Alcom-FT Technical Report Series
ALCOMFT-TR-01-115

A polynomial time algorithm for the cutwidth of bounded degree graphs with small treewidth*

Dimitrios M. Thilikos, Maria J. Serna

Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Campus Nord – Mòdul C5, c/Jordi Girona Salgado 1-3, 08034 Barcelona, Spain

E-mail: {sedthilk, mjserna}@lsi.upc.es

and

Hans L. Bodlaender

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

E-mail: hansb@cs.uu.nl

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 linear layout $[v_1, \ldots, v_n]$ in a way that, for every $i = 1, \ldots, n-1$, there are at most k edges with the 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 the corresponding optimal layout. Also, it is the main subroutine of an algorithm computing the pathwidth of a bounded degree partial w-tree in $n^{O((wd)^2)}$ steps.

Key Words: cutwidth, treewidth, pathwidth

1. INTRODUCTION

A wide variety of optimization problems can be formulated as layout problems. In most of the cases, a graph layout problem asks for the optimal value of some function

* The work of all the authors was supported by the IST Program of the EU under contract number IST-99-14186 (ALCOM-FT). The work of the first author was partially supported by the Ministry of Education and Culture of Spain, Grant number MEC-DGES SB98 0K148809.

defined over all the linear arrangements of the vertices or the edges of a graph (for a survey, see [20]). One of the most known problems of this type, consists in 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 [2, 3, 33, 31], network reliability [28], automatic graph drawing [42, 36], and information retrieval [15]. Cutwidth has been extensively examined [16, 22, 23, 29, 33, 35, 47] and it appears to be closely related with other graph parameters like pathwidth, linear-width, bandwidth, and modified bandwidth [16, 17, 29, 32, 33]. Briefly, the *cutwidth* of a graph $G = (|V(G)|, |E(G)|)$ is equal to the minimum k for which there exists a vertex layout of G such that the edges joining opposite-side vertices, with respect to any of the "gaps" of the linear layout is at most k. Computing cutwidth is an NP-complete problem [24, 25] and it remains NP-complete even if the input is restricted to planar graphs with maximum degree 3 [35] (see also [32, 19]). It is approximable within $O(\log |V(G)| \log\log |V(G)|)$ [21] and admits a polynomial time approximation scheme if $E(G) = \Theta(|V(G)|^2)$ [4]. No much progress has been note on the problem of detecting special graph classes where computing cutwidth can be done in polynomial time. In [18] it was given an algorithm that, in $O(n(\log n)^{d-2})$ steps, computes the cutwidth of any tree with maximum degree bounded by d. This result was improved in 1983 by Yannakakis [47], who presented an $O(n \log n)$ algorithm computing the cutwidth of any tree. Since then, the only polynomial algorithms reported for the cutwidth of graph classes different than trees, concerned special cases such as hypercubes [27] and b-dimensional c-ary cliques [37]. In this paper, we move one step further presenting an 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 [9]). The pathwidth of a graph is defined as treewidth, by restricting the tree in its definition to be a simple line. That way, treewidth can be seen is 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 taking profit of the "tree" or "line" structure of the input graph (see e.g. [8]). 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. Our algorithm is an non trivial extension of the linear time algorithm in [45] concerning the parameterized version of the cutwidth problem.

The parameterized version of the cutwidth problem asks whether the cutwidth of a graph is at most k, where k is a fixed small constant. This problem is known to be solvable in polynomial time. In particular, the first polynomial algorithm for k fixed was given by Makedon and Sudborough in [33] where a $O(n^{k-1})$ dynamic programming algorithm is described. This time complexity has been considerably improved by Fellows and Langston in [23] where, among others, they prove that, for any fixed k, a $O(n^3)$ algorithm can be constructed checking whether a graph has cutwidth at most k. Furthermore, a technique introduced in [22] (see also [6]) further reduced the bound to $O(n^2)$, while in [1] a general method is given to construct a linear time algorithm that decides whether a given graph has cutwidth at most k, for k constant. Finally, in [45], an explicit constructive linear time algorithm was presented able to output the optimal layout in case of a positive answer. This algorithm is based on the fact that graphs with small cutwidth have also small pathwidth and develops further the techniques in [12, 13, 14] in order to use a bounded-width path decomposition for computing the cutwidth of G.

This paper extends the algorithm in [45] in the sense that it uses all of its subroutines and it solves the problem of [45] for graphs with bounded treewidth. Although this extension is not really useful for the parameterized version of cutwidth, it appears that it is useful for solving the *general* cutwidth problem for partial w-trees of bounded degree. This is possible due to the observation that the "hidden constants" of all the subroutines of 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 2.1), our algorithm is able to compute in $n^{O(w^2 d)}$ steps the cutwidth of bounded degree partial w-trees.

The main subroutine of our paper is Algorithm Join-Node in Section 3 that takes profit of the "small treewidth" property of the input graph. Section 2, contains the main definitions and lemmata supporting the operation of Join-Node as well as the proof of its correctness. Subsections 2.1 and 2.2 contain the definitions of treewidth, pathwidth and cutwidth. Most of the preliminary results of Subsection 2.3, concern operations on sequences of integers and the definitions of the most elementary of them was introduced in [45] and [12] (see also [13, 14]). Also, the main tool for exploiting the small treewidth of the input graph is the notion of the *characteristic of a layout*, introduced in [45] and defined in Subsection 2.4 of this paper. For the above reasons, we use notation compatible with the one used in [45].

Algorithm Join-Node is able to give an algorithm only for computing the cutwidth of a bounded degree partial w-tree G but not for *constructing* the corresponding layout. In section 4, we describe how to transform this algorithm to a constructive one in the sense that it now outputs a linear arrangement of G with optimal cutwidth. This uses the analogous constructions of [45] and the procedures Join-Orderings and Construct-Join-Orderings described in Section 3.

An interesting consequence of our result is that the pathwidth of bounded degree partial w-trees can be computed in $n^{O((wd)^2)}$ steps. We mention that the existence of a polynomial time algorithm for this problem, without the degree restriction, has been proved in [12]. However, the time complexity of the involved algorithm appears to be very large and has not been reported. Our technique, described in Section 5, 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.

In the description of our algorithm we will use the two main subroutines of [45]. Towards making the paper shelf-contained, we present them in the Appendix along with the theorems supporting their correctness, proved in [45].

2. DEFINITIONS AND PRELIMINARY RESULTS

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 $V(G)$ ($E(G)$). 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 proceed with a number of definitions and notations, dealing with finite sequences (i.e., ordered sets) of a given finite set $\mathcal{O}$ (most of the notation in this paper is taken from [45] and [12]). For our purposes, $\mathcal{O}$ can be a set of numbers, sequences of numbers, vertices, or vertex sets. Let ω be a sequence of elements from $\mathcal{O}$. We use the notation $[\omega_1, \ldots, \omega_r]$ to represent ω and we define $\omega[i, j]$ as the subsequence $[\omega_i, \ldots, \omega_j]$ of ω (in case $j < i$, the result is the empty subsequence $[\,]$). We also denote as $\omega(i)$ the element of ω indexed by i.

Given a set S containing elements of $\mathcal{O}$, we denote as $\omega[S]$ the subsequence of ω that contains only the elements of ω that are in S. Given two sequences ω^1, ω^2, defined on $\mathcal{O}$, where $\omega^i = [\omega^i_1, \ldots, \omega^i_{r_i}], i = 1, 2$ we define the *concatenation* of ω_1 and ω_2 as $\omega^1 \oplus \omega^2 = [\omega^1_1, \ldots, \omega^1_{r_1}, \omega^2_1, \ldots, \omega^2_{r_2}]$. Unless mentioned otherwise, we will always consider that the first element of a sequence ω is indexed by 1, i.e. $\omega = \omega[1, |\omega|]$.

Let G be a graph and $S \subseteq V(G)$. We call the graph $(S, E(G) \cap \{\{x, y\} \mid x, y \in S\})$ the *subgraph of G induced by S* and we denote it by $G[S]$. We denote by $E_G(S)$ the set of edges of G that have an endpoint in S; we also set $E_G(v) = E_G(\{v\})$ for any vertex v. If $E \subseteq E(G)$ then we denote as $V_G(E)$ the set of all the endpoints of the edges in E i.e. we set $V_G(E) = \cup_{e \in E} e$. The neighborhood of a vertex v in graph G is the set of vertices in G that are adjacent to v in G and we denote it as $N_G(v)$, i.e. $N_G(v) = V_G(E_G(v)) - \{v\}$. If l is a sequences of all the vertices of G without repetitions, then we will call it *vertex ordering of G*. If l is a vertex ordering of G, the *rank* of a vertex $u \in V(l)$ is its position in the ordering, and we denote it by $\text{rank}(u)$.

2.1. Treewidth – Pathwidth

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 $i \in V(I)$ 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, 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. [12].

LEMMA 2.1. *For any constant $k \geq 1$, given a tree decomposition of a graph G of width $\leq k$ and $O(|V(G)|)$ nodes, there exists an algorithm that, in $O(|V(G)|)$ steps, constructs a nice tree decomposition of G of width $\leq k$ and with at most $4|V(G)|$ nodes.*

We now observe that a nice tree decomposition $(\{X_i \mid i \in V(U)\}, U)$ contains nodes of the following four possible types. A node $i \in V(U)$ is called "*start*" if $i \in A(U)$, "*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 $V_i = \cup_{v \in V(U_i)} X_v$. Also, for any $p \in V(U)$ we define G_p in a top down fashion as follows. If p is the root, then $G_p = G$. If p has as parent an *introduce* node q, then $G_p = G[V(G_q) - \{X_q - X_p\}]$. If p has as parent a *forget* node q then $G_p = G_q$. If p has as parent a *join* node q whose other child is p' then we set $G_p = (V(G_q), E(G_q))$ and $G_{p'} = (V(G_q), \emptyset)$ (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 $V(G_p) = V(G_{p'}) = V(G_q)$, $E(G_p) \cap E(p') = \emptyset$, 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 tree decomposition of G_i.

The parameter of pathwidth is defined as the one of treewidth, with the difference that the tree U in a path decomposition is a line (tree with maximum degree 2).

We will need the following result from [11].

LEMMA 2.2. *For any graph G,* $\text{treewidth}(G) \leq \text{pathwidth}(G) \leq (\text{treewidth}(G)+1) \cdot \log(|V(G)|)$.

2.2. Cutwidth

The cutwidth of a graph G with n vertices is defined as follows. Let $l = [v_1, \ldots, v_n]$ be a layout of $V(G)$. For $i = 1, \ldots, n-1$, we define $\theta_{l,G}(i) = E_G(l[1,i]) \cap E_G(l[i+1,n])$ (i.e. $\theta_{l,G}(i)$ is the set of edges of G that have one endpoint in $l[1,i]$ and one in $l[i+1,n]$). The cutwidth of an ordering l of $V(G)$ is $\max_{1 \leq i \leq n-1}\{|\theta_{l,G}(i)|\}$. The cutwidth of a graph is the minimum cutwidth over all the orderings of $V(G)$. It is easy to see the following (see also [33]).

LEMMA 2.3. *For any graph G, with vertices of degree bounded by d,* $\text{pathwidth}(G) \leq \text{cutwidth}(G) \leq d \cdot \text{pathwidth}(G))$.

We will use the notation introduced in [45] (see also [13, 14]).

If $l = [v_1, \ldots, v_n]$ is a vertex ordering of a graph G, we set

$$\mathbf{Q}_{G,l} = [[0], [|\theta_{l,G}(1)|], \ldots, [|\theta_{l,G}(n-1)|], [0]].$$

We also assume that the indices of the elements of $\mathbf{Q}_{G,l}$ start from 0 and finish on n, i.e. $\mathbf{Q}_{G,l} = \mathbf{Q}_{G,l}[0,n]$. Clearly, $\mathbf{Q}_{G,l}$ is a sequence of sequences of numbers each containing only one element. For an example, see Figure 3.

The following is a direct consequence of Lemmata 2.2 and 2.3.

COROLLARY 2.1. *Any graph G with treewidth at most w and maximum degree at most d, has cutwidth bounded by $(w+1)d \log |V(G)|$.*

2.3. Sequences of integers

We denote as $\mathcal{S}$ the set of all the sequences of non-negative integers. For any sequence $A = [a_1, \ldots, a_{|A|}] \in \mathcal{S}$ and any integer $t \geq 0$ we set $A + t = [a_1 + t, \ldots, a_{|A|} + t]$. If

$A, B \in \mathcal{S}$ and $A = [a_1, \ldots, a_{|A|}]$, we say that $A \sqsubseteq B$ if B is a subsequence of A obtained after applying a number of times (possibly none) the following operations.

(i) If for some i, $1 \leq i \leq |A| - 1$ $a_i = a_{i+1}$, then set $A \leftarrow A[1, i] \oplus A[i + 2, |A|]$.

(ii) If the sequence contains two elements a_i and a_j such that $j - i \geq 2$ and $\forall_{i<k<j}\ a_i \leq a_k \leq a_j$ or $\forall_{i<k<j}\ a_i \geq a_k \geq a_j$, then set $A \leftarrow A(1, i) \oplus A(j, |A|)$.

We define the *compression* $\tau(A)$ of a sequence $A \in \mathcal{S}$, as the unique minimum length element of $\{B \mid B \sqsubseteq A\}$. For example, $\tau([\mathbf{5},5,6,7,7,\mathbf{7},4,4,\mathbf{3},5,4,6,\mathbf{8},\mathbf{2},\mathbf{9},3,4,6,7,\mathbf{2},\mathbf{7},5,4,4,6,\mathbf{4}]) = [5,7,3,8,2,9,2,7,4]$.

We call a sequence A *typical* if $A \in \mathcal{S}$ and $\tau(A) = A$.

The following results have been proved in [12] (Lemmata 3.3 and 3.5 respectively).

LEMMA 2.4. *If $A \in \mathcal{S}$ and $\max A \leq k$, then $\tau(A)$ contains at most $2k - 1$ elements.*

LEMMA 2.5. *The number of different typical sequences consisting of integers in $\{0, 1, \ldots, n\}$ is at most $\frac{8}{3}2^{2n}$.*

Notice that $B = \tau(A)$ is a subsequence $[a_{i_1}, \ldots, a_{i_{|B|}}]$ of $A = [a_1, \ldots, a_{|A|}]$ such that for any j, $1 \leq j \leq |B| - 1$ either $a_{i_j} \leq a_{i_j+1} \leq \cdots \leq a_{i_{j+1}-1} \leq a_{i_{j+1}}$ or $a_{i_j} \geq a_{i_j+1} \geq \cdots \geq a_{i_{j+1}-1} \geq a_{i_{j+1}}$. We can now define a function $\beta_A : \{1, \ldots, |\tau(A)|\} \to \{1, \ldots, |A|\}$ where $\beta_A(j) = i_j$ is one of the possible original positions in A of the j-th element in $\tau(A)$. Consider the sequence of the previous example

$$A = [\mathbf{5}, 5, 6, \underline{7, 7, \mathbf{7}, 7}, 4, 4, \mathbf{3}, 5, 4, 6, \mathbf{8}, \mathbf{2}, \mathbf{9}, 3, 4, 6, 7, \mathbf{2}, \mathbf{7}, 5, 4, 4, 6, \mathbf{4}],$$

then we have

$$\begin{array}{lll} \beta_A(1) = 1, & \beta_A(2) = 6 \text{ (or 4 or 5 or 7)}, & \beta_A(3) = 10, \\ \beta_A(4) = 14, & \beta_A(5) = 15, & \beta_A(6) = 16, \\ \beta_A(7) = 21, & \beta_A(8) = 22, & \beta_A(9) = 27. \end{array}$$

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

LEMMA 2.6. *Let A be any sequence in $\mathcal{S}$. Then for any $i, 1 \leq i < |\tau(A)|$, $\tau(A[\beta_A(i), \beta_A(i+1)]) = [A(\beta_A(i)), A(\beta_A(i+1))]$*

Analogously, we define the function $\beta_A^{-1} : \{1, \ldots, |A|\} \to \{1, \ldots, \tau(A)\}$ such that $\beta_A^{-1}(j)$ is the unique i such that there exists a function β_A where $\beta_A(i) = j$.

For any $A \in \mathcal{S}$, we define $\alpha(A)$ in the same way as $\tau(A)$ with the difference that only operation (i) is considered, i.e., we remove repetitions of a number on successive positions in the sequence. If now A is a typical sequence, we define the *set of extensions* of A as

$$\mathcal{E}(A) = \{\tilde{A} \in \mathcal{S} \mid \alpha(\tilde{A}) = A\}.$$

We call a sequence A *dense* if $A \in \mathcal{E}(\tau(A))$. If A is dense then all the sequences in $\mathcal{E}(A)$ are dense. Finally, notice also that if A is dense and $B \sqsubseteq A$ then $B \in \mathcal{E}(\tau(A))$.

Notice that for any typical sequence A, if $B \in \mathcal{E}(A)$, $B(i) \neq B(i+1)$, and $\beta_B^{-1}(i) = j$, then the subsequence $B[i, i+1]$ represents the j-th number change in B.

The results in the following two lemmata are direct consequences of the definitions of β and β^{-1}.

LEMMA 2.7. *Let A be a sequence in S, and $B \in \mathcal{E}(\tau(A))$ then, for any $i, 1 \leq i \leq |B|$,*

1. $A(\beta_A(\beta_B^{-1}(j))) = B(j)$,

2. $B(j) \neq B(j+1) \Rightarrow \beta_B^{-1}(j+1) = \beta_B^{-1}(j) + 1$, *and*

3. $B(j) = B(j+1) \Rightarrow \beta_B^{-1}(j+1) = \beta_B^{-1}(j)$

Let $A = [a_1, \ldots, a_{r_1}]$ and $B = [b_1, \ldots, b_{r_2}]$ be two sequences in $\mathcal{S}$. We say that $A \leq B$ if $r_1 = r_2$ and $\forall_{1 \leq i \leq r_1}\ a_i \leq b_i$. In general, we say that $A \prec B$ if there exist extensions $\tilde{A} \in \mathcal{E}(A)$, and $\tilde{B} \in \mathcal{E}(B)$ such that $\tilde{A} \leq \tilde{B}$. For example if $A = [1, 7, 2, 6, 4]$ and $B = [5, 7, 3, 8]$ then $A \prec B$ because $\tilde{B}$ = [5,7,7,7,4,8,8,8,8] is an extension of B, $\tilde{A}$ = [1,7,2,6,4,4,4,4,4] is an extension of A, and $\tilde{A} \leq \tilde{B}$.

The following lemma corresponds to Corollary 3.11 of [12]

LEMMA 2.8. *If A and B are two sequences then $A \prec B$ if and only if $\tau(A) \prec \tau(B)$.*

Suppose now that $\mathbf{A} = [A_1, \ldots, A_r]$ and $\mathbf{B} = [B_1, \ldots, B_{|r|}]$ are two sequences of typical sequences. We say that $\mathbf{A} \prec \mathbf{B}$ if $\forall_{1 \leq i \leq r}\ A_i \prec B_i$. For any integer t we set $\mathbf{A} + t = [A_1 + t, \ldots, A_{|\mathbf{A}|} + t]$ and $\max(\mathbf{A}) = \max_{1 \leq i \leq |\mathbf{A}|}\{\max A_i\}$. Finally, for any

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

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

sequence of typical sequences $\mathbf{A}$ we set $\tau(\mathbf{A}) = \tau(\mathbf{A}(1) \oplus \cdots \oplus \mathbf{A}(|\mathbf{A}|))$. As an example,

$$
\begin{aligned}
\tau([[\mathbf{5},\mathbf{2},\mathbf{8},\mathbf{1}],[4,9,3],[3],[3,\mathbf{9},\mathbf{2},\mathbf{5},\mathbf{3}]]) &= \tau([5,2,8,1,4,9,3,3,3,9,2,5,3])\\
&= [5,2,8,1,9,2,5,3].
\end{aligned}
$$

Let two equal-size sequences A, B of $\mathcal{S}$ 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 $\forall_{1 \le i < r}\ a_i \neq a_{i+1} \Leftrightarrow b_i = b_{i+1}$ (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

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

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 [12]). 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 Figure 1.

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

$$F_{p,q} = (\{(i,j) \mid 1 \le i \le p, 1 \le j \le q\}, \{((i,j),(i',j')) \mid i \le i', j \le j', i' - i + j' - j = 1\})$$

whose vertex (i,j) is labeled 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

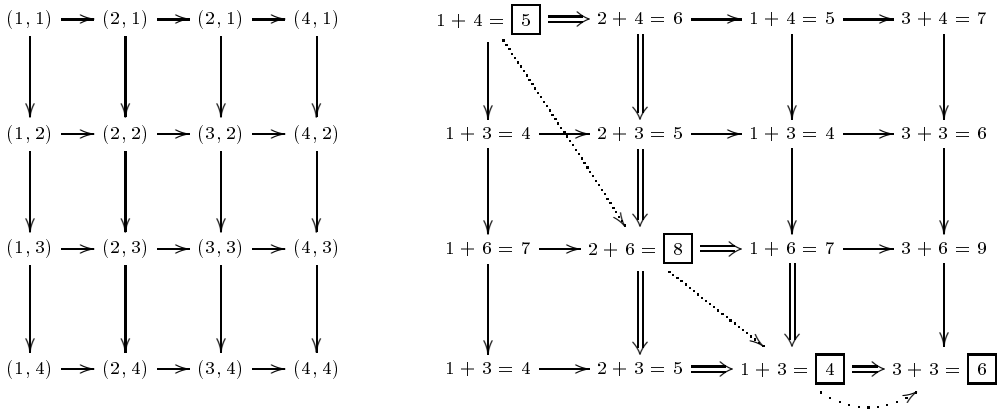

FIG. 2. An example of $F_{4,4}$ and its labeling when $A = [1, 2, 1, 3]$ and $B = [4, 3, 6, 3]$. The double-lined arrows define the two paths in $G_{4,4}$ that can generate the typical sequence $[5, 8, 4, 6]$.

$F_{p,q}$ and the sums $\tilde{A}+\tilde{B}$ where $\tilde{A} \in \mathcal{E}(A), \tilde{B} \in \mathcal{E}(B)$ and, $\tilde{A} \sim \tilde{B}$. For an example, if $A = [1, 2, 1, 3]$ and $B = [4, 3, 6, 3]$, then the sum $\tilde{A} + \tilde{B}$ where $\tilde{A} = [\mathbf{1}, 2, 2, \mathbf{2}, 2, \mathbf{1}, \mathbf{3}]$ and $\tilde{B} = [\mathbf{4}, 4, 3, \mathbf{6}, 3, \mathbf{3}, \mathbf{3}]$ corresponds to the path $((1, 1), (2, 1), (2, 2), (2, 3), (2, 4), (3, 4), (4, 4))$ (see Figure 2).

If a is a positive integer, we call a directed acyclic graph G *a-layered* if there exists an ordered partition $(V_1, \ldots, V_r)$ of its vertices such that

1. $V_1 = \{v\}$ and $V_r = \{u\}$.
2. For each edge (x, y) there exists a $i, 1 \leq i < r$ such that $x \in V_i$ and $y \in V_{i+1}$.
3. For each $i, 1 \leq i \leq r$, $|V_i| \leq a$.
4. Any vertex $w \in V(G)$, belongs to a path connecting v and u.

The following lemma is an easy exercise.

LEMMA 2.9. *Let G be an edge-weighted a-layered directed acyclic graph. If each of the paths of G has total cost $\leq b$, then the total cost of the edges of G is $\leq a^2 b$.*

LEMMA 2.10. *One can construct an algorithm that for any typical sequences A, B, C, where $\max A \oplus B \leq k$, checks whether $C \in A \otimes B$ in $O(k^4)$ steps.*

Proof. Let $A = [a_1, \ldots, a_p]$, $B = [b_1, \ldots, b_q]$ and $C = [c_1, \ldots, c_r]$. We can further assume that $c_1 = a_1 + b_1$ and $c_r = a_p + b_q$ (otherwise, $C \notin A \otimes B$).

We construct a directed graph $G = (V, E)$ applying three steps as follows:

Step 1.

1. $V_{1,2} = \{(1,1)\}$,
2. For $l = 2, \ldots, r$ do

   (i) $V_{l,l+1} = \{(i,j) \mid 1 \leq i \leq p, 1 \leq j \leq q, a_i + b_j = c_l\}$

   (ii) $E_{l-1,l} = \{((i,j),(i',j')) \mid (i,j) \in V_{l-1,l}, (i',j') \in V_{l,l+1}, i \leq i', j \leq j'\}$

3. Set $V = V_{1,2} \cup \cdots, V_{r,r+1}$ and $E = E_{1,2} \cup \cdots E_{r-1,r}$.

Notice that the same number cannot appear more than two times in a typical sequence. Therefore, for each $i \leq p$ the vertices (i,j) of V for which $a_i + b_j$ is the same number of C, are at most 2. We get that, for $l = 1, \ldots, r-1$ $|V_{l,l+1}| \leq 2p \leq 4k-1$ (Lemma 2.4). Observe now that the edges of G connecting vertices in V_l and V_{l+1} are at most $16k^2$. As $\max C \leq 2k$, Lemma 2.4 implies that $r \leq 4k-1$. We conclude that $|E| \leq 64k^3$. Therefore, G can be constructed in $O(k^3)$ steps.

By the way the edges of G are constructed, we have that G is a directed acyclic graph. We further transform G as follows.

Step 2. For any $(i,j) \in V$, if there is not at path connecting $(1,1)$ and (i,j) in G or there is not a path connecting (i,j) and (p,q) then set $G = G[V - \{(i,j)\}]$.

Notice that the identification of all the vertices that should be be removed, can be done in $O(E(G)) = O(k^3)$ steps (e.g. applying back an forward a breath first search of G, starting from (p,q) and $(1,1)$ respectively). Step 2 results to a directed acyclic graph where any vertex in G is in a path connecting $(1,1)$ with (p,q). It is now easy to observe that G is $2k$-layered.

Let $((i,j),(i',j'))$ be any edge of G. We set $\mathbf{f}_{\min} = \min\{a_i + b_j, a_{i'} + b_{j'}\}$ and $\mathbf{f}_{\max} = \max\{a_i + b_j, a_{i'} + b_{j'}\}$. We correspond to $((i,j),(i',j'))$ the graph $F_{i,j,i',j'} = F[V_{i,j,i',j'}]$ where

$$V_{i,j,i',j'} = \{(s,t) \mid i \leq s \leq i', j \leq t \leq j', \mathbf{f}_{\min} \leq a_s + b_t \leq \mathbf{f}_{\max}\}$$

Notice that $F_{i,j,i',j'}$ contains $O(|i'-i| \cdot |j'-j|)$ edges and vertices and its construction can be accomplished in the same time.

Now apply a last transformation on G.

Step 3. Remove from G any edge $((i,j),(i',j'))$ for which there does not exist a path connecting (i,j) and (i',j') in $F_{i,j,i',j'}$.

The existence of such a path can be checked in $O(E(F_{i,j,i',j'})) = O(|i'-i| \cdot |j'-j|)$ steps. In order to determine the total number of steps required by step 3, we assign to

each of the $O(k^3)$ edges of G the time cost of this check. We observe that in the directed acyclic graph G, the sum of the costs of the edges of any path P of G is $O(k^2)$. In order to see this one has to verify that none of the edge sets of graphs $F_{i,j,i',j'}$, F_{i_*,j_*,i'_*,j'_*}, corresponding to any different edges $((i,j),(i',j'))$, $((i_*,j_*),(i'_*,j'_*))$ of P can overlap. Therefore, their total size, determining the total number of steps corresponding to this path, is $O(E(F)) = O(k^2)$. As now $V(G)$ is $2k$-layered, with all the paths of costs $O(k^2)$, Lemma 2.9 implies that the total cost of all the edges of G is $O(k^4)$.

From the analysis above, it follows that all the steps of the construction of G can be computed in $O(k^4)$ steps. As an example, we mention that the graph G for checking whether $C \in A \otimes B$ when $A = [4,3,6,3]$ and $B = [5,8,4,6]$, is the one induced by the dotted arrows in Figure 2. Now, a depth first search of G can report whether $C = A \oplus B$ in $O(|E(G)|) = O(k^3)$ steps and this is correct because of the following claim.

Claim: $C \in A \otimes B$ iff $(1,1)$ and (p,q) are connected by a path in G.

Proof of Claim. Suppose that such a path indeed exists. Notice that any edge $((i,j)(i',j'))$ of this path corresponds to some path $P_{(i,j)(i',j')}$ of F whose non-internal correspond to two consecutive elements c_σ,$c_{\sigma+1}$ of C $(\sigma < r)$. The union of all these paths is a path P of F connecting $(1,1)$ and (p,q). Recall that such a path, corresponds to some sum $\tilde{C} = \tilde{A} + \tilde{B}$ for some $\tilde{A} \in \mathcal{E}(A), \tilde{B} \in \mathcal{E}(B)$ and, $\tilde{A} \sim \tilde{B}$. Moreover, the existence of edge $((i,j)(i',j'))$ implies that all the elements of $\tilde{C}$ between c_σ,$c_{\sigma+1}$ should be removed during the compression of $\tilde{C}$. Applying this argument for any $\sigma < r$ we have that $C \in A \otimes B$.

We now have to show that if $C \in A \otimes B$, a path described by the above algorithm will exists. Notice that C is the compression of a sequence $\tilde{C}$ equal to the sum of two sequences corresponding to a path P in F connecting $(1,1)$ and (p,q). The surviving elements of this sequence are vertices of F that will remain vertices of G during its construction. Notice also that any to vertices (i,j),(i',j') corresponding to consecutive numbers $c_\sigma, c_{\sigma+1}$ in C $(\sigma < r)$ define a part of P whose vertices, in turn, correspond to the numbers of $\tilde{C}$ between c_σ and $c_{\sigma+1}$ that should be removed during its compression. This assures that the $F_{i,i',j,j'}$ contains a path connecting (i,j) and (i',j') and thus (i,j),(i',j') are adjacent in G. Applying the same argument for any $\sigma < r$ we derive the existence of a path in G connecting (i,j) and (i',j'). ∎

The *interleaving* of two sequences of typical sequence $\mathbf{A} = [A_1, \ldots, A_w]$ and $\mathbf{B} = [B_1, \ldots, B_w]$ where $w = |\mathbf{A}| = |\mathbf{B}|$ is defined as follows:

$$\mathbf{A} \bigotimes \mathbf{B} = \{[C_1, \ldots, C_w] \mid C_i \in A_i \otimes B_i, i = 1, \ldots, w\}.$$

LEMMA 2.11. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be two sequences of sequences of integers in* $\{0, \ldots, k\}$, *with* $|\mathbf{A}| = |\mathbf{B}| = w$. *Then* $|\mathbf{A} \bigotimes \mathbf{B}| \leq \frac{8}{3} 2^{4wk}$ *and* $\mathbf{A} \bigotimes \mathbf{B}$ *can be computed in* $O(k^4 2^{4wk})$ *steps.*

Proof. We denote as $\hat{\mathcal{S}}_{2k}$ the set containing all the typical sequences that have numbers in $\{0, \ldots, 2k\}$. From Lemma 2.5, $|\hat{\mathcal{S}}_{2k}| \leq \frac{8}{3} 2^{4k}$. A straightforward enumeration of the elements of $\hat{\mathcal{S}}_{2k}$ can be done in $O(k)$ steps for each element. Therefore, $\hat{\mathcal{S}}_{2k}$ can be generated in $O(k \cdot |\hat{\mathcal{S}}_{2k}|)$ steps. We will denote as A, B any pair of sequences belonging to $\mathbf{A}$ and $\mathbf{B}$ with the same indice. Notice that, for the upper bound, it is enough to prove that $|A \otimes B| \leq \frac{8}{3} 2^{4n}$. Notice that all the sequences in $A \otimes B$ are typical and contain integers in $\{0, \ldots, 2k\}$, therefore we have that $A \otimes B \subseteq \hat{\mathcal{S}}_{2k}$ and the upper bound follows. In order now to compute $\mathbf{A} \bigotimes \mathbf{B}$ we have to show how compute $A \otimes B$. For this, we will need a way to check whether an element C of $\hat{\mathcal{S}}_{2k}$ belongs to $A \otimes B$ or not. From Lemma 2.10, this can be done in $O(k^4)$ steps and the result follows. ▌

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 ν_{B_1,B_2} by ν.

As an example, we consider $B_1 = [6, 6, 9, 9, 9, 1, 1, 1]$ and $B_2 = [1, 8, 8, 2, 7, 7, 3, 5]$. We have

$$\nu(1) = 2 \quad \nu(2) = 1 \quad \nu(3) = 2 \quad \nu(4) = 2 \quad \nu(5) = 1 \quad \nu(6) = 2 \quad \nu(7) = 2$$

The following lemma is a direct consequences of the definitions.

LEMMA 2.12. *Let* $A_i, B_i, i = 1, 2$ *such that* $A_i \sqsubseteq B_i, i = 1, 2$. *Then,*

1.$A_1 \oplus A_2 \sqsubseteq B_1 \oplus B_2$, *and*

2.if $|A_1| = |A_2|$ *then* $\tau(A_1 + A_2) = \tau(B_1 + B_2)$.

The following lemma follows directly from Lemma 2.8 and Lemma 3.13 of [12].

LEMMA 2.13. *Let A and B be two integer sequences where $|A| = |B|$ and let $Y = A + B$. Let $A_0 \prec A$ and $B_0 \prec B$. Then there exists two sequences $A_0^* \in \mathcal{E}(A_0)$ and $B_0^* \in \mathcal{E}(B_0)$ where $\tau(A_0^* + B_0^*) \prec \tau(Y)$.*

The following lemma will be useful in order to adapt some auxiliary results of [12] to our definition of $A \otimes B$.

LEMMA 2.14. *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 $A_1(j) + A_2(j+1) \leq A_1(j+1) + A_2(j)$ 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 $j = 1, \ldots, r-1$,

$$\begin{aligned}
C^*(2j) &= \min\{A_1(j) + A_2(j+1), A_1(j+1) + A_2(j)\} \\
&= \min\{A_1^*(2j-1) + A_2^*(2j+1), A_1^*(2j+1) + A_2^*(2j-1)\} \\
&\leq \max\{A_1^*(2j-1) + A_2^*(2j-1), A_1^*(2j+1) + A_2^*(2j+1)\} \\
&= \max\{A_1(j) + A_2(j), A_1(j+1) + A_2(j+1)\} \\
&= \max\{C(j), C(j+1)\} \qquad (1)
\end{aligned}$$

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

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

Apply now the following operation on A_1^* and A_2^* as long as this is possible: if for some $j, 1 \leq j < |A_1^*|$, $A_i^*(j) = A_i^*(j+1), i = 1, 2$ then set $A_i^* \leftarrow A_i^*(1, j) \oplus A_i^*(j+2, r), i = 1, 2$

(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 2.15. *Let A, B be two typical sequence and C 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 $A' \prec A$, Lemma 2.8 implies that $A' \prec A^*$. Similarly, we get that $B' \prec B^*$. From Lemma 2.13 we have that 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 2.14 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 just a special case of Lemma 3.14 in [12].

LEMMA 2.16. *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 2.17. *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 Lemmata 2.16 and 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 2.14, there exists a typical sequence $C' \in \tau(A') \otimes \tau(B')$ such that $C' \prec \tau(A' + B')$ and the result follows. ▌

2.4. Characteristic pairs

We will now define the notion of the characteristic of a vertex layout of a graph with respect to a small subset of its vertices. Characteristics will serve as key-tools for our algorithm and their definition is the same as in [45]. Several other versions of characteristics have been used for the computation of other parameters like pathwidth and treewidth in [12], linear-width in [13], and branchwidth in [14]. In a few words, a characteristic serves to filter the main data structure of a parameter to its essential part, a part that is able to be constructed from node to node of a tree decomposition. Moreover, as we will see, the information encoded by a characteristic depends on the width of the tree decomposition and, therefore, it is constant for partial w-trees.

A *characteristic pair* is any pair $(\lambda, \mathbf{A})$ where λ is a sequence over a set $\mathcal{O}$ and $\mathbf{A}$ is a sequence of typical sequences such that $|\mathbf{A}| = |\lambda| + 1$. Notice that for any graph G and any order l of $V(G)$ the pair $(l, \mathbf{Q}_{G,l})$ is a characteristic pair. The *width* of a characteristic pair $(\lambda, \mathbf{A})$, is defined to be $\max(\mathbf{A})$.

The following procedure defines the *compression* of a characteristic pair relative to a subset of $\mathcal{O}$.

Procedure $\mathsf{Com}(l, \mathbf{R}, S)$.

Input: A characteristic pair $(l, \mathbf{R})$ and a set S.

Output: A characteristic pair $(\lambda, \mathbf{A})$.

We assume the notations $l = [v_1, \ldots, v_{|l|}]$ and $\lambda = [v_{i_1}, v_{i_2}, \ldots, v_{i_\rho}]$.

1: $\lambda \leftarrow l[S]$.

2: $\mathbf{A} \leftarrow [\tau(\mathbf{R}[0, i_1 - 1]), \tau(\mathbf{R}[i_1, i_2 - 1]), \ldots, \tau(\mathbf{R}[i_{\rho-1}, i_\rho - 1]), \tau(\mathbf{R}[i_\rho, |l|])]$.

3: Output $(\lambda, \mathbf{A})$.

4: End.

The pair $([a, b, c, d, e], [[0, 3], [4], [3, 7, 2], [3], [8, 1, 3], [3, 8, 4, 6]])$ is an example of a characteristic pair. The compression of this characteristic pair to the set $S = \{a, c\}$ is the characteristic pair $([a, c], [[0, 3], [4, 7, 2], [3, 8, 1, 8, 4, 6]])$.

Finally, we will need the following lemma that is a direct consequence of Lemma 2.15.

LEMMA 2.18. *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* $\mathbf{C} \prec \mathbf{C}'$.

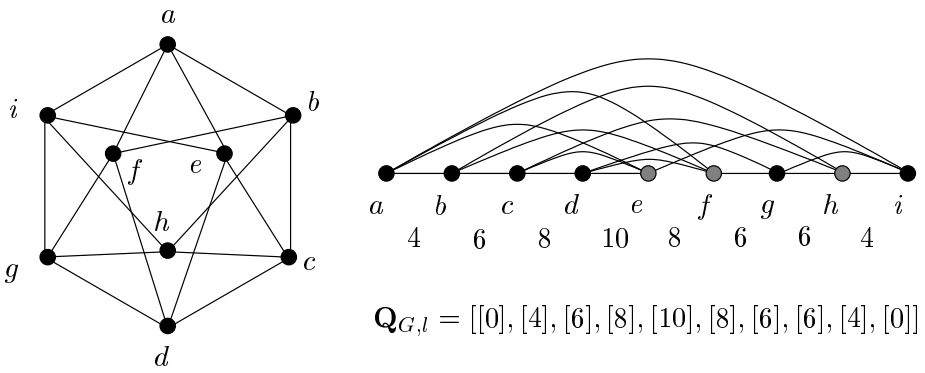

FIG. 3. A graph G and a vertex ordering of l.

Given a graph G with n vertices, a vertex ordering l of G and $S \subseteq V(G)$, the *S-characteristic* of l is $C_S(G,l) = \mathsf{Com}(l, \mathbf{Q}_{G,l}, S)$. Notice that, from the definition of the S-characteristic of a vertex ordering l of a graph G we have that the $V(G)$-characteristic of l is equal to $(l, \mathbf{Q}_{G,l})$, i.e. $C_{V(G)}(G,l) = (l, \mathbf{Q}_{G,l})$ (clearly, $\mathsf{Com}(l, \mathbf{Q}_{G,l}, V(G)) = (l, \mathbf{Q}_{G,l})$). We will simply use the term *characteristic* when there is no confusion on the choice of S and l.

Given the S-characteristics $(\lambda^i, \mathbf{A}^i), i = 1, 2$, of two different vertex orderings of G we say that $(\lambda^1, \mathbf{A}^1) \prec (\lambda^2, \mathbf{A}^2)$ when $\lambda^1 = \lambda^2$ and $\mathbf{A}^1 \prec \mathbf{A}^2$.

Given a graph G and a vertex subset S, we say that a characteristic pair $(\lambda, \mathbf{A})$ is an *S-characteristic* when $(\lambda, \mathbf{A}) = C_S(l, G)$ for some ordering l of the vertices of G. Notice that for any $S \subseteq V(G)$, l is a vertex ordering of G with width at most k iff the width of $C_S(l, G)$ is at most k. For an example of an S characteristic, we consider the graph G and its vertex layout l as shown in Figure 3. If $S = \{e, f, h\}$, then $C_S(l, G) = ([e, f, h], [[0, 8], [8], [6], [4, 0]])$ and if $S = \{a, b, c, d, g, i\}$, then $C_S(l, G) = ([a, b, c, d, g, i], [[0], [4], [6], [8], [10, 6], [6, 4], [0]])$.

The following result, [45, Lemma 3.2], bounds the number of characteristics of a vertex ordering.

LEMMA 2.19. *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 $w! \, (\frac{8}{3} \, 2^{2k})^{w+1}$.*

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 lemma can be derived directly from the definitions.

LEMMA 2.20. *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.*

Lemma 2.19 along with Corollary 2.1 give the following.

COROLLARY 2.2. *Let G be graph with n vertices of degree bounded by d and let (X, U) be a nice tree decomposition of G with width at most w. Then for any node $i \in V(U)$, $|FS(i)| \leq w!\ (\frac{8}{3})^{w+1}\ n^{2d(w+1)^2}$.*

3. 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.

An important consequence of Lemma 2.20 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 [45] 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 a full set of characteristics for a *start* node, can be found in the Appendix. In what follows, we will show how to compute a full set of characteristics for a node i when two full set of characteristics for its children j_1, j_2 are given.

We will now consider the case where X_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 $E(G_{j_1}) \cap E(G_{j_2}) = \emptyset$. Given a full set of characteristics $FS(j_1)$ for j_1 and a full set of characteristics F_{j_2} for j_2, the following algorithm computes a full set of characteristics $FS(i)$ for i.

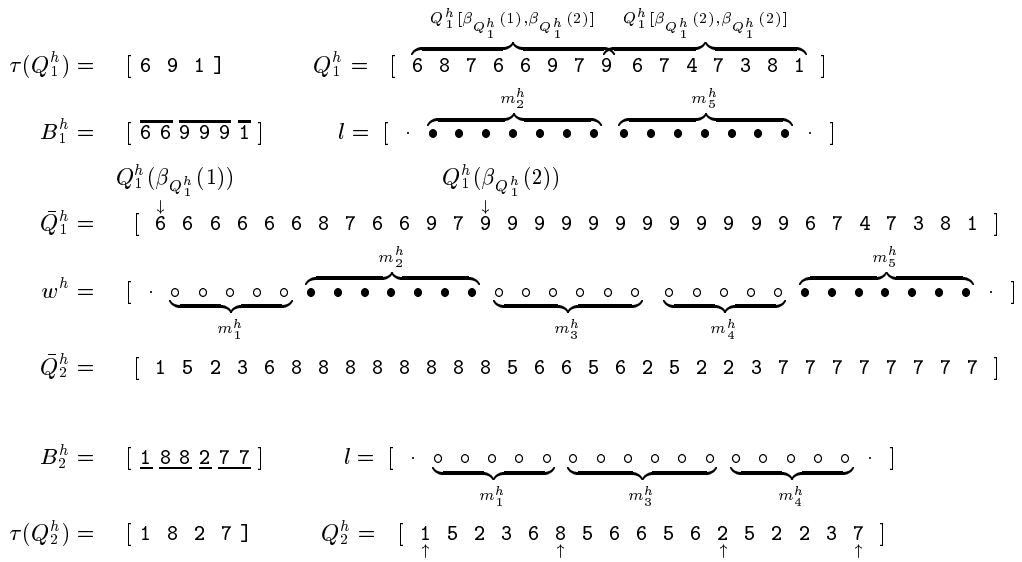

FIG. 4. An example of the proof of Lemma 3.1.

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_i}-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 $\max(\lambda, \mathbf{A}) \leq k$, set $FS(p) \leftarrow FS(p) \cup \{(\lambda, \mathbf{A})\}$.

5: Output $FS(p)$.

6: End.

We need the following lemma.

LEMMA 3.1. *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 following two procedures.

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 $(\lambda, \mathbf{A}_i) = \mathsf{Com}(G_i, l_i, S), i = 1, 2$.

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 $Q_i = \mathbf{Q}_{G_i, l_i}(0) \oplus \cdots \oplus \mathbf{Q}_{G_i, l_i}(r_i)$.

5: For any $h = 0, \ldots, \rho$,

set $l^h_1 = l_1[\kappa^1_h + 1, \kappa^1_{i+1} - 1]$ and $l^h_2 = l_2[\kappa^2_h + 1, \kappa^2_{i+1} - 1]$.

set $Q^h_1 = Q_1[\kappa^1_h, \kappa^1_{i+1} - 1]$ and $Q^h_2 = Q_2[\kappa^2_h, \kappa^2_{i+1} - 1]$.

set $w^h =$ Join-Orderings$(l^h_1, l^h_2, Q^h_1, Q^h_2, \mathbf{A}(h))$.

6: Set $l = w^0 \oplus [\lambda(1)] \oplus w^1 \oplus [\lambda(2)] \oplus w^2 \oplus \cdots \oplus [\lambda(\rho - 1)] \oplus w^{\rho-1} \oplus [\lambda(\rho)] \oplus w^\rho$.

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 $m_1 \oplus \cdots \oplus m_{w-1}$.

Figure 4 gives an example of the operation of 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 $\bar{Q}_i = \mathbf{Q}_{G^*_i, l}(0) \oplus \cdots \oplus \mathbf{Q}_{G^*_i, l}(r), i = 1, 2$, and $Q = \mathbf{Q}_{G, l}(0) \oplus \cdots \oplus \mathbf{Q}_{G_i, l}(r)$

(especially for the sequences $Q, \bar{Q}_1$, and $\bar{Q}_2$ we assume that their first elements are indexed by 0). We also set $Q^h = Q[\kappa_h, \kappa_{h+1} - 1]$ and $\bar{Q}_i^h = \bar{Q}_i[\kappa_h, \kappa_{h+1} - 1]$ 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 $\forall_{0 \leq h \leq \rho} \tau(Q^h) = \mathbf{A}(h)$.

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

$$\forall_{0 \leq h \leq \rho}\ \bar{Q}_1^h + \bar{Q}_2^h = Q^h. \tag{3}$$

We may assume that for any $h = 0, \ldots, \rho$, the computation of l_h is based on a pair B_1^h, B_2^h where

$$\begin{aligned} \mathbf{A}(h) &= \tau(B_1^h + B_2^h) \\ B_1^h &\sim B_2^h \\ B_i^h &\sqsupseteq \tau(Q_i^h), i = 1, 2. \end{aligned} \tag{4}$$

Notice that the result follows by Lemma 2.12, (3),(4), and (5) bellow.

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

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 $m_1^h, \ldots, m_{w-1}^h$ be the layouts produced during step **3** of Join-Orderings$(l_1^h, l_2^h, Q_1^h, Q_2^h, \mathbf{A}(h))$. Let $q_j^h = |m_1^h| + \ldots + |m_j^h|, 1 \leq j \leq w - 1$ (assume that $q_0^h = 0$ and $w = |B_1^h| = |B_2^h|$). The result follows from Lemma 2.12 taking into account that

$$\forall_{j, 1 \leq j \leq w-1}\ \bar{Q}_1^h[q_{j-1}^h + 1, q_j^h + 1] \sqsupseteq [B_1^h(j), B_1^h(j+1)] \tag{6}$$

Towards proving (6) we make first some observations. We will call a *gap* of a vertex layout 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 layout is crossed by the edges with endpoints on different sides. Notice that $\bar{Q}_1^h(1)$ 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 $j, 1 \leq j \leq w - 1$ and any $t, 1 \leq t \leq |m_j^h|$ $\bar{Q}_1^h(q_{j-1}^h + t + 1)$ 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 $j, 1 \leq j \leq w - 1$, $\bar{Q}_1^h(q_{j-1}^h + 1)$ corresponds to the number of edges that cross the "gap" on the left of vertex $m_j^h(1)$ and for any $t, 1 \leq t \leq |m_j^h|$, $\bar{Q}_1^h(q_{j-1}^h + t + 1)$ corresponds to the number of edges that cross the "gap" on the right of vertex $m_j^h(t)$. Let j be an integer $1 \leq j \leq w - 1$.

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,

$$\nu(j) = 1 \ \Rightarrow \bar{Q}_1^h[q_{j-1}+1, q_j^h+1] = Q_1^h[\beta_{Q_1^h}(\beta_{B_1^h}^{-1}(j)), \beta_{Q_1^h}(\beta_{B_1^h}^{-1}(j)+1)] \qquad (7)$$

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

$t_{\text{left}} = \max\{t \mid$ the "gap" corresponding to $\bar{Q}_1^h(t)$ is on the left of $m_j(1)$ and has the vertex on its left (if exists) in l_1 and the vertex on its right not in $l_1\}$.
$t_{\text{right}} = \min\{t \mid$ the "gap" corresponding to $\bar{Q}_1^h(t)$ is on the right of $m_j(|m_j|)$ and has the vertex on its left not in l_1 and the vertex on its right (if exists) in $l_1\}$.

Notice that $t_{\text{left}} \leq q_{j-1}+1$ and $q_j + 1 \leq t_{\text{right}}$. Observe that the vertices in w^h that are delimited by "gaps" corresponding to $\bar{Q}_1^h[t_{\text{left}}, t_{\text{right}}]$ 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 $\bar{Q}_1^h[t_{\text{left}}, t_{\text{right}}]$ 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.

$$\nu(j) = 2 \ \Rightarrow \forall_{t, q_{j-1}+1 \leq t \leq q_j^h+1} \ \bar{Q}_1^h(t) = Q_1^h(\beta_{Q_1^h}(\beta_{B_1^h}^{-1}(j))) \qquad (8)$$

From Lemma 2.6 we have that

$$\tau(Q_1^h[\beta_{Q_1^h}(\beta_{B_1^h}^{-1}(j)), \beta_{Q_1^h}(\beta_{B_1^h}^{-1}(j)+1)]) \ = \ [Q_1^h(\beta_{Q_1^h}(\beta_{B_1^h}^{-1}(j))), Q_1^h(\beta_{Q_1^h}(\beta_{B_1^h}^{-1}(j)+1))] \quad (9)$$

and applying Lemma 2.7, we have

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

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

If $\nu(j)=1$ then Lemma 2.7 implies $\beta^{-1}_{B^h_1}(j+1)=\beta^{-1}_{B^h_1}(j)+1$ and therefore (11) can be written

$$Q^h_1(\beta_{Q^h_1}(\beta^{-1}_{B^h_1}(j)+1)) \;=\; B^h_1(j+1) \tag{12}$$

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

$$\forall_{j,1\leq j\leq w-1,\nu(j)=1}\ \tau(\bar{Q}^h_1[q_{j-1}+1,q^h_j+1]) \;=\; [B^h_1(j),B^h_1(j+1)] \tag{13}$$

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

If $\nu(j)=2$ then Lemma 2.7 implies $\beta^{-1}_{B^h_1}(j+1)=\beta^{-1}_{B^h_1}(j)$ and therefore (10) and (11) give

$$Q^h_1(\beta_{Q^h_1}(\beta^{-1}_{B^h_1}(j)))=B^h_1(j)=B^h_1(j+1) \tag{14}$$

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

LEMMA 3.2. *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 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 3.1, 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 $\bar{Q}_i=\mathbf{Q}_{G^*_i,l}(0)\oplus\cdots\oplus\mathbf{Q}_{G^*_i,l}(r), i=1,2$, and $Q=\mathbf{Q}_{G,l}(0)\oplus\cdots\oplus\mathbf{Q}_{G_i,l}(r)$ (especially for the sequences $Q,\bar{Q}_1$, and $\bar{Q}_2$ we assume that their first elements are indexed by 0). As $E(G_1)\cap E(G_2)=\emptyset$ and $E(G_1)\cup E(G_2)=E(G)$ we get,

$$Q \;=\; \bar{Q}_1+\bar{Q}_2 \tag{15}$$

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

$$\forall_{i=1,2}\ Q_i \;\sqsubseteq\; \bar{Q}_i \tag{16}$$

We assume that if $l=[v_1,\ldots,v_r]$, then $\lambda=[v_{\kappa_1},\ldots,v_{\kappa_\rho}]$ and if $l_i=[u^i_1,\ldots,u^i_{r_i}]$ then that $\lambda=[u^i_{\mu^i_1},\ldots,u^i_{\mu^i_\rho}], i=1,2$. We set $Q^h=Q[\kappa_h,\kappa_{h+1}-1], Q^h_i=Q[\mu^i_h,\mu^i_{h+1}-1], i=$

$1,2$, and $\bar{Q}_i^h = \bar{Q}_i[\kappa_h, \kappa_{h+1} - 1], i = 1,2$ and $h = 0, \ldots, \rho$ (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

$$\forall_{h,0\leq h\leq \rho}\ Q^h \ = \ \bar{Q}_1^h + \bar{Q}_2^h \tag{17}$$

$$\forall_{h,0\leq h\leq \rho}\ \forall_{i=1,2}\ Q_i^h \ \sqsubseteq \ \bar{Q}_i^h \tag{18}$$

Notice also that

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

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

(18) and (19), implies that

$$\forall_{i=1,2}\ \forall_{h,0\leq h\leq \rho}\ \mathbf{A}_i(h) \ = \ \tau(\bar{Q}_i^h) \tag{21}$$

Our target is to prove that for $h = 0, \ldots, \rho$ 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 $h = 0, \ldots, \rho$, we apply Lemma 2.17 for Q^h, $\bar{Q}_1^h$ and $\bar{Q}_2^h$. ▌

LEMMA 3.3. *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* 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 enough 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 algorithm Join Node we can assume that there exist two pairs $(\lambda, \mathbf{A}_h), h = 1,2$ where

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

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

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

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

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

Using now (23)–(25), we can apply Lemma 3.1 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 3.2, there exist a typical sequence $\mathbf{A}$ such that

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

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

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

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

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

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.

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

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

(26), (31), and applying Lemma 2.18 one can see that there exists a characteristic pair $(\lambda, \mathbf{A}^*)$ such that

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

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

Notice now that, from (32), and Lemma 3.1, there exists a layout 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 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 resume the results on this section in the following.

THEOREM 3.1. *An algorithm can be constructed that, 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, checks whether there exists an vertex ordering of $V(G)$ of cutwidth at most k in $O(k^4(w!)^2 2^{2kw}(\frac{8}{3})^{2w}\ n^{4d(w+1)^2+1})$ steps.*

Proof. From Lemma 2.1 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 Corollary 2.2 bounds the number of repetitions of step 2 of algorithm Introduce-Node by $w!\ (\frac{8}{3})^{w+1}\ n^{2d(w+1)^2}$. Moreover, step 3 involves at most w repetitions and step 4 at most $\leq 2k-1$ repetitions (Lemma 2.4). The time cost of Procedure Ins is dominated by its 2nd step which requires $O(w^2k)$ steps: $O(w)$ for each of the edges inserted, $\leq w+1$ for the sequences of $\mathbf{A}'$ whose elements should be incremented, and $\leq 2k-1$ for the elements of each one of these sequences (Lemma 2.4). Therefore, computing $FS(i)$ requires $O(w^3k^2 \cdot w!\ (\frac{8}{3})^w\ n^{2d(w+1)^2})$ steps

If i is a *forget* node then then Corollary 2.2 bounds the number of repetitions of step 2 of algorithm Introduce-Node by $w!\ (\frac{8}{3})^{w+1}\ n^{2d(w+1)^2}$. As procedure Del needs $O(wk)$ steps, computing $FS(i)$ requires $O(wk \cdot w!\ (\frac{8}{3})^w\ n^{2d(w+1)^2})$ steps.

If i is a *join* node then Corollary 2.2 bounds the number of repetitions of step 2 of algorithm Join-Node by $(w!\ (\frac{8}{3})^{w+1}\ n^{2d(w+1)^2})^2$. From Lemma 2.11, the computation of $\mathbf{A}_1 \bigotimes \mathbf{A}_2$ costs $O(k^4 2^{4kw})$ steps and dominates the cost of steps 3 and 4 of the same algorithm. Therefore, computing $FS(i)$ requires $O(k^4 2^{4kw}(w!)^2\ (\frac{8}{3})^{2w}\ n^{4d(w+1)^2})$ 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. ▌

THEOREM 3.2. *An algorithm can be constructed that, given a graph G is a graph 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*
$O((w!)^2(\frac{8}{3})^{2w} n^{4d(2w^2+3w+1)+1}(wd\log n)^5)$ *steps.*

Proof. From Corollary 2.1 it is enough to describe an algorithm checking whether $\mathsf{cutwidth}(G) \leq k$ for any $k = 1, \ldots, \lfloor (w+1)d\log n \rfloor$. This can be done by the algorithm of Theorem 3.1 and the result follows. ▌

4. CONSTRUCTING THE LAYOUT

In this section we will show how the algorithms of Theorems 3.1 and 3.2 can be enhanced in a way that they will also construct the corresponding layout.

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 way 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 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 Algorithm Forget-Node on characteristic $(\lambda_j, \mathbf{A}_j)$ where j is the unique child of i.
- If i is an *join* node then $(\lambda_i, \mathbf{A}_i)$ is one of the characteristics constructed after the application of Algorithm Introduce-Node on characteristics $(\lambda_{j_1}, \mathbf{A}_{j_1})$ and $(\lambda_{j_1}, \mathbf{A}_{j_1})$ 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 layout of $V(G_r)$ with cutwidth $\leq k$. Towards this, we will compute, for each node $i \in V(U)$ a layout 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 [45]. In each of these cases the new layout l_i can be constructed in $O(wk)$ steps. Assume now that i is a *join* node with children i_1 and i_2. Let also $l_{j_h}, h = 1, 2$ be two layouts of $G_{i_h}, h = 1, 2$ respectively, such that $C_{X_{j_h}}(G_{j_h}, l_{j_h}) = (\lambda_{j_h}, \mathbf{A}_{j_h}), h = 1, 2$. We will show how to construct a layout l_i such that $C_{X_{j_h}}(G_{j_h}, l_{j_h}) = (\lambda_i, \mathbf{A}_i)$.

Notice that, from Algorithm Join-Node, $\mathcal{A}_i$ is a member of $\mathbf{A}_{i_1} \bigotimes \mathbf{A}_{i_2}$. Recall now that, from Lemma 3.1, 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}, procedure

Construct-Join-Ordering$(G_{j_1}, G_{j_2}, X_i, l_{j_1}, l_{j_2}, \mathbf{A})$ will call procedure Join-Orderings $|X_i|$ times. If now, for $h = 1, 2$, we additionally maintain a data structure associating each of the "gaps" of λ_{j_h} to the limits of its corresponding sequence of "gaps" in l_{j_h}, we can implement step 3 of 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 layout 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 constructive analogue of Theorem 3.1.

THEOREM 4.1. *An algorithm can be constructed that, 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, checks whether there exists an vertex ordering of $V(G)$ of cutwidth at most k and, if this is the case, outputs an ordering of $V(G)$ of cutwidth$\leq k$ in $O(k^4(w!)^2 2^{2kw}(\frac{8}{3})^{2w}\ n^{4d(w+1)^2+1})$ steps.*

As $k \leq (w+1)d\log n$, we can conclude that the polynomial algorithm of Theorem 3.2 can be adapted to output an optimal layout with an additional cost of $O(wdn\log n)$ steps. Therefore, 3.2 can be rewritten as follows.

THEOREM 4.2. *An algorithm can be constructed that, 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, outputs an ordering of $V(G)$ of minimum cutwidth in $O((w!)^2(\frac{8}{3})^{2w} n^{4d(2w^2+3w+1)+1}(wd\log n)^5)$ steps.*

According to the main result in [7], there exists an algorithm that, in $O(w^{O(w)}2^{O(w^3)}n)$ steps, constructs a minimum width tree decomposition of any partial w-tree (see also [38, 30, 34]). This algorithm can serve as a preprocessing step to the algorithm of 4.2 that with input a partial w-tree G with vertices of degree at most d, outputs a vertex layout of G of minimum cutwidth.

5. 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

$S \in V(G)$ such that $E(S)$ contains all the hyperedges with at least one endpoint in S. We can prove the following extension of 4.1.

THEOREM 5.1. *An algorithm can be constructed that, given an hypergraph 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, outputs an ordering of $V(G)$ of minimum cutwidth in $O(k^4(w!)^2 2^{2kw}(\frac{8}{3})^{2w}\ n^{4d(w+1)^2+1})$ steps.*

Proof. By extending Lemma 2.1 for hypergraphs we assume that (X, U, r) is a nice tree-decomposition of G. To prove the theorem, it is enough to observe that all the algorithms of the previous sections can be straightforwardly generalized to hypergraphs with the same time costs. In particular, algorithms Forget-Node and Join-Node are exactly the same as they involve only operations on sequences of integers. The only changes required, concern procedure Ins, described in the Appendix, and are the following two:

1. The set N should now represent the set $\{U_1, \ldots, U_\sigma\}$ where $\{U_t \cup \{u\} \mid 1 \leq t \leq \sigma\}$ are the hyperedges of G_i that contain u as endpoint. For any $U_t, 1 \leq t \leq \sigma$ 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 λ (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 $|U_t| \leq w+1$, $1 \leq t \leq \sigma$. 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, $\sigma \leq 2k$. These two facts imply that computing $j_t^{\mathbf{l}}$ and $j_t^{\mathbf{r}}$ for $1 \leq t \leq \sigma$ can be done in $O(kw^2)$ steps.

2. In step **2** of Ins, cases (i) and (ii) should now be:

(i) If $j_h^{\mathbf{r}} \leq j$ then set $\mathbf{A}' \leftarrow \mathbf{A}'[0, j_h^{\mathbf{l}} - 1] \oplus (\mathbf{A}'[j_h^{\mathbf{l}}, j] + 1) \oplus \mathbf{A}'[j+1, \rho+1]$.

(ii) If $j_h^{\mathbf{l}} \geq j+1$ then set $\mathbf{A}' \leftarrow \mathbf{A}'[0, j] \oplus (\mathbf{A}'[j+1, j_h^{\mathbf{r}}] + 1) \oplus \mathbf{A}'[j_h^{\mathbf{r}} + 1, \rho+1]$.

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

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 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 5.1. *For any graph G, treewidth$(G^D) \leq$ treewidth$(G) \cdot \Delta(G)$.*

Proof. Let (X, U) be a tree decomposition of G with width $\leq k$. 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 $i \in V(U)$, $|Y_i| \leq \Delta(G) \cdot |X_i|$. 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 enough 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 examine 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 to both X_i and 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 way to compute (Y, U) from (X, U) in $O(k|V(G)|\Delta(G))$ steps.

The notion of linear-width for graphs was introduced by Thomas [46].

The linear-width of a graph G with n vertices is defined as follows. Let $l = [e_1, \ldots, e_r]$ be a layout of $E(G)$. For $i = 1, \ldots, r-1$, 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 $\max_{1\leq i\leq n-1}\{|\zeta_{l,G}(i)|\}$. 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 linearwidth, we have the following.

LEMMA 5.2. *If G is a trunked graph then* linear-width(G) = cutwidth(G^D).

As a consequence of Theorems 5.1, 5.1 and Lemma 5.2, we have an algorithm for linear-width.

THEOREM 5.2. *An algorithm can be constructed that, 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, outputs an ordering of $E(G)$ of minimum linear-width in $O(((dw)!)^2(\frac{32}{3})^{2dw}n^{12(dw)^2+20dw+9}(dw\log n)^5)$ steps.*

Proof. Let G be a n-vertex graph of treewidth w and $\Delta(G)\leq d$. Let also (U,X) be a tree decomposition of G constructed by the $O(w^{O(w)}2^{O(w^3)}n)$ algorithm of Bodlaender in [7]. From Lemma 2.2, we know that the pathwidth of G is at most $(w+1)\log n$ and, as linear-width$(G)\leq$ pathwidth$(G)+1$ (see, e.g. [43] or [44]), we get that linear-width$(G)\leq (w+1)\log n+1$. From Lemma 5.2 we have that cutwidth(G) =linear-width$(G^D)\leq (w+1)\log n+1$. Notice that a vertex ordering of G^D with minimum cutwidth corresponds to an edge ordering of G^n of minimum-linear width. Therefore, it is enough to check whether cutwidth$(G^D)\leq k$ for $k=1,\ldots,\lfloor(w+1)\log n+1\rfloor$ 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 5.1, and get a tree decomposition (Y,U) of G^D with treewidth $\leq dw$. The result now follows from Theorem 5.1, taking in mind that $\Delta(G^D)=2$. ∎

For a proof of the following, see [5].

LEMMA 5.3. *If G^n is the graph occurring from G after replacing every edge in G with two edges in parallel, then* pathwidth(G) = linear-width(G^n).

It is easy to see that there exists a procedure that given an edge ordering of a graph G with width $\leq k$, transforms it to a path decomposition of width $\leq k$ in $O(k|V(G)|)$ steps (e.g. see [13]). This fact along with Theorem 5.2 and Lemma 5.3 yield the following result.

THEOREM 5.3. *An algorithm can be constructed that, 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, outputs a path decomposition of G with minimum width in $O(((dw)!)^2(\frac{32}{3})^{2dw}n^{12(dw)^2+20dw+9}(dw\log n)^5)$ 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 [12]. However, the exponent in the complexity of this algorithm is quite large for any practical purpose. The algorithm proposed in Theorem 5.3 is faster and can serve as a more realistic approach for partial w-trees with bounded degree.

6. OPEN PROBLEMS

We have shown that the cutwidth of graphs with bounded treewidth and maximum degree can be computed in polynomial time. The most insisting open problem is to prove the same when the "bounded maximum degree" requirement is removed. Even if this is the case for pathwidth [12], it seems that our technique cannot be easily modified to solve the general problem because it is strongly depending on Lemma 2.3. 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 w. No algorithm of this type exists for cutwidth when $w > 1$, while, for pathwidth, the best, so far, result is an approximation algorithm in [10] for outerplanar graphs that have treewidth ≤ 2 (see also [26]).

REFERENCES

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

2. D. Adolphson. Single machine job sequencing with precedence constraints. *SIAM J. Comput.*, 6:40–54, 1977.

3. D. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM J. Appl. Math.*, 25(3):403–423, 1973.

4. S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graphs arrangements. In *37th IEEE Symposium on foundations of computer science*, 1996.

5. D. Bienstock and P. Seymour. Monotonicity in graph searching. *J. Algorithms*, 12:239 – 245, 1991.

6. H. L. Bodlaender. Improved self-reduction algorithms for graphs with bounded treewidth. *Discrete Appl. Math.*, 54:101–115, 1994.

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

8. H. L. Bodlaender. Treewidth: algorithmic techniques and results. In *Mathematical foundations of computer science 1997 (Bratislava)*, pages 19–36. Springer, Berlin, 1997.

9. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoret. Comput. Sci.*, 209(1-2):1–45, 1998.

10. H. L. Bodlaender and F. V. Fomin. Approximation of pathwidth of outerpanar graphs. Technical Report UU-CS-2000-23, Dept. of Computer Science, Utrecht University, 2000.

11. H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, and minimum elimination tree height. *J. Algorithms*, 18:238–255, 1995.

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

13. H. L. Bodlaender and D. M. Thilikos. Computing small search numbers in linear time. Technical Report UU-CS-1998-05, Dept. of Computer Science, Utrecht University, 1998.

14. H. L. Bodlaender and D. M. Thilikos. Constructive linear time algorithms for branchwidth. Technical Report UU-CS-2000-38, Dept. of Computer Science, Utrecht University, 2000.

15. R. A. Botafogo. Cluster analysis for hypertext systems. In *16th Annual ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 116–125, 1993.

16. F. R. K. Chung. On the cutwidth and topological bandwidth of a tree. *SIAM J. Alg. Disc. Meth.*, 6:268–277, 1985.

17. F. R. K. Chung and P. D. Seymour. Graphs with small bandwidth and cutwidth. *Disc. Math.*, 75:113–119, 1989.

18. M. Chung, F. Makedon, I. H. Sudborough, and J. Turner. Polynomial time algorithms for the min cut problem on degree restricted trees. In *Symposium on foundations of computer science*, volume 23, pages 262–271, Chicago, Nov. 1982.

19. J. Díaz, M. D. Penrose, J. Petit, and M. J. Serna. Layout problems on lattice graphs. In T. Asano, H. Imai, D. Lee, S. Nakano, and T. Tokuyama, editors, *Computing and Combinatorics*, volume 1627 of *Lecture Notes in Computer Science*, pages 103–112. Springer-Verlag, Berlin, 1999.

20. J. Díaz, J. Petit, and M. Serns. A survay on graph layout problems. Technical Report LSI-00-61R, Departament de Llenguatges i Sistemes Informàtics, 2000.

21. G. Even, J. Naor, S. Rao, and B. Schieber. Divide-and-Conquer Approximation Algorithms via Spreading Metrics. In *36th Proc. on Foundations of Computer Science*, pages 62–71, 1995.

22. M. R. Fellows and M. A. Langston. On well-partial-order theory and its application to combinatorial problems of VLSI design. *SIAM J. Disc. Meth.*, 5:117–126, 1992.

23. M. R. Fellows and M. A. Langston. On search, decision and the efficiency of polynomial-time algorithms. *J. Comp. Syst. Sc.*, 49:769–779, 1994.

24. M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

25. F. Gavril. Some NP-complete problems on graphs. In *Proc. 11th. Conference on Information Sciences and Systems*, pages 91–95, John Hopkins Univ., Baltimore, 1977.

26. R. Govindan, M. A. Langston, and X. Yan. Approximating the pathwidth of outerplanar graphs. *Inform. Process. Lett.*, 68(1):17–23, 1998.

27. L. H. Harper. Optimal assignments of number to vertices. *SIAM Journal*, 12(1):131–135, 1964.

28. D. R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM J. Comput.*, 29(2):492–514 (electronic), 1999.

29. E. Korach and N. Solel. Tree-width, path-width and cutwidth. *Discrete Appl. Math.*, 43:97–101, 1993.

30. J. Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *J. Algorithms*, 20:20–44, 1996.

31. T. Lengauer. Upper and lower bounds on the complexity of the min-cut linear arrangement problem on trees. *SIAM J. Alg. Disc. Meth.*, 3:99–113, 1982.

32. F. S. Makedon, C. H. Papadimitriou, and I. H. Sudborough. Topological bandwidth. *SIAM J. Alg. Disc. Meth.*, 6:418–444, 1985.

33. F. S. Makedon and I. H. Sudborough. On minimizing width in linear layouts. *Discrete Appl. Math.*, 23:243–265, 1989.

34. J. Matoušek and R. Thomas. Algorithms finding tree-decompositions of graphs. *J. Algorithms*, 12:1–22, 1991.

35. B. Monien and I. H. Sudborough. Min cut is NP-complete for edge weighted trees. *Theor. Comp. Sc.*, 58:209–229, 1988.

36. P. Mutzel. A polyhedral approach to planar augmentation and related problems. In P. Spirakis, editor, *European Symposium on Algorithms*, volume 979 of *Lecture Notes in Computer Science*, pages 497–507. Springer-Verlag, 1995.

37. K. Nakano. Linear layouts of generalized hypercubes. In J. van Leewen, editor, *Graph-theoretic concepts in computer science*, volume 790 of *Lecture Notes in Computer Science*, pages 364–375. Springer-Verlag, 1994.

38. B. Reed. Finding approximate separators and computing tree-width quickly. In *Proceedings of the 24th Annual Symposium on Theory of Computing*, pages 221–228, New York, 1992. ACM Press.

39. N. Robertson and P. D. Seymour. Graph minors. I. Excluding a forest. *J. Comb. Theory Series B*, 35:39–61, 1983.

40. N. Robertson and P. D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory Series B*, 36:49–64, 1984.

41. N. Robertson and P. D. Seymour. Graph minors — a survey. In I. Anderson, editor, *Surveys in Combinatorics*, pages 153–171. Cambridge Univ. Press, 1985.

42. F. Sharhokhi, O. Sýkora, L. Székely, and I. Vrt'o. On bipartite drawings and the linear arrangement problem, to appear in. *SIAM J. Comput.*

43. A. Takahashi, S. Ueno, and Y. Kajitani. Mixed-searching and proper-path-width. Technical report, Tokyo Institute of Technology, 1991.

44. D. M. Thilikos. Algorithms and obstructions for linear-width and related search parameters. *Discrete Applied Mathematics*, 105:239–271, 2000.

45. D. M. Thilikos, M. J. Serna, and H. L. Bodlaender. A constructive linear time algorithm for small cutwidth. Technical Report UU-CS-2000-24, Dept. of Computer Science, Utrecht University, 2000.

46. R. Thomas. Tree-decompositions of graphs. Lecture notes, School of Mathematics. Georgia Institute of Technology, Atlanta, Georgia 30332, USA, 1996.

47. M. Yannakakis. A polynomial algorithm for the min-cut linear arrangement of trees. *J. ACM*, 32:950–988, 1985.

APPENDIX

A full set for a *start* node

If $X_i = \{x\}$ is a *start* node, then $FS(i) = \{([x], [[0], [0]])\}$.

A full set for an *introduce* node

If i is an *introduce* node and j is the (unique) child of i then the following algorithm computes $FS(i)$, given $FS(j)$.

Algorithm Introduce-Node

Input: A full set of characteristics $FS(j)$ for j.

Output: A full set of characteristics $FS(i)$ for i.

1: Initialize $FS(i) = \emptyset$ and set $\rho = |X_j|$, $\{u\} = X_i - X_j$, and $N = N_{G_i}(u)$.

2: For any X_j-characteristic $(\lambda, \mathbf{A}) \in FS(j)$ **do**

3: **for** $j = 0$ **to** ρ **do**

4: **for** $m = 1$ **to** $|\mathbf{A}(j)|$ **do**.

5: Let $(\lambda', \mathbf{A}') = \mathsf{Ins}(G_i, u, X_j, N, \lambda, \mathbf{A}, j, m)$

if $\max(\mathbf{A}') \leq k$, then set $FS(i) \leftarrow FS(i) \cup \{(\lambda', \mathbf{A}')\}$.

5: Output $FS(i)$.

6: end.

Procedure $\mathsf{Ins}(G, u, S, N, \lambda, \mathbf{A}, j, m)$.

Input: A graph G, a vertex $u \notin V(G)$, two sets S, N where $N \subseteq S \subseteq V(G)$,

a S-characteristic $(\lambda, \mathbf{A})$ of some vertex ordering l of G,

an integer $j, 0 \leq j \leq |\lambda|$, and an integer $m, 1 \leq m \leq |\mathbf{A}(j)|$.

Output: An $(S \cup \{u\})$-characteristic $(\lambda', \mathbf{A}')$ of some vertex ordering

$l' = l[1, \ldots, \gamma] \oplus [u] \oplus l[\gamma + 1, \ldots, |l|]$ of G' where $0 \leq \gamma \leq |l|$ and

$G' = (V(G) \cup \{u\}, E(G) \cup \{\{u, z\} \mid z \in N\})$.

Assume the notations: $\lambda = [u_1, \ldots, u_\rho]$, and $[u_{j_1}, \ldots, u_{j_\sigma}] = \lambda[N]$.

1: (Insertion of u)

Set $\lambda' = \lambda[1, j] \oplus [u] \oplus \lambda[j + 1, \rho]$

and $\mathbf{A}' = \mathbf{A}[0, j - 1] \oplus [\mathbf{A}(j)[1, m]] \oplus [\mathbf{A}(j)[m, |\mathbf{A}(j)|]] \oplus \mathbf{A}[j + 1, \rho]$.

2: (Insertion of the edges from u)

for $h = 1$ **to** σ **do**

(i) If $j_h \leq j$ then set $\mathbf{A}' \leftarrow \mathbf{A}'[0, j_h - 1] \oplus (\mathbf{A}'[j_h, j] + 1) \oplus \mathbf{A}'[j + 1, \rho + 1]$.

(ii) If $j_h \geq j + 1$ then set $\mathbf{A}' \leftarrow \mathbf{A}'[0, j] \oplus (\mathbf{A}'[j + 1, j_h] + 1) \oplus \mathbf{A}'[j_h + 1, \rho + 1]$.

3: Output $(\lambda', \mathbf{A}')$.

4: End.

LEMMA 6.1 ([45]). *If i is an introduce node with a child j and $FS(j)$ is a full set of characteristics for j then $FS(i)$ constructed by the* Introduce-Node *algorithm is a full set of characteristics for i.*

A full set for a *forget* node

If i is a *forget* node and j is the (unique) child of i then the following algorithm computes $FS(i)$, given $FS(j)$.

Algorithm Forget-Node

Input: A full set of characteristics $FS(j)$ for j.

Output: A full set of characteristics $FS(i)$ for i.

1: Initialize $FS(i) = \emptyset$ and let u be the forget vertex of G_i.

2: For any $(\lambda, \mathbf{A}) \in FS(j)$ **do**

3: $\quad FS(i) \leftarrow FS(i) \cup \{\mathsf{Del}(\lambda, \mathbf{A}, u)\}$.

4: Output $FS(i)$.

5: end.

Procedure $\mathsf{Del}(\lambda, \mathbf{A}, v)$.

Input: A characteristic $(\lambda, \mathbf{A})$ and a vertex $u \in V(\lambda)$.

Output: A characteristic pair $(\lambda', \mathbf{A}')$.

Assume the notation $\lambda = [u_1, \ldots, u_j, \ldots, u_\rho]$ where

$u = u_j$.

1: $\lambda' \leftarrow \lambda(1, j-1) \oplus \lambda(j+1, \rho)$.

2: $\mathbf{A}' \leftarrow \mathbf{A}[0, j-2] \oplus [\tau(\mathbf{A}(j-1) \oplus \mathbf{A}(j))] \oplus \mathbf{A}[j+1, \rho]$.

3: Output $(\lambda', \mathbf{A}')$.

4: End.

LEMMA 6.2 ([45]). *If i is an forget node with a child j and $FS(j)$ is a full set of characteristics for j then $FS(i)$ constructed by the* Forget-Node *algorithm is a full set of characteristics for i.*