid f(w) < f(v) \wedge \exists(w, x) \in A : f(x) \geq f(v)\}$. If $w \in J_v$, then $b_w < b_v$, and there is a vertex x with $\{w, x\} \in A$ and $f(v) \leq f(x)$. Hence, $b_w < b_v \leq b_x < e_w$, so the arc (b_w, e_w) 'crosses' the gap after b_v. As (b_v, e_v) also crosses this gap, there can be at most k vertices in J_v.

Now, suppose we have a topological sort f of G of directed vertex separation number at most k. Make a sort of G^D in the following way: first sort the vertices b_v, $v \in V$ in the same order as the corresponding vertices are ordered in f. Now, for every vertex $w \in V$, look to the highest numbered vertex v with $(v, w) \in A$, and insert e_w in the ordering after

b_v and before the next vertex of the form b_x. The number of arcs crossing a gap is bounded by $k+1$. Consider a gap, and suppose v is the last vertex such that b_v is before this gap. Then, the gap can be crossed by arc (b_v, e_v), and arcs (b_w, e_w) with $w \in J_v$, but not by other arcs of weight one. $\square$

Thus, this lemma shows that we can translate the directed vertex separation number problem to a directed weighted cutwidth problem, and hence have a constructive proof of the following result.

Theorem 15 *(See [2].) For each k, there is a linear time algorithm, that given a directed acyclic graph G, decides if the directed vertex separation number of G is at most k, and if so, finds a topological sort of G with directed vertex separation number at most k.*

4.6 Modified cutwidth

In [12], it is shown that the modified cutwidth problem can be transformed to the Gate Matrix Layout problem, which on its turn can be transformed to the Pathwidth problem (see [11, 16].) Below, we describe the resulting transformation from modified cutwidth to pathwidth.

Let $G = (V, E)$ be a graph, and k be an integer. First, we add to each vertex $v \in V$, $2k + 2 - d$ self loops (edges from v to v.) Let $G' = (V, E')$ be the resulting graph with self loops. Let $H = (E, F)$ be the edge graph of G', i.e., the vertex set of H is the edge set of G', and we have for each pair $e, e' \in E$: $\{e, e'\} \in F$, if and only if $e \neq e'$ and e and e' have at least one endpoint in common.

Lemma 16 *The modified cutwidth of G is at most k, if and only if the pathwidth of H is at most $3k + 1$.*

Proof: Write $n = |V|$. Suppose we have a linear ordering f of G of modified cutwidth at most k. Take the path decomposition $(X_1, \ldots, X_n)$ with for all i, $1 \leq i \leq n$, X_i the set of vertices in H representing edges in G' that either that $f^{-1}(i)$ as an endpoint, or have endpoints $f^{-1}(j_1)$ and $f^{-1}(j_2)$ with $j_1 < i < j_2$ or $j_2 < i < j_1$. As there are precisely $2k+2$ edges of the former type by construction, and at most k edges of the latter type (as f has modified cutwidth at most k), for each i, $1 \leq i \leq n$, $|X_i| \leq 3k + 2$.

Let $(X_1, \ldots, X_r)$ be a path decomposition of H of width at most $3k + 1$. For each $v \in V$, let E'_v be the set of edges in G' that have v as endpoint. E'_v forms a clique in H. By a well known property of path decompositions, there is an $i_v \in I$ with $E'_v \subseteq X_{i_v}$. If $v \neq w$, then $i_v \neq i_w$ as $|E_v \cup E_w| \geq 4k + 3$. Let f be the linear ordering of G with for all v, $w \in V$: $f(v) < f(w)$, if and only if $i_v < i_w$. We claim that the modified cutwidth of f is at most k. Consider a vertex $u \in V$, and look to the edges $e = \{v, w\}$ with $f(v) < f(u) < f(w)$. We have $i_v < i_u < i_w$, and $e \in X_{i_v}$, $e \in X_{i_w}$, hence $e \in X_{i_u}$. As X_{i_u} contains $2k + 2$ vertices representing an edge with u as endpoint, there can be at most k edges with $f(v) < f(u) < f(w)$. $\square$

As the proof also shows how to translate a path decomposition of H of width at most $3k+2$ to a linear ordering of G of modified cutwidth at most k, we have a constructive proof of the following theorem.

Theorem 17 *For each k, there is a linear time algorithm, that given a graph G, decides if the modified cutwidth of G is at most k, and if so, finds a linear ordering of G with modified cutwidth at most k.*

4.7 Directed modified cutwidth

Similar to the directed cutwidth, the directed modified cutwidth of a directed acyclic graph $G = (V, A)$ is the minimum directed modified cutwidth of a topological sort f of G. We have the following result.

Theorem 18 *For each k, there is a linear time algorithm, that given a directed acyclic graph G, decides if the directed modified cutwidth of G is at most k, and if so, finds a topological sort of G with directed modified cutwidth at most k.*

The result follows from a transformation from directed modified cutwidth to directed weighted cutwidth.

Let $G = (V, A)$ be a directed acyclic graph. The weighted directed acyclic graph $G^Q = (V^Q, A^Q)$ is defined as follows.

- $V^Q = \{b_v \mid v \in V\} \cup \{e_v \mid v \in V\}$.
- G^Q has two types of arcs. For every $v \in V$, we take an arc (b_v, e_v) of weight $k+1$. For every arc $(v, w) \in A$, we take an arc (e_v, b_v) of weight 1.

Lemma 19 *G has directed modified cutwidth at most k, if and only if G^Q has weighted directed cutwidth at most $2k+1$.*

Proof: Suppose f is a topological sort of G of modified cutwidth at most k. Set for all $v \in V$, $g(b_v) = 2f(b_v) - 1$ and $g(e_v) = 2f(e_v)$. g is a topological sort of G^Q of weighted cutwidth at most $2k+1$.

Suppose g is a topological sort of G^Q of weighted cutwidth at most $2k+1$. For $v, w \in V$, $v \neq w$, we have $[g(b_v), g(e_v)] \cap [g(b_w), g(e_w)] = \emptyset$, as we would otherwise have a contradiction with the weighted cutwidth of g. So, for all $v \in V$, $g(e_v) = g(b_v) + 1$, and we can set $f(v) = g(e_v)/2$. For each arc $(v, w) \in A$, $g(e_v) < g(b_w) < g(e_w)$, hence $f(v) < f(w)$, so f is a topological sort of G. Consider a vertex u. For each arc $(v, w) \in A$ with $f(v) < f(u) < f(w)$, the arc (e_v, b_w) crosses the spot between b_u and e_u. As the arc (b_u, e_u) also crosses this spot and has weight $k+1$, there can be at most k arcs $(v, w) \in A$ with $f(v) < f(u) < f(w)$. $\square$

Theorem 18 now follows from Lemma 19 and Theorem 10.

5 Conclusions

The techniques described here only deal with problems where a linear order has to be found, and as common characteristic we have that yes-instances have bounded pathwidth. Similar algorithms are known however for graphs of bounded treewidth (like branchwidth [8], carving width [20] and treewidth itself [15, 7]). In order to be able to present or extend such algorithms like we did above, additional techniques have to be added to the machinery. In particular, there are two additional complications that must be mastered:

- The desired output of the problem has a tree structure. Basically, one should show that certain parts of the tree (tree-decomposition or branch-decomposition) are 'not interested' and can be forgotten in the decision algorithm, and that the remainder of the tree then can be formed by gluing a constant number of paths together.

- The input has bounded treewidth, thus a nice tree decomposition can be formed. We now have, in addition to the introduce and forget operations, a join operation, where two partial solutions of two subgraphs have to be combined, basically by 'interleaving' these two.

The constant factors of the algorithms resulting from the methodology presented in this paper are very large. To improve upon these factors, especially as much that the resulting algorithms have implementations that run fast enough for moderate values of k is a remaining challenge.

References

[1] Abrahamson, K. R., and Fellows, M. R. Finite automata, bounded treewidth and well-quasiordering. In *Proceedings of the AMS Summer Workshop on Graph Minors, Graph Structure Theory, Contemporary Mathematics vol. 147* (1993), N. Robertson and P. Seymour, Eds., American Mathematical Society, pp. 539–564.

[2] Bodlaender, H., Gustedt, J., and Telle, J. A. Linear-time register allocation for a fixed number of registers. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA, 1998)* (New York, 1998), ACM, pp. 574–583.

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

[4] Bodlaender, H. L. Treewidth: Algorithmic techniques and results. In *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science, MFCS'97, Lecture Notes in Computer Science, volume 1295* (Berlin, 1997), I. Privara and P. Ruzicka, Eds., Springer-Verlag, pp. 19–36.

[5] Bodlaender, H. L. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sc. 209* (1998), 1–45.

[6] Bodlaender, H. L., Fellows, M. R., and Evans, P. A. Finite-state computability of annotations of strings and trees. In *Proc. Conference on Pattern Matching* (1996), pp. 384–391.

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

[8] Bodlaender, H. L., and Thilikos, D. M. Constructive linear time algorithms for branchwidth. In *Proceedings 24th International Colloquium on Automata, Languages, and Programming* (1997), P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, Eds., Springer Verlag, Lecture Notes in Computer Science, vol. 1256, pp. 627–637.

[9] Chen, M.-H., and Lee, S.-L. Linear time algorithms for k-cutwidth problem. In *Proceedings Third International Symposium on Algorithms and Computation, ISAAC'92* (Berlin, 1992), Springer Verlag, Lecture Notes in Computer Science, vol. 650, pp. 21–30.

[10] Downey, R. G., and Fellows, M. R. Fixed-parameter tractability and completeness I: Basic results. *SIAM J. Comput. 24* (1995), 873–921.

[11] Fellows, M. R., and Langston, M. A. An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science* (1989), pp. 520–525.

[12] Fellows, M. R., and Langston, M. A. On well-partial-order theory and its application to combinatorial problems of VLSI design. *SIAM J. Disc. Math. 5* (1992), 117–126.

[13] Kinnersley, N. G. The vertex separation number of a graph equals its path width. *Information Processing Letters 42* (1992), 345–350.

[14] Kinnersley, N. G., and Kinnersley, W. M. Tree automata for cutwidth recognition. *Congressus Numerantium 104* (1994), 129–142.

[15] Lagergren, J., and Arnborg, S. Finding minimal forbidden minors using a finite congruence. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming* (1991), Springer Verlag, Lecture Notes in Computer Science, vol. 510, pp. 532–543.

[16] Möhring, R. H. Graph problems related to gate matrix layout and PLA folding. In *Computational Graph Theory, Comuting Suppl. 7* (1990), E. Mayr, H. Noltemeier, and M. Sysło, Eds., Springer Verlag, pp. 17–51.

[17] MONIEN, B., AND SUDBOROUGH, I. H. Min cut is NP-complete for edge weighted trees. *Theor. Comp. Sc. 58* (1988), 209–229.

[18] ROBERTSON, N., AND SEYMOUR, P. D. Graph minors. I. Excluding a forest. *J. Comb. Theory Series B 35* (1983), 39–61.

[19] THILIKOS, D. M., SERNA, M. J., AND BODLAENDER, H. L. A constructive linear time algorithm for small cutwidth. Tech. Rep. LSI-00-48-R, Departament de Llenguatges i Sistemes Informatics, Universitat Politecnica de Catalunya, Barcelona, Spain, 2000.

[20] THILIKOS, D. M., SERNA, M. J., AND BODLAENDER, H. L. Constructive linear time algorithms for small cutwidth and carving-width. In *Proc. 11th International Symposium on Algorithms And Computation ISAAC '00, Lecture Notes in Computer Science 1969* (2000), D. Lee and S.-H. Teng, Eds., Springer-Verlag, pp. 192–203.