Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Algorithms 56 (2005) 25–49

Journal of Algorithms

www.elsevier.com/locate/jalgor

# Cutwidth II: Algorithms for partial $w$-trees of bounded degree [☆,☆☆]

Dimitrios M. Thilikos [a,*], Maria Serna [a], Hans L. Bodlaender [b]

[a] *Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Campus Nord – Edifici Ω, c/Jordi Girona Salgado 1-3, 08034 Barcelona, Spain*

[b] *Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands*

Received 7 February 2001

Available online 21 February 2005

## Abstract

The *cutwidth* of a graph $G$ is defined to be the smallest integer $k$ such that the vertices of $G$ can be arranged in a vertex ordering $[v_1, \ldots, v_n]$ in a way that, for every $i = 1, \ldots, n-1$, there are at most $k$ edges with one endpoint in $\{v_1, \ldots, v_i\}$ and the other in $\{v_{i+1}, \ldots, v_n\}$. We examine the problem of computing in polynomial time the cutwidth of a partial $w$-tree with bounded degree. In particular, we show how to construct an algorithm that, in $n^{O(w^2 d)}$ steps, computes the cutwidth of any partial $w$-tree with vertices of degree bounded by a fixed constant $d$. Our algorithm is constructive in the sense that it can be adapted to output a corresponding optimal vertex ordering. Also, it is the main subroutine of an algorithm computing the pathwidth of a bounded degree partial $w$-tree with the same time complexity.

*Keywords:* Cutwidth; Treewidth; Pathwidth; Graph layout

DOI of related article: 10.1016/j.jalgor.2004.12.001.

☆ The work of all the authors was supported by the IST Program of the EU under contract number IST-99-14186 (ALCOM-FT). The research of the first 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).

☆☆ This paper is the full version of the paper titled “A polynomial time algorithm for the cutwidth of bounded degree graphs with small treewidth” which appeared in the proceedings of ESA 2001.

* Corresponding author.

*E-mail addresses:* sedthilk@lsi.upc.edu (D.M. Thilikos), mjserna@lsi.upc.edu (M. Serna), hansb@cs.uu.nl (H.L. Bodlaender).

0196-6774/$ – see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jalgor.2004.12.003

## 1. Introduction

A wide variety of optimization problems can be formulated using vertex ordering problems. In many cases, such a problem asks for the optimal value of some function defined over all the linear orderings of the vertices or the edges of a graph (for a survey, see [19]). One of the most known problems of this type is the problem of computing the *cutwidth* of a graph. It is also known as the MINIMUM CUT LINEAR ARRANGEMENT problem and has several applications such as VLSI design [1,2,16,32], network reliability [29], automatic graph drawing [36,42], and information retrieval [13].

Briefly, the *cutwidth* of a graph $G = (V(G), E(G))$ is equal to the minimum $k$ for which there exists a vertex ordering of $G$ such that for any "gap" (place between two successive vertices) of the ordering, there are at most $k$ edges crossing the gap. Cutwidth has been extensively examined from both algorithmic [14,16,22,23,30,35,48] and graph theoretic point of view [8,14–16,30,33]. Computing cutwidth is an NP-complete problem [25,26] and it remains NP-complete even if the input is restricted to planar graphs with maximum degree 3 [35] (see also [33]). There is a polynomial time approximation algorithm with a ratio of $O(\log|V(G)|\log\log|V(G)|)$ [21] and there is a polynomial time approximation scheme if $|E(G)| = \Theta(|V(G)|^2)$ [3]. Results for different models of random graphs can be found in [17,18,20]. Relatively few work has been done on detecting special graph classes where computing cutwidth can be done in polynomial time. In [16], an algorithm was given that computes the cutwidth of any tree with maximum degree bounded by $d$ in $O(n(\log n)^{d-2})$ time. This result was improved in 1983 by Yannakakis [48], who presented an $O(n\log n)$ algorithm computing the cutwidth of any tree. Since then, the only others polynomial algorithms reported for the cutwidth of graph classes different than trees, concerned special cases such as hypercubes [28] and $b$-dimensional $c$-ary cliques [37]. In this paper, we move one step further presenting a polynomial time algorithm for the cutwidth of bounded degree graphs with small treewidth.

The notions of *treewidth* and *pathwidth* appear to play a central role in many areas of graph theory. Roughly, a graph has small treewidth if it can be constructed by assembling small graphs together in a tree structure, namely a tree decomposition of small width (graphs with treewidth at most $w$ are alternatively called *partial $w$-trees*—see Section 2 for the formal definitions). A big variety of graph classes appear to have small treewidth, such as trees, outerplanar graphs, series parallel graphs, and Halin graphs (for a detailed survey of classes with bounded treewidth, see [7]). The pathwidth of a graph is defined similarly to treewidth, but now the tree in its definition is required to be a simple line (path). That way, treewidth can be seen as a "tree"-generalization of pathwidth. Pathwidth and treewidth were introduced by Robertson and Seymour in [39,40] and served as some of the cornerstones of their lengthy proof of the Wagner conjecture, known now as the Graph Minors Theorem (for a survey see [41]). Treewidth appears to have interesting applications in algorithmic graph theory. In particular, a wide range of, otherwise intractable, combinatorial problems are polynomially, even linearly, solvable when restricted to graphs with bounded treewidth or pathwidth. In this direction, numerous techniques have been developed in order to construct dynamic programming algorithms making use of the "tree" or "line" structure of the input graph (see, e.g., [6]). The results of this paper show how these techniques can be used for constructing a polynomial time algorithm for the cutwidth of partial $w$-trees with vertices of degrees bounded by fixed constants.

This paper is strongly based on the previous one of this series [46] where there is a linear time algorithm that for any fixed $k$ decides whether an input graph $G$ has cutwidth at most $k$ and, if so, constructs the corresponding linear layout. Most of the basic tools used in this paper are defined and developed in [46]. We warn the reader that the understanding of this paper is not possible if he/she is not familiar with the notation, concepts, and results used and/or introduced in [46] such as typical sequences, characteristics, sets of characteristics, and the basic algorithms Introduce-Node and Forget-Node.

At this point, we want to give an informal description of our algorithm. Our algorithm starts computing a "nice" tree decomposition of bounded width of the input graph $G$ (for a formal definition see Section 2). The tree decomposition allows the definition of an appropriate subgraph associated to each node, in such a way that the root has associated the graph $G$. The algorithm proceeds from bottom to top of the tree decomposition. First, consider the following exponential algorithm, that builds the set of all layouts of cutwidth at most $k$. The algorithm goes through the tree decomposition in a bottom–up direction, having at each point a data structure of the set of all layouts of cutwidth at most $k$ of the subgraph induced by the vertices encountered so far. When a new vertex is encountered, we try to insert it at all different spots in all the layouts in the data structure; from the layouts we thus obtain, we keep those whose cutwidth is at most $k$. When we arrive to a tree branching the two sets of layouts have to be merged, the data structure will keep all those layouts with cutwidth at most $k$. Clearly, this algorithm can use exponential time, but solves the problem: $G$ has cutwidth at most $k$, if and only if the data structure contains at least one layout when the whole tree decomposition has been processed. As in [46], we fasten this possibly exponential time algorithm by not keeping all layouts, but only the characteristics of the layouts. Layouts with the same characteristic thus are treated only once.

In this paper we design a polynomial time algorithm for computing a vertex ordering with minimum cutwidth when the input graph is a partial $w$-trees of bounded degree. This is possible due to the observation that the "hidden constants" of all the subroutines used by our algorithm remain polynomial even when we ask whether $G$ has cutwidth at most $O(dk) \cdot \log n$. As this upper bound for cutwidth is indeed satisfied (see Corollary 4), our algorithm is able to compute in $n^{O(w^2 d)}$ steps the cutwidth of bounded degree partial $w$-trees. The proposed algorithm extends the algorithm in [46] in the sense that it uses all of its subroutines and it solves the problem of [46] for graphs with bounded treewidth. The main technical contribution of this paper is an algorithm (algorithm Join-Node in Section 4) that performs the merging of the representatives at a branching of the tree decomposition. This algorithm uses the "small treewidth" property of the input graph.

An interesting consequence of our result is that the pathwidth of bounded degree partial $w$-trees can be computed in $n^{O(w^2 d)}$ steps. We mention that the existence of a polynomial time algorithm for this problem, without the degree restriction, has been proved in [11]. However, the time complexity of the involved algorithm appears to be very large and has not been reported. Our technique, described in Section 6, reduces the computation of pathwidth to the problem of computing the cutwidth on hypergraphs. Then the pathwidth is computed using a generalization of our algorithm for hypergraphs with bounded treewidth. That way, we report more reasonable time bounds, provided that the input graph has bounded degree.

The paper is organized as follows. Section 2 contains the definitions of treewidth, pathwidth and cutwidth. Sections 3 and 4 concern operations on "interleavings" of sequences of integers and are building on the definitions and results of Section 2 of [46]. These new concepts will be the main ingredient for the algorithm Join-Node that is presented in Section 4. The algorithm Join-Node only helps to compute the cutwidth of a bounded degree partial $w$-tree $G$ but not to *construct* the corresponding vertex ordering. In Section 5, we describe how to transform this algorithm to a *constructive* one in the sense that we now can output a linear arrangement of $G$ with optimal cutwidth. This uses the analogous constructions of [46] and the procedures Join-Orderings and Construct-Join-Orderings described in Section 4.

## 2. Treewidth—pathwidth—cutwidth

Although we assume the reader familiar with the notation of [46] we repeat here some basic items. All the graphs of this paper are finite, undirected, and without loops or multiple edges (our results can be straightforwardly generalized to the case where the last restriction is altered). We denote the vertex (edge) set of a graph $G$ by $V(G)$ ($E(G)$) and set $n = |V(G)|$. As our graphs will not have multiple edges, we represent an edge $e \in E(G)$ by a two vertex subset. A linear (one-dimensional) layout of the vertices of $G$ is a bijection, mapping $V(G)$ to the integers in $\{1, \ldots, n\}$. We denote such a layout by the sequence $[v_1, \ldots, v_n]$.

We deal with finite sequences (i.e., ordered sets) of a given finite set $\mathcal{O}$. And denote by $\mathcal{S}$ the set of all finite sequences of non-negative integers.

A *tree decomposition* of a graph $G$ is a pair $(X, U)$ where $U$ is a tree whose vertices we will call *nodes* and $X = (\{X_i \mid i \in V(U)\})$ is a collection of subsets of $V(G)$ such that

(1) $\bigcup_{i \in V(U)} X_i = V(G)$,
(2) for each edge $\{v, w\} \in E(G)$, there is an $i \in V(U)$ such that $v, w \in X_i$, and
(3) for each $v \in V(G)$ the set of nodes $\{i \mid v \in X_i\}$ forms a subtree of $U$.

The *width* of a tree decomposition $(\{X_i \mid i \in V(U)\}, U)$ equals $\max_{i \in V(U)}\{|X_i| - 1\}$. The *treewidth* of a graph $G$ is the minimum width over all tree decompositions of $G$.

A *rooted* tree decomposition is a triple $D = (X, U, r)$ in which $U$ is a tree rooted at $r$ and $(X, U)$ is a tree decomposition.

Let $D = (X, U, r)$ be a rooted tree decomposition of a graph $G$ where $X = \{X_i \mid i \in V(U)\}$. $D$ is called a *nice* tree decomposition if the following are satisfied

(1) every node of $U$ has at most two children,
(2) if a node $i$ has two children $j, h$ then $X_i = X_j = X_h$,
(3) if a node $i$ has one child $j$, then either $|X_i| = |X_j| + 1$ and $X_j \subset X_i$ or $|X_i| = |X_j| - 1$ and $X_i \subset X_j$.

Notice that a nice tree decomposition is always a rooted tree decomposition. For the following, see, e.g., [11].

**Lemma 1.** *For any constant $k \geqslant 1$, given a tree decomposition of a graph $G$ of width $\leqslant k$, there exists an algorithm that, in $O(n)$ steps, constructs a nice tree decomposition of $G$ of width $\leqslant k$ and with at most $4n$ nodes.*

We now observe that a nice tree decomposition $(\{X_i \mid i \in V(U)\}, U, r)$ contains nodes of the following four possible types. A node $i \in V(U)$ is called "*start*" if $i$ is a leaf of $U$, "*join*" if $i$ has two children, "*forget*" if $i$ has only one child $j$ and $|X_i| < |X_j|$, "*introduce*" if $i$ has only one child $j$ and $|X_i| > |X_j|$. We may also assume that if $i$ is a *start* node then $|X_i| = 1$: the effect of *start* nodes with $|X_i| > 1$ can be obtained by using a *start* node with a set containing 1 vertex, and then $|X_i| - 1$ *introduce* nodes, which add all the other vertices.

Let $D = (X, U, r)$ be a nice tree decomposition of a graph $G$. For each node $i$ of $U$, let $U_i$ be the subtree of $U$, rooted at node $i$. For any $i \in V(U)$, we set $V_i = \bigcup_{v \in V(U_i)} X_v$ and $G_i = G[V_i]$. For any $p \in V(U)$ we refine $G_p$ in a top down fashion as follows. If $q$ is a *join* with children $p$ and $p'$, select one of its two children, say $p$. Then, for any $i \in U_p$ remove from $G_i$ any edge in the set $E(G_q[X_q])$ (in fact, any partition of $E(G_q[X_q])$ for the edges induced by $G_p[X_p]$ and $G_{p'}[X_{p'}]$ would be suitable for the purposes of this paper). In this construction, we have $V(G_p) = V_p$ for any $p \in V(U)$ and we guarantee that if $q$ is a *join* node with children $p$ and $p'$ then $V(G_p) = V(G_{p'}) = V(G_q)$, $E(G_p[X_q]) \cap E(G_{p'}[X_q]) = \emptyset$, and $E(G_p) \cup E(G_{p'}) = E(G_q)$. Notice that if $r$ is the root of $U$, then $G_r = G$. We call $G_i$ the subgraph of $G$ *rooted* at $i$. We finally set, for any $i \in V(U)$, $D_i = (X^i, U_i, i)$ where $X^i = \{X_v \mid v \in V(U_i)\}$. Observe that for each node $i \in V(U)$, $D_i$ is a nice tree decomposition of $G_i$.

A tree decomposition $(X, U)$ is a path decomposition, if $U$ is a path (i.e., tree $U$ has no nodes of degree more than two). The pathwidth of a graph $G$ is defined as the minimum width of a path decomposition of $G$.

We will need the following result given in [10].

**Lemma 2.** *For any graph $G$,*

$$\text{treewidth}(G) \leqslant \text{pathwidth}(G) \leqslant \bigl(\text{treewidth}(G) + 1\bigr) \cdot \log(n).$$

The cutwidth of a graph $G$ with $n$ vertices is defined as follows. Let $l = [v_1, \dots, v_n]$ be a layout of $V(G)$. For $i = 1, \dots, n-1$, we define the *cut at position $i$*, denoted by $\theta_{l,G}(i)$, as the set of crossover edges of $G$ that have one endpoint in $l[1,i]$ and one in $l[i+1,n]$, i.e., $\theta_{l,G}(i) = E_G(l[1,i]) \cap E_G(l[i+1,n])$. The *cutwidth of a layout $l$* of $V(G)$ is $\max_{i, 1 \leqslant i \leqslant n-1}\{|\theta_{l,G}(i)|\}$. The *cutwidth* of a graph is the minimum cutwidth over all the vertex orderings of $V(G)$.

It is easy to see the following (see also [16]).

**Lemma 3.** *For any graph $G$, with vertices of degree bounded by $d$,* $\text{pathwidth}(G) \leqslant \text{cutwidth}(G) \leqslant d \cdot \text{pathwidth}(G)$.

The following is a direct consequence of Lemmata 2 and 3.

**Corollary 4.** *Any graph $G$ with treewidth at most $w$ and maximum degree at most $d$, has cutwidth bounded by $(w+1)d\log n$.*

## 3. Interleavings of sequences

Building on Section 2.3 of [46], we introduce some further notation on interleavings of sequences of non-negative numbers. This new operation will be need to deal with join nodes. We assume that the reader knows the definitions of functions $\mathcal{E}$, $\tau$, relations "$\leqslant$", "$\sqsubseteq$", "$\prec$", and the concept of *typical* and *dense* sequence defined in [46].

Let $A$ and $B$ two equal-size sequences of non-negative integers where $A=[a_1,\ldots,a_r]$, $B=[b_1,\ldots,b_r]$. We define $A+B=[a_1+b_1,\ldots,a_r+b_r]$ and we say that $A\sim B$ iff $\forall_{1\leqslant i<r}\ a_i\neq a_{i+1}\Leftrightarrow b_i=b_{i+1}$ (and, therefore, $b_i\neq b_{i+1}\Leftrightarrow a_i=a_{i+1}$). As an example, we mention that

$$[1,1,8,5,5,6,7]\sim[3,6,6,6,9,9,9].$$

The *interleaving* $A\otimes B$ of two typical sequences $A$ and $B$ is a set of typical sequences defined as follows

$$A\otimes B=\bigl\{\tau\bigl(\tilde{A}+\widetilde{B}\bigr)\mid \tilde{A}\in\mathcal{E}(A),\ \widetilde{B}\in\mathcal{E}(B)\text{ and }\tilde{A}\sim\widetilde{B}\bigr\}.$$

We stress out that the above definition of interleaving is an important ingredient for the algorithm and the proofs of this paper (a similar, but simpler, definition of "interleaving" was used in [11]). Notice that the length of the resulting sequences is at most $|A|+|B|-1$.

For an example for the case where $A=[1,2,1,3]$ and $B=[4,3,6,3]$, see Fig. 1.

A useful geometrical interpretation of the operation $A\otimes B$ where $p=|A|$ and $q=|B|$, can be given by the labelled $(p\times q)$-grid

$$\begin{aligned}F_{p,q}=\bigl(&\bigl\{(i,j)\mid 1\leqslant i\leqslant p,\ 1\leqslant j\leqslant q\bigr\},\\ &\bigl\{\bigl((i,j),\bigl(i',j'\bigr)\bigr)\mid i\leqslant i',\ j\leqslant j',\ i'-i+j'-j=1\bigr\}\bigr)\end{aligned}$$

whose vertex $(i,j)$ is labelled by the sum $A(i)+B(j)$. Notice that there exists a one to one correspondence between the labels of the paths connecting vertices $(1,1)$ and $(p,q)$ in $F_{p,q}$ and the sums $\tilde{A}+\widetilde{B}$ where $\tilde{A}\in\mathcal{E}(A)$, $\widetilde{B}\in\mathcal{E}(B)$ and, $\tilde{A}\sim\widetilde{B}$. For an example, if $A=[1,2,1,3]$ and $B=[4,3,6,3]$, then the sum $\tilde{A}+\widetilde{B}$ where $\tilde{A}=[\mathbf{1},2,2,\mathbf{2},2,\mathbf{1},\mathbf{3}]$ and $\widetilde{B}=[\mathbf{4},4,3,\mathbf{6},3,\mathbf{3},\mathbf{3}]$ corresponds to the path $((1,1),(2,1),(2,2),(2,3),(2,4),(3,4),(4,4))$ (see Fig. 2). This observation is used in the proof the following lemma.

**Lemma 5.** *Let $A$ and $B$ be typical sequences of integers in $\{0,k\}$. Then, $|A\otimes B|\leqslant 2^{4(k-1)}$ and $A\otimes B$ can be computed in $O(k\cdot 2^{4(k-1)})$ steps.*

**Proof.** From [46, Lemma 2.6] we obtain that $p\leqslant 2k-1$ and $q\leqslant 2k-1$. Let $\mathcal{P}(F_{p,q})$ be the set of directed paths in $F_{p,q}$ connecting $(1,1)$ to $(p,q)$. By labelling each horizontal arrow in $F_{p,q}$ with a 2 and each vertical arrow in $F_{p,q}$ with an 1 we easily derive that there

$$\begin{aligned}
A \otimes B = \big\{ & \tau\big([1,1,1,1,2,1,3]+[4,3,6,3,3,3,3]\big), \tau\big([1,1,1,2,2,1,3]+[4,3,6,6,3,3,3]\big),\\
& \tau\big([1,1,1,2,1,1,3]+[4,3,6,6,6,3,3]\big), \tau\big([1,1,1,2,1,3,3]+[4,3,6,6,6,6,3]\big),\\
& \tau\big([1,1,2,2,2,1,3]+[4,3,3,6,3,3,3]\big), \tau\big([1,1,2,2,1,1,3]+[4,3,3,6,6,3,3]\big),\\
& \tau\big([1,1,2,2,1,3,3]+[4,3,3,6,6,6,3]\big), \tau\big([1,1,2,1,1,1,3]+[4,3,3,3,6,3,3]\big),\\
& \tau\big([1,1,2,1,1,3,3]+[4,3,3,3,6,6,3]\big), \tau\big([1,1,2,1,3,3,3]+[4,3,3,3,3,6,3]\big),\\
& \tau\big([\underline{\mathbf{1},2,2,\mathbf{2},2,\mathbf{1},\mathbf{3}}]+[\underline{\mathbf{4},4,3,\mathbf{6},3,\mathbf{3},\mathbf{3}}]\big), \tau\big([\underline{\mathbf{1},2,2,\mathbf{2},1,\mathbf{1},\mathbf{3}}]+[\underline{\mathbf{4},4,3,\mathbf{6},6,\mathbf{3},\mathbf{3}}]\big),\\
& \tau\big([1,2,2,2,1,3,3]+[4,4,3,6,6,6,3]\big), \tau\big([1,2,2,1,1,1,3]+[4,4,3,3,6,3,3]\big),\\
& \tau\big([1,2,2,1,1,3,3]+[4,4,3,3,6,6,3]\big), \tau\big([1,2,2,1,3,3,3]+[4,4,3,3,3,6,3]\big),\\
& \tau\big([1,2,1,1,1,1,3]+[4,4,4,3,6,3,3]\big), \tau\big([1,2,1,1,1,3,3]+[4,4,4,3,6,6,3]\big),\\
& \tau\big([1,2,1,1,3,3,3]+[4,4,4,3,3,6,3]\big), \tau\big([1,2,1,3,3,3,3]+[4,4,4,4,3,6,3,]\big)\big\}\\
= \big\{ & \tau\big([5,4,7,4,5,4,6]\big), \tau\big([5,4,7,8,5,4,6]\big), \tau\big([5,4,7,8,7,4,6]\big), \tau\big([5,4,7,8,7,9,6]\big),\\
& \tau\big([5,4,5,8,5,4,6]\big), \tau\big([5,4,5,8,7,4,6]\big), \tau\big([5,4,5,8,7,9,6]\big), \tau\big([5,4,5,4,7,4,6]\big),\\
& \tau\big([5,4,5,5,7,9,6]\big), \tau\big([5,4,5,4,6,9,6]\big), \tau\big([\underline{\mathbf{5},6,5,\mathbf{8},5,\mathbf{4},\mathbf{6}}]\big), \tau\big([\underline{\mathbf{5},6,5,\mathbf{8},7,\mathbf{4},\mathbf{6}}]\big),\\
& \tau\big([5,6,5,8,7,9,6]\big), \tau\big([5,6,5,4,7,4,6]\big), \tau\big([5,6,5,4,7,9,6]\big), \tau\big([5,6,5,4,7,9,6]\big),\\
& \tau\big([5,6,5,4,7,4,6]\big), \tau\big([5,6,5,4,7,9,6]\big), \tau\big([5,5,6,4,6,9,6]\big), \tau\big([5,6,5,7,6,9,6]\big)\big\}\\
= \big\{ & [5,4,7,4,6], [5,4,8,4,6], [5,4,9,6], [\underline{\mathbf{5},\mathbf{8},\mathbf{4},\mathbf{6}}], [5,9,6], [5,6,4,7,4,6], [5,6,4,9,6],\\
& [5,6,4,9,6], [5,6,5,9,6]\big\}
\end{aligned}$$

Fig. 1. An example of the interleaving of the typical sequences $A = [1, 2, 1, 3]$ and $B = [4, 3, 6, 3]$.

exists a bijection between $\mathcal{P}(F_{p,q})$ and the set of all sequences of 1's and 2's that have length $p - 1 + q - 1$ and exactly $p - 1$ 1's. One can easily bound $|\mathcal{P}(F_{p,q})|$ by observing that there are $2^{p-1+p-1}$ sequences consisting of 1's and 2's and with length $p - 1 + p - 1$, thus $|\mathcal{P}(F_{p,q})| < 2^{p-1+p-1} \leqslant 2^{4(k-1)}$. As we mentioned above, the construction of $A \otimes B$ requires the generation of all paths in $\mathcal{P}$, each corresponding to a sum $\tilde{A} + \widetilde{B}$ where $\tilde{A} \in \mathcal{E}(A)$, $\widetilde{B} \in \mathcal{E}(B)$ and, $\tilde{A} \sim \widetilde{B}$. The result follows as each such path can be generated in $O(k)$ steps and the compression of the corresponding sequence $\tilde{A} + \widetilde{B}$ can be done in $O(k)$ steps. $\square$

As we did in [46], we will use bold letters to denote sequences of sequences of integers and we will assume that they are indexed starting from 0.

The *interleaving* of two sequences of typical sequence $\mathbf{A} = [A_0, \ldots, A_w]$ and $\mathbf{B} = [B_0, \ldots, B_w]$ where $w = |\mathbf{A}| = |\mathbf{B}|$ is defined as follows:

$$\mathbf{A} \bigotimes \mathbf{B} = \big\{[C_0, \ldots, C_w] \mid C_i \in A_i \otimes B_i,\ i = 0, \ldots, w\big\}.$$

The following lemma is a direct consequence of Lemma 5 and the definition of $\mathbf{A} \bigotimes \mathbf{B}$.

**Lemma 6.** *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be two sequences of sequences of integers in* $\{0, \ldots, k\}$, *with* $|\mathbf{A}| = |\mathbf{B}| = w + 1$. *Then* $|\mathbf{A} \bigotimes \mathbf{B}| \leqslant 2^{4(k-1)(w+1)}$ *and* $\mathbf{A} \bigotimes \mathbf{B}$ *can be computed in* $O(k \cdot 2^{4(k-1)(w+1)})$ *steps.*

$$\begin{tikzcd}
(1,1) \arrow[r, "1"] \arrow[d, "2"] & (2,1) \arrow[r, "1"] \arrow[d, "2"] & (2,1) \arrow[r, "1"] \arrow[d, "2"] & (4,1) \arrow[d, "2"] \\
(1,2) \arrow[r, "1"] \arrow[d, "2"] & (2,2) \arrow[r, "1"] \arrow[d, "2"] & (3,2) \arrow[r, "1"] \arrow[d, "2"] & (4,2) \arrow[d, "2"] \\
(1,3) \arrow[r, "1"] \arrow[d, "2"] & (2,3) \arrow[r, "1"] \arrow[d, "2"] & (3,3) \arrow[r, "1"] \arrow[d, "2"] & (4,3) \arrow[d, "2"] \\
(1,4) \arrow[r, "1"] & (2,4) \arrow[r, "1"] & (3,4) \arrow[r, "1"] & (4,4)
\end{tikzcd}$$

$$\begin{tikzcd}
1+4=\boxed{5} \arrow[r, Rightarrow] \arrow[d] \arrow[ddr, dotted] & 2+4=6 \arrow[r] \arrow[d, Rightarrow] & 1+4=5 \arrow[r] \arrow[d] & 3+4=7 \arrow[d] \\
1+3=4 \arrow[r] \arrow[d] & 2+3=5 \arrow[r] \arrow[d, Rightarrow] & 1+3=4 \arrow[r] \arrow[d] & 3+3=6 \arrow[d] \\
1+6=7 \arrow[r] \arrow[d] & 2+6=\boxed{8} \arrow[r, Rightarrow] \arrow[d, Rightarrow] \arrow[dr, dotted] & 1+6=7 \arrow[r] \arrow[d, Rightarrow] & 3+6=9 \arrow[d] \\
1+3=4 \arrow[r] & 2+3=5 \arrow[r, Rightarrow] & 1+3=\boxed{4} \arrow[r, Rightarrow] \arrow[r, dotted, bend right] & 3+3=\boxed{6}
\end{tikzcd}$$

Fig. 2. An example of $F_{4,4}$ and its labeling when $A = [1, 2, 1, 3]$ and $B = [4, 3, 6, 3]$. The double-lined arrows define the two paths in that are able to generate the typical sequence $[5, 8, 4, 6]$.

We will now introduce some more notation using the geometrical interpretation of $A \otimes B$. Given two sequences $B_1$ and $B_2$ where $B_1 \sim B_2$ we define function $\nu_{B_1,B_2} : \{1, \ldots, |B_1| - 1\} \to \{1, 2\}$, $\nu_{B_1,B_2}(j) = 1$ if $B_1(j) \neq B_1(j + 1)$ and $\nu(j) = 2$ if $B_1(j) = B_1(j + 1)$ ($\nu_{B_1,B_2}(j)$ indicates which one of $B_1$, $B_2$ changes value between indexes $j$ and $j + 1$). When the sequences $B_1$ and $B_2$ are obvious, we simply denote $\nu_{B_1,B_2}$ by $\nu$.

As an example, we consider $B_1 = [1, 2, 2, 2, 2, 1, 3]$ and $B_2 = [4, 4, 3, 6, 3, 3, 3]$. We have

$$\nu(1) = 1, \quad \nu(2) = 2, \quad \nu(3) = 2, \quad \nu(4) = 2, \quad \nu(5) = 1, \quad \nu(6) = 1.$$

(Notice that in Fig. 2, these values correspond to the labels of the edges in the path $((1, 1), (2, 1), (2, 2), (2, 3), (2, 4), (3, 4), (4, 4))$ of $F_{4,4}$.)

The following lemma will be useful in order to adapt some auxiliary results of [11] to our definition of $A \otimes B$.

**Lemma 7.** *For any two dense sequences $A_1$ and $A_2$ of equal length there exists a typical sequence $S \in \tau(A_1) \otimes \tau(A_2)$ such that $S \prec \tau(A_1 + A_2)$.*

**Proof.** In other words, we have to prove that there exist two sequences $\tilde{A}_1 \in \mathcal{E}(\tau(A_1))$ and $\tilde{A}_2 \in \mathcal{E}(\tau(A_2))$ such that $\tilde{A}_1 \sim \tilde{A}_2$, and $\tau(\tilde{A}_1 + \tilde{A}_2) \prec \tau(A_1 + A_2)$. We set $C = A_1 + A_2$ and $r = |A_1|$. For $i = 1, 2$, we set up a function $t_i : \{1, \ldots, r - 1\} \to \{0, 1\}$ such that, for $j = 1, \ldots, r - 1$, $t_i(j) = i - 1$ if $A_1(j) + A_2(j + 1) \leqslant A_1(j + 1) + A_2(j)$ and $t_i(j) = 2 - i$ otherwise. Notice that, $j = 1, \ldots, r - 1$, $t_1(j) + t_2(j) = 1$. For $i = 1, 2$, we construct a sequence $A_i^*$ from $A_i$ by setting $A_i^*(2j - 1) = A_i(j)$ and $A_i^*(2j) = A_i(j + t_i(j))$, $j = 1, \ldots, |A|$. Notice that the density of $A_i$ and the construction of $A_i^*$ implies that $A_i^*$ is also dense for $i = 1, 2$. Moreover, $A_i \sqsubseteq A_i^*$, $i = 1, 2$, hence $\tau(A_i^*) = \tau(A_i)$, $i = 1, 2$. Let $C^* = A_1^* + A_2^*$ and notice that, by the construction of $A_i^*$ we have that for any $j = 1, \ldots,$ $r - 1$,

$$\begin{aligned}C^*(2j) &= \min\bigl\{A_1(j)+A_2(j+1), A_1(j+1)+A_2(j)\bigr\}\\ &= \min\bigl\{A_1^*(2j-1)+A_2^*(2j+1), A_1^*(2j+1)+A_2^*(2j-1)\bigr\}\\ &\leqslant \max\bigl\{A_1^*(2j-1)+A_2^*(2j-1), A_1^*(2j+1)+A_2^*(2j+1)\bigr\}\\ &= \max\bigl\{A_1(j)+A_2(j), A_1(j+1)+A_2(j+1)\bigr\}\\ &= \max\bigl\{C(j), C(j+1)\bigr\}. \end{aligned}\tag{1}$$

We now define $C^{\text{ext}} = [c_1, c_1', c_2, c_2', \dots, c_{r-1}, c_{r-1}', c_r]$ where $\forall_{j,1\leqslant j\leqslant r}\ c_j = A_1(j) + A_2(j) = C(j) = C^*(2j-1)$ and $\forall_{j,1\leqslant j<r}\ c_j' = \max\{c_j, c_{j+1}\}$. Notice that $C \sqsubseteq C^{\text{ext}}$ and, from (1), $C^{\text{ext}} \leqslant C^*$. Therefore $C^* \prec C$ and thus, $\tau(C^*) \prec \tau(C)$. Notice also that

$$\forall_{i=1,2}\ \forall_{1\leqslant j<r} \quad A_i^*(j) \neq A_i^*(j+1) \quad \Rightarrow \quad A_{3-i}^*(j) \neq A_{3-i}^*(j+1). \tag{2}$$

Apply now the following operation on $A_1^*$ and $A_2^*$ as long as this is possible: if for some $j$, $1 \leqslant j < |A_1^*|$, $A_i^*(j) = A_i^*(j+1)$, $i = 1,2$, then set $A_i^* \leftarrow A_i^*(1,j) \cdot A_i^*(j+2,r)$, $i = 1,2$ (in other words, we apply operation (i) in parallel as long as it removes elements of the same ranks in $A_1^*$ and $A_2^*$). Clearly, (2) is invariant under this operation. Moreover, it returns two sequences $\tilde{A}_i$, $i = 1,2$, where $|\tilde{A}_1| = |\tilde{A}_2|$ and the inverse of (2) holds as well. Therefore $\tilde{A}_1 \sim \tilde{A}_2$. Clearly, $\tilde{A}_i \sqsubseteq A_i^*$, $i = 1,2$, and as $A_i^*$ is dense we get $\tilde{A}_i = \mathcal{E}(\tau(A_i^*))$, $i = 1,2$. Recall that $\tau(A_i^*) = \tau(A_i)$ and therefore $\tilde{A}_i \in \mathcal{E}(\tau(A_i))$, $i = 1,2$. Notice now that $\tilde{A}_1 + \tilde{A}_2 \sqsubseteq A_1^* + A_2^*$. We conclude that $S = \tau(\tilde{A}_1 + \tilde{A}_2) = \tau(C^*)$ and the result follows. □

**Lemma 8.** *Let $A$, $B$ be two typical sequences and let $C$ be a sequence such that $C \in A \otimes B$. Suppose also that $A'$, $B'$ are two typical sequence such that $A \prec A'$ and $B \prec B'$. Then there exists a sequence $C' \in A' \otimes B'$ such that $C' \prec C$.*

**Proof.** As $C \in A \otimes B$, there exist three integer sequences of equal length $Y$, $A^*$ and $B^*$ such that $C = \tau(Y)$, $A^* = \mathcal{E}(A)$, $B^* = \mathcal{E}(B)$, $Y = A^* + B^*$ and $A^* \sim B^*$. From $A^* = \mathcal{E}(A)$, we have that $\tau(A^*) = \tau(A)$ and, as $A' \prec A$, [46, Lemma 2.8] implies that $A' \prec A^*$. Similarly, we get that $B' \prec B^*$. From [46, Lemma 2.9], there exist sequences $A'^* \in \mathcal{E}(A')$ and $B'^* \in \mathcal{E}(B')$ such that $\tau(A'^* + B'^*) \prec \tau(Y) = C$. We set $C' = \tau(A'^* + B'^*)$. Notice now that $A'^*$ and $B'^*$ are both dense. Therefore, Lemma 7 can be applied and if $C' = \tau(A'^* + B'^*)$ there exists an $S \in \tau(A'^*) \otimes \tau(B'^*) = \tau(A') \otimes \tau(B')$ such that $C' \prec S$. □

The following lemma is a direct consequence of Lemma 8.

**Lemma 9.** *Let $\mathbf{A}$, $\mathbf{B}$ be two sequences of typical sequences where $|\mathbf{A}| = |\mathbf{B}|$ and $\mathbf{C}$ a sequence of typical sequences such that $\mathbf{C} \in \mathbf{A} \bigotimes \mathbf{B}$. Suppose also that $\mathbf{A}'$, $\mathbf{B}'$ are two typical sequence such that $\mathbf{A} \prec \mathbf{A}'$ and $\mathbf{B} \prec \mathbf{B}'$. Then there exists a sequence $\mathbf{C}' \in \mathbf{A}' \otimes \mathbf{B}'$ such that $\mathbf{C} \prec \mathbf{C}'$.*

The following lemma is just a special case of Lemma 3.14 in [11].

**Lemma 10.** *Let $A$ and $B$ be two integer sequences with the same length. Then there exists two equal length integer sequences $A' \in \mathcal{E}(\tau(A))$, and $B' \in \mathcal{E}(\tau(B))$, where $A' + B' \prec A + B$.*

**Lemma 11.** *Let $A, B, C$ be sequences such that $|A| = |B|$ and $C = A + B$. Then there exists a sequence $C' \in \tau(A) \otimes \tau(B)$ such that $\tau(C') \prec \tau(C)$.*

**Proof.** From Lemma 10 and [46, Lemma 2.8], there exist two equal length integer sequences $A' \in \mathcal{E}(\tau(A))$ and $B' \in \mathcal{E}(\tau(B))$ such that $\tau(A' + B') \prec \tau(A + B) = \tau(C)$. Notice now that $A'$ and $B'$ are dense, and, from Lemma 7, there exists a typical sequence $C' \in \tau(A') \otimes \tau(B')$ such that $C' \prec \tau(A' + B')$ and the result follows. □

## 4. A decision algorithm for cutwidth

In this section, we give for any pair of integer constants $k$, $w$, an algorithm that, given a graph $G$ with maximum degree $d$ and a nice tree decomposition $(X, U)$ of width at most $w$, decides whether $G$ has cutwidth at most $k$. We assume that the reader is familiar with the concept of an $X$-characteristic of a layout as well as the sequence of sequences $\mathbf{Q}_{G,l}$ defined in Section 2.2 of [46] (for an example of the sequence $\mathbf{Q}_{G,l}$ see Fig. 3).

Assume from now on that we have a graph $G$ and that $(X, U)$ is a nice tree decomposition of $G$, with width at most $w$. A set $FS(i)$ of $X_i$-characteristics of vertex orderings of the graph $G_i$ with cutwidth at most $k$ is called a *full set of characteristics for node $i$* if for each vertex ordering $l$ of $G_i$ with cutwidth at most $k$, there is a vertex ordering $l'$ of $G_i$ such that $C_{X_i}(G_i, l') \prec C_{X_i}(G_i, l)$ and $C_{X_i}(G_i, l') \in FS(i)$, i.e., the $X_i$-characteristic of $l'$ is in $FS(i)$.

The following bound on the number of characteristics of a vertex ordering correspond to Lemma 3.1 of [46].

**Lemma 12.** *Let $G$ be a graph and let $(X, U)$ be a nice tree decomposition of $G$ with width at most $w$. Let $X_i$, $i \in V(U)$, be some node of $(X, U)$. The number of different $X_i$-characteristics of all possible vertex orderings of $G_i$ with cutwidth at most $k$, is bounded by $(w + 1)! \cdot (\frac{8}{3}2^{2k})^{w+1}$.*

The following lemma, that can be derived directly from the definitions, establishes the validity of the concept of full set of characteristics for checking the cutwidth of a graph.

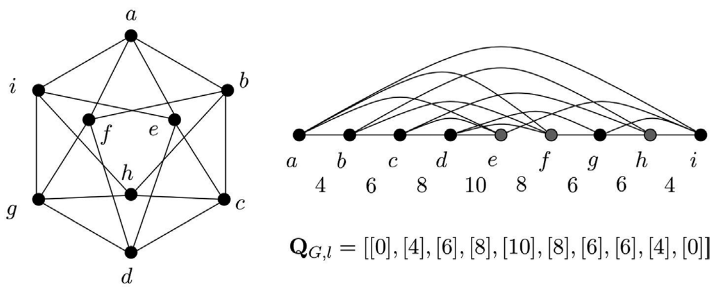
a
b
c
d
e
f
g
h
i
a b c d e f g h i
4 6 8 10 8 6 6 4
$\mathbf{Q}_{G,l} = [[0], [4], [6], [8], [10], [8], [6], [6], [4], [0]]$

Fig. 3. A graph $G$, a vertex ordering $l$ of $G$, and the sequence of sequences $\mathbf{Q}_{G,l}$.

```
Algorithm Join-Node.

Input: A full set of characteristics $FS(j_1)$ for $j_1$ and
       a full set of characteristics $FS(j_2)$ for $j_2$.
Output: A full set of characteristics $FS(i)$ for $i$.

1: Initialize $FS(i) = \emptyset$.
2: For any pair of $X_{j_h}$-characteristics $(\lambda, \mathbf{A}_h) \in FS(j_h)$, $h = 1, 2$, do
3: For any $\mathbf{A} \in \mathbf{A}_1 \bigotimes \mathbf{A}_2$, do
4: If $\max(\lambda, \mathbf{A}) \leqslant k$, set $FS(i) \leftarrow FS(i) \cup \{(\lambda, \mathbf{A})\}$.
5: Output $FS(i)$.
6: End.
```

Fig. 4. The algorithm to compute a full set of characteristics for a join node.

**Lemma 13.** *A full set of characteristics for a node $i$ is non-empty if and only if the cutwidth of $G_i$ is at most $k$. If some full set of characteristics for $i$ is non-empty, then any full set of characteristics for $i$ $G_i$ is non-empty.*

An important consequence of Lemma 13 is that the cutwidth of $G$ is at most $k$, if and only if any full set of characteristics for the root $r$ is non-empty (recall that $G_r = G$). In [46] there are given algorithms able to construct a full set of characteristics for an *insert* or a *forget* node when a full set of characteristics for the unique child of $i$ is given. These algorithms, as well as the way to obtain a full set of characteristics for a *start* node, can be found in [46]. In what follows, we will show how to compute a full set of characteristics for a *join* node $i$ when two full set of characteristics for its children $j_1$, $j_2$ are given.

We consider now the case that node $i$ is a *join* node and $j_h$, $h = 1, 2$, are the two children of $i$ in $U$. We observe that $V(G_{j_1}) \cap V(G_{j_2}) = X_i$, $G_{j_1} \cup G_{j_2} = G_i$ and we recall that $E(G_{j_1}[X_i]) \cap E(G_{j_2}[X_i]) = \emptyset$. Given a full set of characteristics $FS(j_1)$ for $j_1$ and a full set of characteristics $F_{j_2}$ for $j_2$, we show that the algorithm Join-Node, given in Fig. 4, correctly computes a full set of characteristics $FS(i)$ for $i$.

We need the following lemma that relates the interleaving of sequences with particular types of layouts.

**Lemma 14.** *Let $G$, $G_1$ and $G_2$ be graphs where $G_1 \cup G_2 = G$ and $G_1 \cap G_2 = (S, \emptyset)$. Let also $l_1$, $l_2$ be vertex orderings of $G_1$ and $G_2$ respectively where $l_1[S] = l_2[S] = \lambda$. If $C_S(G_i, l_i) = (\lambda, \mathbf{A}_i)$, $i = 1, 2$, then, for any $\mathbf{A} \in \mathbf{A}_1 \bigotimes \mathbf{A}_2$, there exists a vertex ordering $l$ of $G$ where $l[V(G_i)] = l_i$, $i = 1, 2$, and $C_S(G, l) = (\lambda, \mathbf{A})$.*

**Proof.** We claim that the ordering $l$ in question is constructed by the procedures Construct-Join-Ordering and Join-Orderings given in Fig. 5. Figure 6 gives an example of the operation of the procedure Join-Orderings as well as for the proof that follows.

By construction, $l[V(G_i)] = l_i$, $i = 1, 2$. We have to prove that $C_S(G, l) = (\lambda, \mathbf{A})$. We set $G_i^* = (V(G), E(G_i))$, $i = 1, 2$, and we observe that $l$ is a vertex ordering for both $G_i$, $i = 1, 2$, where $|l| = r_1 + r_2 - \rho$. Let $l = [v_1, \ldots, v_r]$ and $\lambda = [v_{\kappa_1}, \ldots, v_{\kappa_\rho}]$. We use the notations

$$\overline{Q}_i = \mathbf{Q}_{G_i^*,l}(0) \cdot \cdots \cdot \mathbf{Q}_{G_i^*,l}(r), \quad i = 1, 2, \qquad \text{and} \qquad Q = \mathbf{Q}_{G,l}(0) \cdot \cdots \cdot \mathbf{Q}_{G_i,l}(r)$$

```
Procedure Construct-Join-Ordering($G_1, G_2, S, l_1, l_2, \mathbf{A}$).

Input: Two graphs $G_1$, $G_2$ and a set $S$ where $G_1 \cap G_2 = (S, \emptyset)$.
       Two vertex orderings $l_1$ and $l_2$ of $G_1$ and $G_2$, where $l_1[S] = l_2[S] = \lambda$.
       A sequence of typical sequences $\mathbf{A} \in \mathbf{A}_1 \bigotimes \mathbf{A}_2$ where $(\lambda, \mathbf{A}_i) = \mathrm{Com}(G_i, l_i, S)$, $i = 1, 2$.
Output: A vertex ordering $l$ of $G$ where $C_S(G, l) = (\lambda, \mathbf{A})$.

1: Assume that for $i = 1, 2$, let $l_i = [v^i_1, \ldots, v^i_{r_i}]$.
2: Let
       $\lambda = [v^1_{\kappa^1_1}, \ldots, v^1_{\kappa^1_\rho}] = [v^2_{\kappa^2_1}, \ldots, v^2_{\kappa^2_\rho}]$   where $\rho = |S|$.
3: For $i = 1, 2$, set
       $\kappa^i_0 = 0$   and   $\kappa^i_{\rho+1} = r_i + 1$.
4: For $i = 1, 2$, set $Q_i = \mathbf{Q}_{G_i, l_i}(0) \cdot \cdots \cdot \mathbf{Q}_{G_i, l_i}(r_i)$.
5: For any $h = 0, \ldots, \rho$, set
       $l^h_1 = l_1[\kappa^1_h + 1, \kappa^1_{i+1} - 1]$   and   $l^h_2 = l_2[\kappa^2_h + 1, \kappa^2_{i+1} - 1]$;
       $Q^h_1 = Q_1[\kappa^1_h, \kappa^1_{i+1} - 1]$   and   $Q^h_2 = Q_2[\kappa^2_h, \kappa^2_{i+1} - 1]$;
       $w^h = \text{Join-Orderings}(l^h_1, l^h_2, Q^h_1, Q^h_2, \mathbf{A}(h))$.
6: Set $l = w^0 \cdot [\lambda(1)] \cdot w^1 \cdot [\lambda(2)] \cdot w^2 \cdot \cdots \cdot [\lambda(\rho - 1)] \cdot w^{\rho-1} \cdot [\lambda(\rho)] \cdot w^\rho$.
7: Output $l$.
8: End.
```

```
Procedure Join-Orderings($l_1, l_2, Q_1, Q_2, A$).

Input: Two orderings $l_1, l_2$, two sequences $Q_1$, $Q_2$ where $|Q_i| = |l_i| + 1, 1, 2$, and
        a sequence $A \in \tau(Q_1) \otimes \tau(Q_2)$.
Output: An ordering $l$.

1: Compute $B_1$, $B_2$ so that $A = \tau(B_1 + B_2)$, where $B_1 \sim B_2$, and $B_i \in \mathcal{E}(\tau(Q_i))$, $i = 1, 2$.
2: Set $w = |B_1| = |B_2|$, and denote $\nu = \nu_{B_1, B_2}$.
3: For $j = 1, \ldots, w - 1$ set $m_j = l_{\nu(j)}[\beta_{Q_{\nu(j)}}(\beta^{-1}_{B_{\nu(j)}}(j)), \beta_{Q_{\nu(j)}}(\beta^{-1}_{B_{\nu(j)}}(j) + 1)) - 1]$.
4: Output $m_1 \cdot \cdots \cdot m_{w-1}$.
5: End.
```

Fig. 5. Procedures Construct-Join-Ordering and Join-Orderings.

(recall that for the sequences $Q$, $\overline{Q}_1$, and $\overline{Q}_2$ we insisted that their first elements are indexed by 0). We also set $Q^h = Q[\kappa_h, \kappa_{h+1} - 1]$ and $\overline{Q}^h_i = \overline{Q}_i[\kappa_h, \kappa_{h+1} - 1]$ for $i = 1, 2$ and $h = 0, \ldots, \rho$ (where $\kappa_0 = 0$ and $\kappa_{h+1} = r + 1$). Our target is to prove that $\forall_{0 \leqslant h \leqslant \rho}\, \tau(Q^h) = \mathbf{A}(h)$.

From the fact that $G_1$ and $G_2$ do not have edges in common we get that

$$\forall_{0 \leqslant h \leqslant \rho} \quad \overline{Q}^h_1 + \overline{Q}^h_2 = Q^h. \tag{3}$$

We may assume that for any $h = 0, \ldots, \rho$, the computation of $l_h$ is based on a pair $B^h_1$, $B^h_2$ where

$$\begin{aligned} &\mathbf{A}(h) = \tau(B^h_1 + B^h_2),\\ &B^h_1 \sim B^h_2,\\ &B^h_i \sqsupseteq \tau(Q^h_i), \quad i = 1, 2. \end{aligned} \tag{4}$$

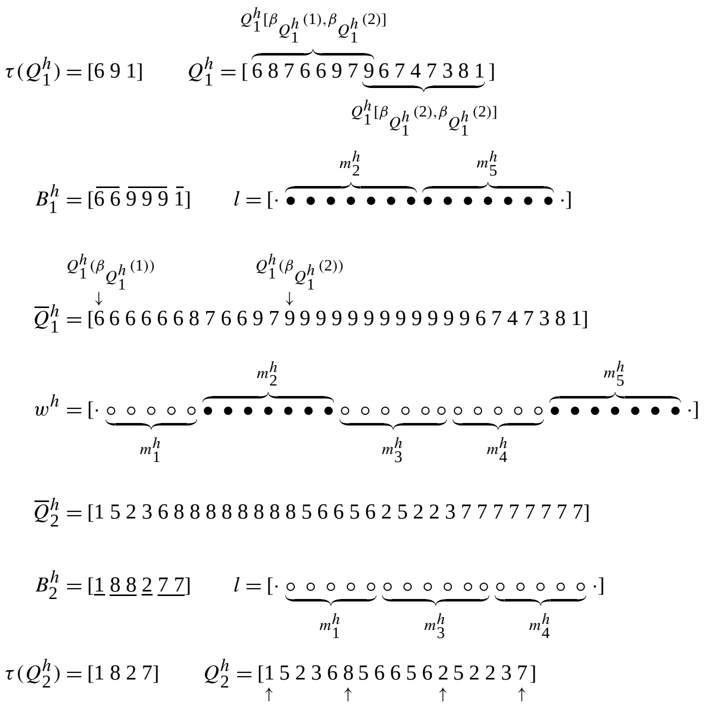
τ(Q₁ʰ) = [6 9 1]
Q₁ʰ = [6 8 7 6 6 9 7 9 6 7 4 7 3 8 1]
B₁ʰ = [6 6 9 9 9 1]
Q̄₁ʰ = [6 6 6 6 6 6 8 7 6 6 9 7 9 9 9 9 9 9 9 9 9 9 9 9 6 7 4 7 3 8 1]
Q̄₂ʰ = [1 5 2 3 6 8 8 8 8 8 8 8 8 5 6 6 5 6 2 5 2 2 3 7 7 7 7 7 7 7 7]
B₂ʰ = [1 8 8 2 7 7]
τ(Q₂ʰ) = [1 8 2 7]
Q₂ʰ = [1 5 2 3 6 8 5 6 6 5 6 2 5 2 2 3 7]

Fig. 6. An example of the proof of Lemma 14.

Notice that the result follows by [46, Lemma 2.3], (3), (4), and (5) bellow.

$$\forall_{i=1,2}\ \forall_{0\leqslant h\leqslant \rho}\quad \overline{Q}_i^h \sqsupseteq B_i^j. \tag{5}$$

It now remains to prove the correctness of (5). We will examine only the case where $i=1$ (the case $i=2$ is symmetric). Let $m_1^h,\dots,m_{w-1}^h$ be the vertex orderings produced during step **3** of Join-Orderings$(l_1^h, l_2^h, Q_1^h, Q_2^h, \mathbf{A}(h))$. Let

$$q_j^h = |m_1^h| + \cdots + |m_j^h|,\quad 1\leqslant j\leqslant w-1$$

(assume that $q_0^h=0$ and $w=|B_1^h|=|B_2^h|$). The result follows from [46, Lemma 2.3] taking into account that

$$\forall_{j,1\leqslant j\leqslant w-1}\quad \overline{Q}_1^h\big[q_{j-1}^h+1, q_j^h+1\big] \sqsupseteq \big[B_1^h(j), B_1^h(j+1)\big]. \tag{6}$$

Towards proving (6) we make first some observations. We will call a *gap* of a vertex ordering the “space” between two consecutive vertices, the “space” on the left of the first vertex, and the “space” on the right of the last vertex. Clearly, each gap of a vertex ordering is crossed by the edges with endpoints on different sides. Notice that $\overline{Q}_1^h(1)$ corresponds to the number of edges that cross the “gap” of $l$ that is on the left of vertex $w^h(1)$ and that for any $j$, $1\leqslant j\leqslant w-1$, and any $t$, $1\leqslant t\leqslant |m_j^h|$, $\overline{Q}_1^h(q_{j-1}^h+t+1)$ corresponds to the number of edges that cross the “gap” on the right of vertex $w^h(q_{j-1}^h+t)$. Moreover, for any $j$, $1\leqslant j\leqslant w-1$, $\overline{Q}_1^h(q_{j-1}^h+1)$ corresponds to the number of edges that cross the “gap” on the left of vertex $m_j^h(1)$ and for any $t$, $1 \leqslant t \leqslant |m_j^h|$, $\overline{Q}_1^h(q_{j-1}^h + t + 1)$ corresponds to the number of edges that cross the "gap" on the right of vertex $m_j^h(t)$. Let $j$ be an integer $1 \leqslant j \leqslant w - 1$.

Let

$$\nu = \nu_{B_1^h, B_2^h}.$$

We will consider two cases depending on whether $B_1^h(j) = B_1^h(j+1)$ or $B_1^h(j) \neq B_1^h(j+1)$.

If $\nu(j) = 1$, then $m_j$ is a copy of a part of the linear ordering $l_1$ of $G_1$. As the relative position of the vertices of $G_1$ are the same in $l$ as in $l_1$ and $E(G_1^*) = E(G_1)$, the edges crossing the "gaps" of $l$ delimiting the vertices of $m_j$ are the same as the edges crossing the "gaps" delimiting the same vertices in $l_1$. Therefore, the sequence of their cardinalities is $Q_1^h[\beta_{Q_1^h}(\beta_{B_1^h}^{-1}(j)), \beta_{Q_1^h}(\beta_{B_1^h}^{-1}(j)+1)]$. Therefore,

$$\begin{aligned} \nu(j) = 1 \quad \Rightarrow \quad & \overline{Q}_1^h\big[q_{j-1} + 1, q_j^h + 1\big] \\ & = Q_1^h\big[\beta_{Q_1^h}\big(\beta_{B_1^h}^{-1}(j)\big), \beta_{Q_1^h}\big(\beta_{B_1^h}^{-1}(j) + 1\big)\big]. \end{aligned} \tag{7}$$

If $\nu(j) = 2$, then $m_j$ is a copy of a part of the linear ordering $l_2$ of $G_2$. We define

$$\begin{aligned} t_{\text{left}} = \max\big\{t \mid & \text{ the "gap" corresponding to } \overline{Q}_1^h(t) \text{ is on the left of } m_j(1) \text{ and has the} \\ & \text{vertex on its left (if exists) in } l_1 \text{ and the vertex on its right not in } l_1\big\}, \end{aligned}$$

$$\begin{aligned} t_{\text{right}} = \min\big\{t \mid & \text{ the "gap" corresponding to } \overline{Q}_1^h(t) \text{ is on the right of } m_j(|m_j|) \text{ and} \\ & \text{has the vertex on its left not in } l_1 \text{ and the vertex on its right} \\ & \text{(if exists) in } l_1\big\}. \end{aligned}$$

Notice that $t_{\text{left}} \leqslant q_{j-1} + 1$ and $q_j + 1 \leqslant t_{\text{right}}$. Observe that the vertices in $w^h$ that are delimited by "gaps" corresponding to $\overline{Q}_1^h[t_{\text{left}}, t_{\text{right}}]$ are all vertices not in $V(G_1)$. Hence they are all isolated vertices of $G_1^*$. Recall that if we remove from $l$ all the vertices in $V(G_2) - S$, what remains is $l_1$. This operation replaces the "gaps" corresponding to $\overline{Q}_1^h[t_{\text{left}}, t_{\text{right}}]$ with only one "gap" corresponding to the $\beta_{Q_1^h}(s_j)$th gap of $Q_1^h$ where

$$s = \beta_{B_1^h}^{-1}(j).$$

However, the fact that the relative position of the vertices of $G_1$ is the same in $l$ as in $l_1$ and the fact that all the removed vertices are isolated makes the crossing edges of the replaced "gaps" to be exactly the same as the crossing edges of the resulting "gap". Resuming, we have the following.

$$\nu(j) = 2 \quad \Rightarrow \quad \forall_{t, q_{j-1}+1 \leqslant t \leqslant q_j^h + 1} \quad \overline{Q}_1^h(t) = Q_1^h\big(\beta_{Q_1^h}\big(\beta_{B_1^h}^{-1}(j)\big)\big). \tag{8}$$

From [46, Lemma 2.6], we have that

$$\begin{aligned} & \tau\big(Q_1^h\big[\beta_{Q_1^h}\big(\beta_{B_1^h}^{-1}(j)\big), \beta_{Q_1^h}\big(\beta_{B_1^h}^{-1}(j) + 1\big)\big]\big) \\ & \quad = \big[Q_1^h\big(\beta_{Q_1^h}\big(\beta_{B_1^h}^{-1}(j)\big)\big), Q_1^h\big(\beta_{Q_1^h}\big(\beta_{B_1^h}^{-1}(j) + 1\big)\big)\big] \end{aligned} \tag{9}$$

and applying [46, Lemma 2.7(1)]

$$Q_1^h\big(\beta_{Q_1^h}\big(\beta_{B_1^h}^{-1}(j)\big)\big) = B_1^h(j), \tag{10}$$

$$Q_1^h\big(\beta_{Q_1^h}\big(\beta_{B_1^h}^{-1}(j+1)\big)\big) = B_1^h(j+1). \tag{11}$$

If $\nu(j) = 1$ then [46, Lemma 2.7(2)] implies $\beta_{B_1^h}^{-1}(j+1) = \beta_{B_1^h}^{-1}(j) + 1$ and therefore (11) can be written

$$Q_1^h\big(\beta_{Q_1^h}\big(\beta_{B_1^h}^{-1}(j) + 1\big)\big) = B_1^h(j+1). \tag{12}$$

From (7), (9), (10), and (12) we conclude that

$$\forall_{j, 1 \leqslant j \leqslant w-1, \nu(j)=1} \quad \tau\big(\overline{Q}_1^h\big[q_{j-1}+1, q_j^h+1\big]\big) = \big[B_1^h(j), B_1^h(j+1)\big] \tag{13}$$

which clearly yields (6) for the cases where $\nu(j) = 1$.

If $\nu(j) = 2$ then [46, Lemma 2.7(3)] implies $\beta_{B_1^h}^{-1}(j+1) = \beta_{B_1^h}^{-1}(j)$ and therefore (10) and (11) give

$$Q_1^h\big(\beta_{Q_1^h}\big(\beta_{B_1^h}^{-1}(j)\big)\big) = B_1^h(j) = B_1^h(j+1) \tag{14}$$

and (14) combined with (8), gives (6) for the cases where $\nu(j) = 2$. $\square$

**Lemma 15.** *Let $G$, $G_1$ and $G_2$ be graphs where $G_1 \cup G_2 = G$ and $G_1 \cap G_2 = (S, \emptyset)$. Let also $l$ be a vertex ordering of $G$. We denote $l_i = l[V(G_i)]$, $C_S(G, l) = (\lambda, \bar{\mathbf{A}})$, and $C_S(G_i, l_i) = (\lambda, \mathbf{A}_i)$, $i = 1, 2$. Then there exists a sequence of typical sequences $\mathbf{A} \in \mathbf{A}_1 \bigotimes \mathbf{A}_2$ such that $\mathbf{A} \prec \bar{\mathbf{A}}$.*

**Proof.** Let $r_i = |l_i|$, $i = 1, 2$, and $\rho = |\lambda| = |S|$. As in the proof of Lemma 14, we set $G_i^* = (V(G), E(G_i))$, $i = 1, 2$, and we observe that $l$ is a vertex ordering for both $G_i^*$, $i = 1, 2$, where $|l| = r_1 + r_2 - \rho$. We use the notations $\overline{Q}_i = \mathbf{Q}_{G_i^*, l}(0) \cdot \dots \cdot \mathbf{Q}_{G_i^*, l}(r)$, $i = 1, 2$, and $Q = \mathbf{Q}_{G,l}(0) \cdot \dots \cdot \mathbf{Q}_{G_i, l}(r)$. As $E(G_1) \cap E(G_2) = \emptyset$ and $E(G_1) \cup E(G_2) = E(G)$ we get,

$$Q = \overline{Q}_1 + \overline{Q}_2. \tag{15}$$

We denote $Q_i = \mathbf{Q}_{G_i, l_i}(0) \cdot \dots \cdot \mathbf{Q}_{G_i, l}(r)$, $i = 1, 2$, and we observe that the facts that $V(G_i) \subseteq V(G_i^*)$, $i = 1, 2$, and $E(G_i) = E(G_i^*)$, $i = 1, 2$, imply that,

$$\forall_{i=1,2} \quad Q_i \sqsubseteq \overline{Q}_i. \tag{16}$$

We assume that if $l = [v_1, \dots, v_r]$, then $\lambda = [v_{\kappa_1}, \dots, v_{\kappa_\rho}]$ and if $l_i = [u_1^i, \dots, u_{r_i}^i]$ then that $\lambda = [u_{\mu_1^i}^i, \dots, u_{\mu_\rho^i}^i]$, $i = 1, 2$. We set

$$Q^h = Q[\kappa_h, \kappa_{h+1} - 1], \quad Q_i^h = Q\big[\mu_h^i, \mu_{h+1}^i - 1\big], \quad i = 1, 2, \quad \text{and}$$
$$\overline{Q}_i^h = \overline{Q}_i[\kappa_h, \kappa_{h+1} - 1], \quad i = 1, 2 \text{ and } h = 0, \dots, \rho$$

(where $\kappa_0 = \mu_0^1 = \mu_0^2 = 0$ and $\kappa_{h+1} = \mu_{h+1}^1 = \mu_{h+1}^2 = r + 1$). From the view point of these new notations, (15) and (16) can be rewritten as follows

$$\forall_{h,0\leqslant h\leqslant \rho} \quad Q^h = \overline{Q}_1^h + \overline{Q}_2^h, \tag{17}$$

$$\forall_{h,0\leqslant h\leqslant \rho}\ \forall_{i=1,2} \quad Q_i^h \sqsubseteq \overline{Q}_i^h. \tag{18}$$

Notice also that

$$\forall_{i=1,2}\ \forall_{h,0\leqslant h\leqslant \rho} \quad \mathbf{A}_i(h) = \tau\big(Q_i^h\big), \tag{19}$$

$$\forall_{h,0\leqslant h\leqslant \rho} \quad \bar{\mathbf{A}}(h) = \tau\big(Q^h\big), \tag{20}$$

(18) and (19), implies that

$$\forall_{i=1,2}\ \forall_{h,0\leqslant h\leqslant \rho} \quad \mathbf{A}_i(h) = \tau\big(\overline{Q}_i^h\big). \tag{21}$$

Our target is to prove that for $h = 0, \dots, \rho$ there exists a typical sequence $A'$ in $\mathbf{A}_1(h) \otimes \mathbf{A}_2(h)$ such that $A' \prec \bar{\mathbf{A}}(h)$. This follows from (17), (20), and (21) if, for $h = 0, \dots, \rho$, we apply Lemma 11 for $Q^h$, $\overline{Q}_1^h$ and $\overline{Q}_2^h$. $\square$

**Lemma 16.** *If $i$ is a join node with children $j_h$, $h = 1, 2$, and, for $h = 1, 2$, $FS(j_h)$ is a full set of characteristics for $j_h$. Then, the set $FS(i)$ constructed by the algorithm* Join-Node *is a full set of characteristics for $i$.*

**Proof.** We will prove first that $FS(i)$ is a set of characteristics. To avoid overloaded expressions, whenever we refer to a characteristic, we will insist that its width is bounded by $k$. For this, it is sufficient to show that for any $(\lambda, \mathbf{A}) \in FS(i)$, there exists a vertex ordering $l$ of $G$ such that $C_{X_i}(G, \lambda) = (\lambda, \mathbf{A})$.

By the algorithm Join Node we can assume that there exist two pairs $(\lambda, \mathbf{A}_h)$, $h = 1, 2$, where

$$(\lambda, \mathbf{A}_h) \in FS(j_h), \quad h = 1, 2, \tag{22}$$

$$\mathbf{A} \in \mathbf{A}_1 \bigotimes \mathbf{A}_2. \tag{23}$$

As $FS(j_h)$, $h = 1, 2$, are both sets of characteristics (22) implies that there exist two orderings $l_1, l_2$ of $G_{X_{j_1}}$ and $G_{X_{j_2}}$ respectively such that

$$\lambda = l_1[X_{j_1}] = l_2[X_{j_2}], \tag{24}$$

$$(\lambda, \mathbf{A}_h) = C_{X_{j_h}}(G_h, l_h), \quad i = 1, 2. \tag{25}$$

Using now (23)–(25), we can apply Lemma 14 and conclude that there exists a vertex ordering $l$ of $G_i$ such that $C_{X_i}(G, l) = (\lambda, \mathbf{A})$. Therefore, $FS(i)$ is a set of characteristics.

It remains now to prove that $FS(i)$ is a full set of characteristics. To prove this we have to show that, for any vertex ordering $l$ of $G_i$ there exists a vertex ordering $l^*$ of $G_i$ such that $C_{X_i}(G_i, l^*) \in FS(i)$ and $C_{X_i}(G_i, l) \prec C_{X_i}(G_i, l^*)$.

Let $l_h = l[V(G_{j_h})]$, $h = 1, 2$, and set $(\lambda, \mathbf{A}_h) = C_{X_{j_h}}(G_{j_h}, l_h)$, $h = 1, 2$, and $(\lambda, \bar{\mathbf{A}}) = C_{X_i}(G_i, l)$. From Lemma 15, there exist a typical sequence $\mathbf{A}$ such that

$$\mathbf{A} \in \mathbf{A}_1 \bigotimes \mathbf{A}_2, \tag{26}$$

$$(\lambda, \mathbf{A}) \prec \big(\lambda, \bar{\mathbf{A}}\big). \tag{27}$$

```
Algorithm Check-Cutwidth$(G, X, k)$.

Input: A graph $G$, a tree decomposition $X$ of $G$ with width $w$, and an integer $k$.
Output: Whether cutwidth$(G) \leqslant k$.

  Assume that the nodes on $X$ are topologically sorted, and that $X = [X_1, \ldots, X_r]$.

 1: For any $i$, $1 \leqslant i \leqslant r$, compute a full set of $X_i$-characteristics $F_i$ for $G_i$
      setting $F_i = \{[x], [[0][0]]\}$ when $X_i$ is a start node and $X_i = \{x\}$;
      using $F_j$ when $X_i$ is an introduce or a forget node, relatively to some $X_j$, $j < i$;
      using $F_{j_1}$ and $F_{j_2}$ if $X_i$ is the join of $X_{j_1}$ and $X_{j_2}$.
 2: Output $F_r \neq \emptyset$.
 3: End.
```

Fig. 7. Algorithm Check-Cutwidth.

Recall now that, for $h = 1, 2$, that $FS(j_h)$ is a full set of characteristics for $j_h$ and therefore, for $h = 1, 2$, there exists a vertex ordering $l_h^*$ of $G_{j_h}$ where

$$C_{X_{j_h}}\bigl(G_{j_h}, l_i^*\bigr) \in FS(j_h), \tag{28}$$

$$C_{X_{j_h}}\bigl(G_{j_h}, l_i^*\bigr) \prec C_{X_{j_h}}\bigl(G_{j_h}, l_h\bigr). \tag{29}$$

Let $(\lambda, \mathbf{A}_i^*) = C_{X_{j_h}}(G_{j_h}, l_h^*)$, $h = 1, 2$, and (28) and (29) can be rewritten as follows.

$$\bigl(\lambda, \mathbf{A}_h^*\bigr) \in FS(j_h), \quad h = 1, 2, \tag{30}$$

$$\bigl(\lambda, \mathbf{A}_h^*\bigr) \prec (\lambda, \mathbf{A}_h), \quad h = 1, 2. \tag{31}$$

From (26), (31), and applying Lemma 9, there exists a characteristic $(\lambda, \mathbf{A}^*)$ such that

$$\mathbf{A}^* \in \mathbf{A}_1^* \bigotimes \mathbf{A}_2^*, \tag{32}$$

$$\bigl(\lambda, \mathbf{A}^*\bigr) \prec (\lambda, \mathbf{A}). \tag{33}$$

Notice now that, from (32), and Lemma 14, there exists a vertex ordering $l^*$ of $G$ such that $C_{X_i}(G_i, l^*) = (\lambda, \mathbf{A}^*)$. The fact that $C_{X_i}(G_i, l^*) \in FS(i)$ follows from (30), (32), and the algorithm Join-Node. Finally, (27), and (33) imply that $C_{X_i}(G_i, l^*) = (\lambda, \mathbf{A}^*) \prec (\lambda, \bar{\mathbf{A}}) = C_{X_i}(G_i, l)$ and this completes the proof of the lemma. □

We can now put together the results on this section and those of Section 3 of [46] to define the algorithm Cutwidth, as given in Fig. 7.

**Theorem 17.** *The algorithm* Check-Cutwidth, *given a graph $G$ with $n$ vertices and a tree decomposition $(X, U)$ of $G$ of $O(n)$ nodes and width at most $w$, checks whether there exists an vertex ordering of $V(G)$ of cutwidth at most $k$ in $O(wk \cdot ((w+1)!)^2 (\frac{8}{3})^{2w} 2^{k \cdot 8(w+1)})$ steps.*

**Proof.** From Lemma 1 there exists an algorithm that in $O(n)$ steps transforms $(X, U)$ to a nice tree decomposition $(X, U, r)$. We have to determine the cost of computing $FS(i)$ for all the nodes of $U$. Let $i$ be such a node.

If $i$ is a *start* node then the computation of $FS(i)$ needs $O(1)$ steps.

If $i$ is an *introduce* node then Lemma 12 bounds the number of repetitions of step **2** of the algorithm Introduce-Node by $(w+1)!(\frac{8}{3}2^{2k})^{w+1}$. Moreover, step **3** involves at most $w$ repetitions and step **4** at most $2k-1$ calls of the procedure Ins [46, Lemma 2.4]. The procedure Ins is dominated by its 2nd step which requires $O(w^2k)$ steps: $O(w)$ for each of the edges inserted, at most $w+1$ for the sequences of $\mathbf{A}'$ whose elements should be incremented, and at most $2k-1$ for the elements of each one of these sequences [46, Lemma 2.4]. Similarly, calculating $\max(\mathbf{A}')$ requires at most $(w+1)(2k-1)=O(wk)$ steps. Therefore, computing $FS(i)$ requires $O(w^3k^2\cdot(w+1)!(\frac{8}{3}2^{2k})^{w+1})$ steps.

If $i$ is a *forget* node then Lemma 12 bounds the number of repetitions of step **2** of the algorithm Forget-Node by $(w+1)!(\frac{8}{3}2^{2k})^{w+1}$. As the procedure Del needs $O(wk)$ steps, computing $FS(i)$ requires $O(wk\cdot(w+1)!(\frac{8}{3}2^{2k})^{w+1})$ steps.

If $i$ is a *join* node then Lemma 12 bounds the number of repetitions of step **2** of the algorithm Join-Node by $((w+1)!(\frac{8}{3}2^{2k})^{w+1})^2$. From Lemma 6, the computation of $\mathbf{A}_1\bigotimes\mathbf{A}_2$ costs $O(k\cdot 2^{4(k-1)(w+1)})$ steps and

$$\left|\mathbf{A}_1\bigotimes\mathbf{A}_2\right|\leqslant 2^{4(k-1)(w+1)}\leqslant 2^{4k(w+1)}.$$

As step **4** costs $O(wk)$ steps, we conclude that computing $FS(i)$ requires

$$O\left(wk\cdot 2^{4k(w+1)}\cdot\big((w+1)!\big)^2\left(\frac{8}{3}2^{2k}\right)^{2w+2}\right)$$
$$=O\left(wk\cdot\big((w+1)!\big)^2\left(\frac{8}{3}\right)^{2w+2}2^{k(8w+8)}\right)$$

steps.

Notice that according to the analysis above, the prevailing time is the one of the *join* nodes. As $U$ contains $O(n)$ nodes, the result follows. $\square$

Let us refer as the Cutwidth algorithm to the algorithm that for any $k=1,\ldots,\lfloor(w+1)d\log n\rfloor$ checks whether cutwidth$(G)\leqslant k$ by calling the algorithm Check-Cutwidth as a subroutine. From Corollary 4 and Theorem 17, if we set $k=(w+1)d\log n$, we have the following result.

**Theorem 18.** *The algorithm* Cutwidth, *given a graph $G$ with $n$ vertices of degree no more than $d$ and a tree decomposition $(X,U)$ of $G$ of $O(n)$ nodes and width at most $w$, computes the cutwidth of $G$ in $k$ in $O(w^3d^2\cdot\log^2 n\cdot((w+1)!)^2(\frac{8}{3})^{2w}n^{8d(w+1)^2})$ steps.*

## 5. Computing a vertex ordering

In this section we show how the algorithms Check-Cutwidth given in Fig. 7 can be modified in a way that, in the case that the graph has cutwidth $k$, it also construct a vertex ordering with cutwidth $k$, we will refer to such modification as the algorithm Layout-Cutwidth.

Suppose now that, given a tree decomposition $(X,U)=(X_i\mid i\in V(U),U)$ of $G$ with width bounded by $w$, after running the algorithm described in the previous subsections we know that a graph $G$ has cutwidth at most $k$, i.e., the computed set $FS(r)$ is not empty. We will now describe a method to construct a vertex ordering of $G$ with cutwidth at most $k$. By observing the flow of the algorithm, it follows that we can assign to each node $i$ of $(X, U)$, a characteristic $(\lambda_i, \mathbf{A}_i)$ *witness on node* $i$, in a bottom–up fashion:

- If $i$ is a *start* node then $(\lambda_i, \mathbf{A}_i) = ([x], [[0], [0]])$ is the unique characteristic of the vertex ordering consisting of the unique vertex $x$ in $X_i$.
- If $i$ is an *introduce* node then $(\lambda_i, \mathbf{A}_i)$ is one of the characteristics constructed after the application of the algorithm Introduce-Node on characteristic $(\lambda_j, \mathbf{A}_j)$ where $j$ is the unique child of $i$.
- If $i$ is a *forget* node then $(\lambda_i, \mathbf{A}_i)$ is one of the characteristics constructed after the application of the algorithm Forget-Node on characteristic $(\lambda_j, \mathbf{A}_j)$ where $j$ is the unique child of $i$.
- If $i$ is a *join* node then $(\lambda_i, \mathbf{A}_i)$ is one of the characteristics constructed after the application of the algorithm Join-Node on characteristics $(\lambda_{j_1}, \mathbf{A}_{j_1})$ and $(\lambda_{j_2}, \mathbf{A}_{j_2})$ where $j_1$ and $j_2$ are the children of $i$.
- $(\lambda_r, \mathbf{A}_r)$ is one of the characteristics in $FS(r)$.

We call the collection $\mathcal{W} = ((\lambda_i, \mathbf{A}_i), i \in V(U))$, *witness* tree. Notice that if at each time a new characteristic is computed, we set up a pointer to the characteristic it was constructed from, we obviously have a suitable structure for constructing also a witness tree in $O(|V(U)|)$ steps. Let us show now how to compute, using the information of $\mathcal{W}$, a vertex ordering of $V(G_r)$ with cutwidth $\leqslant k$. Towards this, we will compute, for each node $i \in V(U)$ a vertex ordering $l_i$ of $V(G_i)$ such that $C_{X_i}(G_i, l_i) = (\lambda_i, \mathbf{A}_i)$. The case where $i$ is a start node is obvious. The cases where $i$ is either an insert or a forget node are omitted as they are presented in detail in Section 4 of [46]. In each of these cases the new vertex ordering $l_i$ can be constructed in $O(wk)$ steps.

Assume now that $i$ is a *join* node with children $j_1$ and $j_2$. Let also $l_{j_h}$, $h = 1, 2$, be two vertex orderings of $G_{j_h}$, $h = 1, 2$, respectively, such that $C_{X_{j_h}}(G_{j_h}, l_{j_h}) = (\lambda_{j_h}, \mathbf{A}_{j_h})$, $h = 1, 2$. We show how to construct a vertex ordering $l_i$ such that $C_{X_i}(G_i, l_i) = (\lambda_i, \mathbf{A}_i)$.

Notice that, from the algorithm Join-Node, $\mathcal{A}_i$ is a member of $\mathbf{A}_{i_1} \bigotimes \mathbf{A}_{i_2}$. Recall now that, from Lemma 14, the procedure Construct-Join-Ordering$(G_{j_1}, G_{j_2}, X_i, l_{j_1}, l_{j_2}, \mathbf{A})$ is able to construct such a vertex ordering. Notice that if, for $h = 1, 2$, we maintain for each $v \in X_{j_h}$ a pointer indicating the position of the same vertex in $l_{j_h}$, the procedure Construct-Join-Ordering$(G_{j_1}, G_{j_2}, X_i, l_{j_1}, l_{j_2}, \mathbf{A})$ will call the procedure Join-Orderings $|X_i|$ times. If now, for $h = 1, 2$, we additionally maintain a data structure associating each of the "gaps" of $\lambda_{j_h}$ to the limits of its corresponding sequence of "gaps" in $l_{j_h}$, we can implement step **3** of the procedure Join-Orderings in $O(k)$ steps. Resuming, we have that the construction of $l_i$ costs $O(kw)$ steps.

Therefore, the computation of a vertex ordering $l_r$ where $C_{X_r}(G, l_r) = (\lambda_{l_r}, \mathbf{A}_{l_r})$, can be done in $O(nkw)$ steps. Therefore, we have the following analogue of Theorem 17.

**Theorem 19.** *Given a graph $G$ with $n$ vertices and a tree decomposition $(X, U)$ of $G$ of $O(n)$ nodes and width at most $w$, the algorithm* Layout-Cutwidth *checks whether there exists a vertex ordering of $V(G)$ of cutwidth at most $k$ and, if this is the case, outputs a vertex ordering of $V(G)$ of cutwidth at most $k$ in $O(wk \cdot ((w + 1)!)^2 (\frac{8}{3})^{2w} 2^{k \cdot 8(w+1)})$ steps.*

As $k \leqslant (w+1)d \log n$, we obtain that the algorithm Cutwidth can be modified to output an optimal vertex ordering with an additional call to Layout-Cutwidth once the cutwidth of $G$ has been computed. We refer to this modification as the algorithm Min-Layout-Cutwidth. Theorem 18, can now be rewritten as follows.

**Theorem 20.** *Given a graph $G$ with $n$ vertices of degree no more than $d$ and a tree decomposition $(X, U)$ of $G$ with $O(n)$ nodes and width at most $w$, the algorithm* Min-Layout-Cutwidth *outputs a vertex ordering of $V(G)$ of minimum cutwidth in $O(w^3 d^2 \cdot \log^2 n \cdot ((w+1)!)^2 (\frac{8}{3})^{2w} n^{8d(w+1)^2})$ steps.*

According to the main result in [5], a minimum width tree decomposition of any partial $w$-tree can be constructed in $O(w^{O(w)} 2^{O(w^3)} n)$ steps (see also [31,34,38]). This algorithm can serve as a preprocessing step to the algorithm Layout-Cutwidth of Theorem 20 that with input a partial $w$-tree $G$ with vertices of degree at most $d$, outputs a vertex ordering of $G$ of minimum cutwidth.

## 6. Computing the pathwidth of bounded degree partial $w$-trees

We will now show how to use the algorithms of the previous sections in order to compute the pathwidth of a partial $w$-tree with bounded maximum degree.

The definition of treewidth is extended to hypergraphs by replacing edges with hyperedges. We define cutwidth for hypergraphs by extending the definition of $E(S)$ for $S \subseteq V(G)$ such that $E(S)$ contains all the hyperedges with at least one endpoint in $S$. We can prove the following extension of 19.

**Theorem 21.** *Given a hypergraph $G$ with $n$ vertices and a tree decomposition $(X, U)$ of $G$ of $O(n)$ nodes and width at most $w$, the algorithm* Layout-Cutwidth *checks whether there exists a vertex ordering of $V(G)$ of cutwidth at most $k$ and, if this is the case, outputs a vertex ordering of $V(G)$ of cutwidth at most $k$ in $O(wk \cdot ((w+1)!)^2 (\frac{8}{3})^{2w} 2^{k \cdot 8(w+1)})$ steps.*

**Proof.** By extending Lemma 1 for hypergraphs we assume that $(X, U, r)$ is a nice tree-decomposition of $G$. To prove the theorem, it is sufficient to observe that all the algorithms of the previous sections and [46] can be straightforwardly generalized to hypergraphs with the same time costs. In particular, the algorithms Forget-Node and Join-Node are exactly the same as they involve only operations on sequences of integers. The only changes required, concern the procedure Ins, described in [46], and are the following two:

1. The set $N$ should now represent the set $\{U_1, \ldots, U_\sigma\}$ where $\{U_t \cup \{u\} \mid 1 \leqslant t \leqslant \sigma\}$ are the hyperedges of $G_i$ that contain $u$ as endpoint. For any $U_t$, $1 \leqslant t \leqslant \sigma$, we set up $j_t^{\mathbf{l}}$ ($j_t^{\mathbf{r}}$) as the smallest (biggest) of the indices corresponding to the vertices of $U_t$ in $\lambda$ (notice that in the case where $G$ is a graph $j_t^{\mathbf{l}} = j_t^{\mathbf{r}}$).

Notice that none of the hyperedges of $G$ can have size bigger than $w+1$, as they have all to fit in some node of the tree decomposition. Therefore $|U_t| \leqslant w+1$, $1 \leqslant t \leqslant \sigma$. Moreover, we can assume that no vertex is an endpoint of more than $2k$ hyperedges as in such a case the cutwidth of $G$ should be greater than $k$. Therefore, $\sigma \leqslant 2k$. These two facts imply that computing $j_t^{\mathbf{l}}$ and $j_t^{\mathbf{r}}$ for $1 \leqslant t \leqslant \sigma$ can be done in $O(kw^2)$ steps.

2. In step **2** of $\mathsf{Ins}$, cases (i) and (ii) should now be:

(i) If $j_h^{\mathbf{r}} \leqslant j$ then set $\mathbf{A}' \leftarrow \mathbf{A}'[0, j_h^{\mathbf{l}} - 1] \cdot (\mathbf{A}'[j_h^{\mathbf{l}}, j] + 1) \cdot \mathbf{A}'[j + 1, \rho + 1]$.
(ii) If $j_h^{\mathbf{l}} \geqslant j + 1$ then set $\mathbf{A}' \leftarrow \mathbf{A}'[0, j] \cdot (\mathbf{A}'[j + 1, j_h^{\mathbf{r}}] + 1) \cdot \mathbf{A}'[j_h^{\mathbf{r}} + 1, \rho + 1]$.
(iii) If $j_h^{\mathbf{l}} \leqslant j < j_h^{\mathbf{r}}$ then set $\mathbf{A}' \leftarrow \mathbf{A}'[0, j_h^{\mathbf{l}} - 1] \cdot (\mathbf{A}'[j_h^{\mathbf{l}} + 1, j_h^{\mathbf{r}}] + 1) \cdot \mathbf{A}'[j_h^{\mathbf{r}} + 1, \rho + 1]$.

The third case above examines the case where the added hyperedge contains vertices residing in both sides of the insertion point. Notice that the time cost of the modified step **2** is the same as the time cost of the old one. Finally, the complexity of $\mathsf{Ins}$ does not change with this small modification as the time required to compute the pairs $j_h^{\mathbf{l}}, j_h^{\mathbf{r}}$ for each of the inserted hyperedges does not prevail the time cost of the second step. □

We call a graph *trunked* if it does not contain vertices of degree 1. Given a trunked graph $G$ we define its *dual hypergraph* as $G^D = (E(G), \{E_G(v) \mid v \in V(G)\})$. In what follows, we will denote as $\Delta(G)$ the maximum degree of the vertices of a graph $G$. The following lemma shows how to transform a tree decomposition of $G$ to one of $G^D$.

**Lemma 22.** *For any trunked graph $G$,* $\text{treewidth}(G^D) \leqslant \text{treewidth}(G) \cdot \Delta(G)$.

**Proof.** Let $(X, U)$ be a tree decomposition of $G$ with width $\leqslant k$. Notice that $\Delta(G)$ is equal to the maximum size of a hyperedge in $G^D$. We construct a tree decomposition $(Y, U)$ of $G^D$ using the same tree $U$ and setting $Y_i = \{e \in E(G) \mid e \cap X_i \neq \emptyset\}$. Notice that, for any $i \in V(U)$, $|Y_i| \leqslant \Delta(G) \cdot |X_i|$. It remains to prove that $(Y, U)$ is a tree decomposition. Condition (1) is obvious. For condition (2), suppose that $e^* = \{e_1, \ldots, e_r\}$ is a hyperedge of $G^D$. By the construction of $e^*$, all of its endpoints share a common vertex $v$ of $G$. Let $X_i$ be some set in $X$ containing $v$. From the definition of $Y_i$, all the edges in $e^*$ will be members of $Y_i$ and condition (2) holds. For any two vertices $i, j$ of $U$ we denote as $P(i, j)$ the vertices of the path connecting them in $U$. For condition (3), let $e$ be a vertex of $G^D$ such that $e \in Y_i$ and $e \in Y_j$ for two different vertices $i, j$ of $U$. It is sufficient to prove that for any vertex $h \in P(i, j)$, $e \in Y_h$. From the definition of $Y_i$, $e$ has an endpoint $v_e \in V(G)$ that belongs to $X_i$. Similarly, $e$ has an endpoint $u_e \in V(G)$ that belongs to $X_j$. We consider two cases. If $v_e = u_e$, then from condition (3) for $(X, Y)$, we get that $v$ belongs to any $X_h$ where $h \in P(i, j)$. From the definition of $Y$, we have that, since $v_e = u_e$ is an endpoint of $e$, $e$ belongs also to any $Y_h$ for any vertex $h \in P(i, j)$. From condition (2) for $(X, U)$, we have that there exists a vertex $k$ of $U$ where $v_e, u_e \in X_k$. Clearly, $k$ should be a vertex in $P(i, j)$ in $U$ as, otherwise, either $v_e \in X_j$ or $u_e \in X_i$, a contradiction. Let $h$ be any vertex in $P(i, k)$. As $v_e$ belongs both to $X_i$ and to $X_k$, condition (2) for $(X, U)$ implies that $v_e \in X_h$ and from the definition of $Y_h$ we have that $e \in Y_h$. Finally, if $h \in P(k, j)$, then applying the same argument for this path we can conclude that $e \in Y_h$ and condition (3) holds for $(Y, U)$. □

Notice that the proof of the above lemma gives a method to compute $(Y, U)$ from $(X, U)$ in $O(kn\Delta(G))$ steps.

The notion of linear-width for graphs was introduced by Thomas [47].

The linear-width of a graph $G$ with $n$ vertices is defined as follows. Let $l = [e_1, \dots, e_r]$ be a linear ordering of $E(G)$. For $i = 1, \dots, r-1$, we define $\zeta_{l,G}(i) = V_G(l[1,i]) \cap V_G(l[i+1,n])$ (i.e., $\zeta_{l,G}(i)$ is the set of vertices of $G$ that are endpoints of edges in both $l[1,i]$ and $l[i+1,n]$). The linear-width of an ordering $l$ of $E(G)$ is $\max_{1 \leqslant i \leqslant n-1}\{|\zeta_{l,G}(i)|\}$. The linear-width of a graph is the minimum linear-width over all the orderings of $E(G)$. From the definitions of dual hypergraph and linear-width, we have the following.

**Lemma 23.** *If $G$ is a trunked graph then* linear-width$(G) =$ cutwidth$(G^D)$.

As a consequence of Theorems 21, Lemma 22, and Lemma 23, we have an algorithm for the linear-width of trunked graphs.

**Lemma 24.** *Given a trunked graph $G$ with $n$ vertices of degree no more than $d$ and a tree decomposition $(X, U)$ of $G$ with $O(n)$ nodes and width at most $w$, an ordering of $E(G)$ of minimum linear-width can be computed in $O(dw^3 \log^2 n \cdot ((dw+1)!)^2 2^{12dw} n^{8(w+1)(dw+1)})$ steps.*

**Proof.** Let $G$ be a $n$-vertex graph of treewidth $w$ and $\Delta(G) \leqslant d$. Let also $(U, X)$ be a tree decomposition of $G$. From Lemma 2, we know that the pathwidth of $G$ is at most $(w+1)\log n$ and, as linear-width$(G) \leqslant$ pathwidth$(G) + 1$ (see, e.g., [44] or [45]), we get that linear-width$(G) \leqslant (w+1)\log n + 1$. From Lemma 23 we have that cutwidth$(G^D) =$ linear-width$(G) \leqslant (w+1)\log n + 1$. Notice that a vertex ordering of $G^D$ with minimum cutwidth corresponds to an edge ordering of $G$ of minimum-linear width. Therefore, it is sufficient to check whether cutwidth$(G^D) \leqslant k$ for $k = 1, \dots, \lfloor (w+1)\log n + 1 \rfloor$ and output the vertex ordering corresponding to the minimum $k$ for which the result of this check is positive. To do this, we use the construction of Lemma 22, and get a tree decomposition $(Y, U)$ of $G^D$ with treewidth $\leqslant dw$. From Theorem 21, this check requires

$$O\left(dw^3 \log^2 n \cdot \big((dw+1)!\big)^2 \left(\frac{8}{3}\right)^{2dw} 2^{((w+1)\log n + 1)(8dw+8)}\right)$$
$$= O\left(dw^3 \log^2 n \cdot \big((dw+1)!\big)^2 \left(\frac{8}{3}\right)^{2dw} n^{(w+1)(8dw+8)} 2^{8dw}\right)$$

steps, and the claimed result follows. □

For a proof of the following, see [4].

**Lemma 25.** *If $G^n$ is the graph obtained from $G$ by replacing every edge in $G$ with two edges in parallel, then* pathwidth$(G) =$ linear-width$(G^n)$.

It is easy to derive a procedure that given an edge ordering of a graph $G$ with linear-width at most $k$, transforms it to a path decomposition of width at most $k$ in $O(kn)$ steps (e.g., see [4,12]). Also observe that $G^n$ is a trunked graph. These facts along with Lemmata 24 and 25 yield the following result.

**Theorem 26.** *Given a graph $G$ with $n$ vertices of degree no more than $d$ and a tree decomposition $(X, U)$ of $G$ of $O(n)$ nodes and width at most $w$, a path decomposition of $G$ with minimum width can be constructed in $O(dw^3 \log^2 n \cdot ((dw+1)!)^2 2^{12dw} n^{8(w+1)(dw+1)})$ steps.*

We mention that, in general, the problem of computing the pathwidth of partial $w$-trees can be solved in polynomial time. The algorithm for the general case was proposed by Bodlaender and Kloks in [11]. However, the exponent in the complexity of this algorithm is quite large for any practical purpose. The algorithm proposed in Theorem 26 is faster and can serve as a more realistic approach for partial $w$-trees with bounded degree.

## 7. Open problems

We have shown that the cutwidth of graphs with bounded treewidth and maximum degree can be computed in polynomial time. The problem that remains is to prove the same when the "bounded maximum degree" requirement is removed. Even if this is the case for pathwidth [11], it seems that our technique cannot be easily modified to solve the general problem because it is strongly depending on Lemma 3. However, even in the case of computing the pathwidth of partial $w$-trees, it is interesting to find realistic polynomial algorithms. Another line of research is to try to solve the problem for specific (typically small) values of the treewidth $w$. No algorithm of this type exists for cutwidth when $w > 1$, while, for pathwidth, the best known results are approximation algorithms for outerplanar graphs [9,27] and Halin graphs [24] (see also [43]).

## Acknowledgment

We wish to thank one of the anonymous referees for his detailed comments and remarks that helped us improving the overall presentation of this work.

## References

[1] D. Adolphson, Single machine job sequencing with precedence constraints, SIAM J. Comput. 6 (1977) 40–54.
[2] D. Adolphson, T.C. Hu, Optimal linear ordering, SIAM J. Appl. Math. 25 (3) (1973) 403–423.
[3] S. Arora, A. Frieze, H. Kaplan, A new rounding procedure for the assignment problem with applications to dense graphs arrangements, in: Proc. 37th IEEE Symposium on Foundations of Computer Science, IEEE, 1996, pp. 21–30.
[4] D. Bienstock, P. Seymour, Monotonicity in graph searching, J. Algorithms 12 (2) (1991) 239–245.
[5] H.L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (6) (1996) 1305–1317.
[6] H.L. Bodlaender, Treewidth: algorithmic techniques and results, in: Mathematical Foundations of Computer Science 1997, 22nd International Symposium, MFCS'97, Proceedings, in: Lecture Notes in Comput. Sci., vol. 1295, Springer-Verlag, 1997, pp. 19–36.

[7] H.L. Bodlaender, A partial $k$-arboretum of graphs with bounded treewidth, Theoret. Comput. Sci. 209 (1–2) (1998) 1–45.
[8] H.L. Bodlaender, M.R. Fellows, D.M. Thilikos, Starting with nondeterminism: The systematic derivation of linear-time graph layout algorithms, in: Proc. 26th International Symposium on Mathematical Foundations of Computer Science, MFCS 2003, in: Lectures Notes in Comput. Sci., vol. 2747, Springer-Verlag, 2003, pp. 239–248.
[9] H.L. Bodlaender, F.V. Fomin, Approximation of pathwidth of outerplanar graphs, J. Algorithms 43 (2) (2002) 190–200.
[10] H.L. Bodlaender, J.R. Gilbert, H. Hafsteinsson, T. Kloks, Approximating treewidth, pathwidth, frontsize, and shortest elimination tree, J. Algorithms 18 (2) (1995) 238–255.
[11] H.L. Bodlaender, T. Kloks, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, J. Algorithms 21 (1996) 358–402.
[12] H.L. Bodlaender, D.M. Thilikos, Computing small search numbers in linear time, Technical Report UU-CS-1998-05, Department of Computer Science, Utrecht University, 1998.
[13] R.A. Botafogo, Cluster analysis for hypertext systems, in: 16th Annual ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 1993, pp. 116–125.
[14] F.R.K. Chung, On the cutwidth and the topological bandwidth of a tree, SIAM J. Algebraic Discrete Methods 6 (2) (1985) 268–277.
[15] F.R.K. Chung, P.D. Seymour, Graphs with small bandwidth and cutwidth, Discrete Math. 75 (1–3) (1989) 113–119.
[16] M.J. Chung, F. Makedon, I.H. Sudborough, J. Turner, Polynomial time algorithms for the MIN CUT problem on degree restricted trees, SIAM J. Comput. 14 (1) (1985) 158–177.
[17] J. Díaz, M. Penrose, J. Petit, M. Serna, Approximating layout problems on random geometric graphs, J. Algorithms 39 (1) (2001) 78–117.
[18] J. Díaz, M.D. Penrose, J. Petit, M.J. Serna, Convergence theorems for some layout measures on random lattice and random geometric graphs, Combin. Probab. Comput. 9 (2000) 489–511.
[19] J. Díaz, J. Petit, M. Serna, A survey on graph layout problems, ACM Comput. Surveys 34 (3) (2002) 313–356.
[20] J. Díaz, J. Petit, M. Serna, L. Trevisan, Approximating graph layout problems on random graphs, Discrete Math. 235 (2001) 245–253.
[21] G. Even, J. Naor, S. Rao, B. Schieber, Divide-and-conquer approximation algorithms via spreading metrics, in: 36th Proc. Foundations of Computer Science, IEEE, 1995, pp. 62–71.
[22] M.R. Fellows, M.A. Langston, On well-partial-order theory and its application to combinatorial problems of VLSI design, SIAM J. Discrete Math. 5 (1) (1992) 117–126.
[23] M.R. Fellows, M.A. Langston, On search, decision, and the efficiency of polynomial-time algorithms, J. Comput. System Sci. 49 (3) (1994) 769–779.
[24] F.V. Fomin, D.M. Thilikos, A 3-approximation for the pathwidth of Halin graphs, in: Cologne Twente Workshop on Graphs and Combinatorial Optimization, CTW 2004, 2004.
[25] M.R. Garey, D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness, Freeman, San Francisco, CA, 1979.
[26] F. Gavril, Some NP-complete problems on graphs, in: Proc. 11th Conference on Information Sciences and Systems, John Hopkins Univ., Baltimore, 1977, pp. 91–95.
[27] R. Govindan, M.A. Langston, X. Yan, Approximating the pathwidth of outerplanar graphs, Inform. Process. Lett. 68 (1) (1998) 17–23.
[28] L.H. Harper, Optimal assignments of number to vertices, SIAM J. 12 (1) (1964) 131–135.
[29] D.R. Karger, A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem, SIAM J. Comput. 29 (2) (1999) 492–514.
[30] E. Korach, N. Solel, Tree-width, path-width, and cutwidth, Discrete Appl. Math. 43 (1) (1993) 97–101.
[31] J. Lagergren, Efficient parallel algorithms for graphs of bounded tree-width, J. Algorithms 20 (1) (1996) 20–44.
[32] T. Lengauer, Upper and lower bounds on the complexity of the min-cut linear arrangement problem on trees, SIAM J. Algebraic Discrete Methods 3 (1) (1982) 99–113.
[33] F.S. Makedon, C.H. Papadimitriou, I.H. Sudborough, Topological bandwidth, SIAM J. Algebraic Discrete Methods 6 (3) (1985) 418–444.

[34] J. Matoušek, R. Thomas, Algorithms finding tree-decompositions of graphs, J. Algorithms 12 (1) (1991) 1–22.
[35] B. Monien, I.H. Sudborough, Min cut is NP-complete for edge weighted trees, Theoret. Comput. Sci. 58 (1–3) (1988) 209–229.
[36] P. Mutzel, A polyhedral approach to planar augmentation and related problems, in: P. Spirakis (Ed.), European Symposium on Algorithms, in: Lecture Notes in Comput. Sci., vol. 979, Springer-Verlag, 1995, pp. 497–507.
[37] K. Nakano, Linear layout of generalized hypercubes, in: Advances in Parallel and Distributed Computational Models, Ft. Lauderdale, FL, 2002, Internat. J. Found. Comput. Sci. 14 (1) (2003) 137–156.
[38] B.A. Reed, Finding approximate separators and computing tree width quickly, in: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing, ACM Press, 1992, pp. 221–228.
[39] N. Robertson, P.D. Seymour, Graph minors. I. Excluding a forest, J. Combin. Theory Ser. B 35 (1) (1983) 39–61.
[40] N. Robertson, P.D. Seymour, Graph minors. III. Planar tree-width, J. Combin. Theory Ser. B 36 (1) (1984) 49–64.
[41] N. Robertson, P.D. Seymour, Graph minors—a survey, in: Surveys in Combinatorics 1985, Glasgow, 1985, in: London Math. Soc. Lecture Note Ser., vol. 103, Cambridge Univ. Press, Cambridge, 1985, pp. 153–171.
[42] F. Shahrokhi, O. Sýkora, L.A. Székely, I. Vrťo, On bipartite drawings and the linear arrangement problem, SIAM J. Comput. 30 (6) (2001) 1773–1789.
[43] K. Skodinis, Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time, J. Algorithms 47 (1) (2003) 40–59.
[44] A. Takahashi, S. Ueno, Y. Kajitani, Mixed searching and proper-path-width, Theoret. Comput. Sci. 137 (2) (1995) 253–268.
[45] D.M. Thilikos, Algorithms and obstructions for linear-width and related search parameters, Discrete Appl. Math. 105 (2000) 239–271.
[46] D.M. Thilikos, M.J. Serna, H.L. Bodlaender, Cutwidth I: A linear time fixed parameter algorithm, J. Algorithms 56 (1) (2005) 1–24.
[47] R. Thomas, Tree-decompositions of graphs, Lecture notes, School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332, USA, 1996.
[48] M. Yannakakis, A polynomial algorithm for the min-cut linear arrangement of trees, J. ACM 32 (4) (1985) 950–988.