A unified FPT Algorithm for Width of Partition Functions

Pascal Berthomé, Nicolas Nisse

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A unified FPT Algorithm for Width of Partition Functions

Pascal Berthomé — Nicolas Nisse

N° 6646

September 2008

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6646--FR+ENG

A unified FPT Algorithm for Width of Partition Functions

Pascal Berthomé*, Nicolas Nisse†

Thème COM — Systèmes communicants
Projets Mascotte

Rapport de recherche n° 6646 — September 2008 — 36 pages

Abstract: During the last decades, several polynomial-time algorithms have been designed that decide if a graph has treewidth (resp., pathwidth, branchwidth, etc.) at most k, where k is a fixed parameter. Amini *et al.* (to appear in SIAM J. Discrete Maths.) use the notions of partitioning-trees and partition functions as a generalized view of classical decompositions of graphs, namely tree-decomposition, path-decomposition, branch-decomposition, etc. In this paper, we propose a set of simple sufficient conditions on a partition function Φ, that ensures the existence of a linear-time explicit algorithm deciding if a set A has Φ-width at most k (k fixed). In particular, the algorithm we propose unifies the existing algorithms for treewidth, pathwidth, linearwidth, branchwidth, carvingwidth and cutwidth. It also provides the first Fixed Parameter Tractable linear-time algorithm deciding if the q-branched treewidth, defined by Fomin *et al.* (Algorithmica 2007), of a graph is at most k (k and q are fixed). Our decision algorithm can be turned into a constructive one by following the ideas of Bodlaender and Kloks (J. of Alg. 1996).

Key-words: Tree-decomposition, FPT-algorithm, width-parameters, characteristics.

This work was partially funded by the support of CONICYT via Anillo en Redes ACT08, and of the European projects 1ST FET AEOLUS and COST 293 GRAAL.

* LIFO, ENSI-Bourges, Université d'Orléans, Bourges, France. {pascal.berthome@ensi-bourges.fr@sophia.inria.fr
† MASCOTTE, INRIA, I3S, CNRS, UNS, Sophia Antipolis, France. {nicolas.nisse@sophia.inria.fr

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Algorithme FPT unifié pour le calcul des fonctions de partition

Résumé : Depuis une vingtaine d'annes, de nombreux algorithmes polynomiaux ont été conçu pour les problèmes consistant à décider si la largeur arborescente (resp., largeur linéaire, largeur en branche, etc.) d'un graphe est au plus k, où k est un paramètre fixé. Amini *et al.* (à paraître dans SIAM J. Discrete Maths.) utilisent les notions d'arbre de partition et de fonctions de partition pour généraliser les décompositions "classiques" des graphes, comme par exemple la décomposition arborescente, la décomposition linéaire, la décomposition en branches, etc. Dans ce papier, nous proposons des conditions simples et suffisantes qui, si elles sont satisfaites par une fonction de partition Φ, suffisent à assurer l'existence d'un algorithme linéaire qui décide si un ensemble A possède une Φ-largeur au plus k (k étant fixé). En particulier, l'algorithme que nous proposons unifie les algorithmes existants pour la largeur arborescente, la largeur linéaire, la largeur en branche, etc. Notre algorithme est également le premier algorithme FPT décidant en temps linéaire si la largeur arborescente q-branchée, définie par Fomin *et al.* (à paraître dans Algorithmica), d'un graphe est au plus k (k et q étant fixés). Notre algorithme de décision peut être modifié en un algorithme constructif en utilisant les idées de Bodlaender and Kloks (J. of Alg. 1996).

Mots-clés : Décomposition arborescente, algorithme FPT, largeurs de graphes, caractéristique.

1 Introduction

The notion of *treewidth* is central in the theory of the Graph Minors developed by Robertson and Seymour [RS86]. Roughly, the treewidth of a graph measures how close a graph is to a tree. More formally, a *tree-decomposition* $(T, \mathcal{X})$ of a graph $G = (V, E)$ is a tree T together with a family $\mathcal{X} = (X_t)_{t \in V(T)}$ of subsets of V, such that: (1) $\bigcup_{t \in V(T)} X_t = V$, (2) for any edge $e = \{u, v\} \in E$, there is $t \in V(T)$ such that $u, v \in X_t$, and (3) for any $v \in V$, the set of t such that $v \in X_t$ induces a subtree of T. The width of $(T, \mathcal{X})$ is the maximum size of X_t minus 1, $t \in V(T)$, and the treewidth $tw(G)$ of a graph G is the minimum width among its tree-decompositions. If T is restricted to be a path, we get a path-decomposition of G, and the *pathwidth* $pw(G)$ of G is the minimum width among its path-decompositions.

Both pathwidth and treewidth have a nice theoretical-game interpretation (see [Bie91, FT08] for surveys). Pathwidth can be described as a graph searching game where a team of searchers aims at capturing an invisible and arbitrary fast fugitive hidden on the vertices of the graph, whereas treewidth deals with the capture of a visible fugitive. In [FFN07], Fomin *et al.* introduce a variant of these games, called *non-deterministic graph searching*, that establishes a link between pathwidth and treewidth. Loosely speaking, in non-deterministic graph searching, the fugitive is invisible, but the searchers are allowed to query an oracle that possesses complete information about the position of the fugitive. However, the number of times the searchers can query the oracle is limited. The *q-limited search number* of a graph G, denoted by $s_q(G)$, is the smallest number of searchers required to capture an invisible fugitive in G, performing at most $q \geq 0$ queries to the oracle. Fomin *et al.* give the following interpretation of non-deterministic graph searching in terms of graph decomposition. A tree-decomposition $(T, \mathcal{X})$ is *q-branched* if T can be rooted in such a way that any path from the root to a leaf contains at most $q \geq 0$ vertices with at least two children. $q \geq 0$ being fixed, the q-branched treewidth $tw_q(G)$ of a graph G is the minimum width among its q-branched tree-decompositions ($tw_\infty(G) = tw(G)$ and $tw_0(G) = pw(G)$). For any $q \geq 0$ and any graph G, $s_q(G) = tw_q(G) + 1$ [FFN07, MN08]. Fomin *et al.* prove that deciding $s_q(G)$ is NP-complete for any $q \geq 0$, and design an algorithm that decides whether $s_q(G) \leq k$ in time $O(n^{k+1})$ for any n-node graph G [FFN07]. Prior to this work, no explicit *Fixed Parameter Tractable* (FPT) algorithm for this problem was known.

The notion of treewidth also plays an important role in the domain of algorithmic computational complexity. Indeed, many graph theoretical problems that are NP-complete in general are tractable when input graphs have bounded treewidth. Thus, an important challenge consists in computing optimal tree-decompositions of graphs. Much research has been done on the problem of finding an optimal tree-decomposition. This problem is NP-complete [ACP87] and special interest has been directed toward special graph classes [Bod93, BM93, BKK95]. The case of the class of graphs with bounded treewidth has been widely studied in the literature [ACP87, Ree92].

In their seminal work on Graph Minors [RS94, RS04], Robertson and Seymour give a non-constructive proof of the existence of a $O(n^2)$ decision algorithm for the problems of deciding whether a graph belongs to some minor-closed class of graphs. Given that, for any

k, the class of graphs of treewidth at most k is minor-closed, an immediate consequence is the existence of a polynomial-time algorithm deciding whether a graph has treewidth at most k, where k is a fixed parameter. In [BK96], Bodlaender and Kloks design a linear time algorithm for solving this problem. More precisely, k and k' being fixed, given a n-node graph G and a tree-decomposition of width at most k' of G, the Bodlaender and Kloks' algorithm decides if $tw(G) \leq k$ in time $O(n)$. The big-oh hides a constant more than exponential in k and k'. In the last decades, analogous algorithms have been designed for other width parameters of graphs like pathwidth [BK96], branchwidth [BT97], linearwidth [BT04], carvingwidth and cutwidth [TSB00]. These algorithms are mainly based on the notion of *characteristic* (see Section 4). This paper aims at unifying and generalizing these FPT algorithms. As a particular application, our algorithm decides in linear time if the q-limited search number of a graph G is at most k, $q \geq 0$ and $k \geq 1$ fixed.

In order to generalize the algorithm of [BK96], we use the notions of partition function and partitioning-tree defined in [AMNT07]. Given a finite set A, a partition function Φ is a function from the set of partitions of A into the integers. A partitioning-tree of A is a tree T together with a one-to-one mapping between A and the leaves of T. The Φ-width of T is the maximum $\Phi(\mathcal{P})$, for any partition $\mathcal{P}$ of A defined by the internal vertices of T, and the Φ-width of A is the minimum Φ-width of its partitioning-trees. Partition functions are a unified view for a large class of width parameters like treewidth, pathwidth, branchwidth, etc. In [AMNT07] is given a simple sufficient property that a partition function over A must satisfy to ensure that either A admits a partitioning-tree of width at most $k \geq 1$, or there exists a *k-bramble* (a dual structure).

In this paper, we extend the definition of Φ-width to the one of q-branched Φ-width of a set A. Then, we use the framework of [BK96] applied to the notions of partition functions and partitioning-tree in order to design a unified linear-time algorithm that decides if a finite set has q-branched Φ-width at most k. Again, $q \geq 0$ and $k \geq 1$ are fixed parameters.

Our results: We propose a simple set of sufficient properties and an algorithm such that, for any k and q fixed parameters, and any partition function Φ satisfying the properties, our algorithm decides in time $O(|A|)$ if a finite set A has q-branched Φ-width at most k (Theorem 1). Since treewidth, pathwidth, branchwidth, cutwidth, linearwidth, and carvingwidth can be defined in terms of Φ-width for some particular partition functions Φ that satisfy our properties (Theorem 2), our algorithm unifies the works in [BK96, BT97, TSB00, BT04]. Moreover, our algorithm generalizes the previous algorithms since it is not restricted to width-parameters of graphs but works as well for any partition function (not restricted to graphs) satisfying some simple properties. Finally, it provides the first explicit linear-time algorithm that decides if a graph G can be searched in a non-deterministic way by k searchers performing at most q queries, for any $k \geq 1, q \geq 0$ fixed. Due to lack of space, most of the proofs are omitted and can be found in the Appendices.

2 Main theorem.

2.1 Partition function and partitioning-tree.

Let A be a finite set. Let $\mathcal{P} = \{A_1, \cdots, A_r\}$ and $\mathcal{Q} = \{B_1, \cdots, B_p\}$ be two partitions of A. For any subset $A' \subseteq A$, the *restriction* $\mathcal{P} \cap A'$ of $\mathcal{P}$ to A' is the partition $\{A_1 \cap A', \cdots, A_r \cap A'\}$ of A' (where the empty parts have been removed). $\mathcal{Q}$ is a *subdivision* of $\mathcal{P}$ if, for any $j \leq p$, there exists $i \leq r$ with $B_j \subseteq A_i$.

A *partition function* Φ_A *over* A is a function from the set of partitions of A into the integers. Φ_A is *monotone* if, for any subdivision $\mathcal{Q}$ of a partition $\mathcal{P}$ of A, $\Phi_A(\mathcal{P}) \leq \Phi_A(\mathcal{Q})$. A (monotone) *partition function* Φ is a function that associates a (monotone) partition function Φ_A over A to any finite set A. A partition function Φ is *closed under taking subset* if, for any $A' \subseteq A$ and any partition $\mathcal{P}$ of A, $\Phi_{A'}(\mathcal{P} \cap A') \leq \Phi_A(\mathcal{P})$.

A *partitioning-tree* (T, σ) of A is a tree T together with a one-to-one mapping σ between A and the leaves of T. If T is rooted in $r \in V(T)$, the partitioning-tree is denoted by (T, r, σ). Any internal vertex $v \in V(T)$ corresponds to a partition $\mathcal{T}_v$ of A, defined by the sets of leaves of the connected components of $T \setminus v$. Similarly, any edge $e \in E(T)$ defines a bi-partition $\mathcal{T}_e$ of A. The Φ_A*-width* of (T, σ) is the maximum of $\Phi_A(\mathcal{T}_v)$ over the internal (i.e., non leaf) vertices v of T. The Φ-width of A is the minimum Φ_A-width of its partitioning-trees (cf. Figure 1 in Appendix A).

A *branching node* of a rooted tree (T, r) is either the root or a vertex of T with at least two children. A tree T is q-branched if there exists a root $r \in V(T)$ such that any path from r to a leaf contains at most $q \geq 0$ branching nodes. For instance, T is 0-branched if and only if T is a path. The *corpse* $cp(T)$ of a tree T denotes the rooted tree obtained from T by removing all its leaves. A partitioning tree (T, σ) is q-branched if the corpse $cp(T)$ of T is q-branched. For instance, a partitioning-tree (T, σ) is 0-branched if and only if T is a caterpillar. The q-branched Φ-width of A is the minimum Φ_A-width of its q-branched partitioning-trees.

2.2 Sufficient conditions for a linear time algorithm.

A *nice decomposition* $(D, \mathcal{X})$ of a finite set A is a $O(|A|)$-node rooted tree D, together with a familly $\mathcal{X} = (X_t)_{t \in V(D)}$ of subsets of A such that, $\cup_{t \in V(D)} X_t = A$, and for any $v \in V(D)$: (a) *start node*: v is a leaf, or (b) *introduce-node*: v has a unique child u, $X_u \subset X_v$ and $|X_v| = |X_u| + 1$, or (c) *forget-node*: v has a unique child u, $X_v \subset X_u$ and $|X_u| = |X_v| + 1$, or (d) *join-node*: v has exactly 2 children u and w, and $X_v = X_u = X_w$.

For any $v \in V(D)$, let D_v denote the subtree of D rooted in v, and $A_v = \cup_{t \in V(D_v)} X_t$.

Let Φ be a partition function. A *nice decomposition* $(D, \mathcal{X})$ for A is *compatible* with Φ if

- it exists a function F_Φ that associates an integer $F_\Phi(x, \mathcal{P}, e)$ to any integer x, partition $\mathcal{P}$ of some subset of A and element e of A, such that, F is strictly increasing in its first coordinate, and, for any introduce node $v \in V(D)$ with child u, any partition $\mathcal{P}$ of A_v,

$$\Phi_{A_v}(\mathcal{P}) = F_\Phi(\Phi_{A_u}(\mathcal{P} \cap A_u), \mathcal{P} \cap X_v, A_v \setminus A_u),$$

- it exists a function H_Φ that associates an integer $H_\Phi(x, y, \mathcal{P})$ to any pair of integers x, y, and partition $\mathcal{P}$ of some subset of A, such that, F is strictly increasing in its first and second coordinates, and, for any join node $v \in V(D)$ with children u and w, any partition $\mathcal{P}$ of A_v,

$$\Phi_{A_v}(\mathcal{P}) = H_\Phi(\Phi_{A_u}(\mathcal{P} \cap A_u), \Phi_{A_w}(\mathcal{P} \cap A_w), \mathcal{P} \cap X_v).$$

Intuitively, the existence of F_Φ and H_Φ means that it is possible to compute the Φ-width of some partitions $\mathcal{P}$ without knowing explicitly $\mathcal{P}$, but only knowing a restriction of $\mathcal{P}$ and the Φ-width of some restrictions of $\mathcal{P}$, these restriction being defined by the decomposition $(D, \mathcal{X})$.

Theorem 1. *Let Φ be a monotone partition function that is closed under taking subgraph. Let $k, k' \geq 1$ and $q \geq 0$ be three fixed integers (q may be ∞). There exists an algorithm that solves the following problem in time linear in the size of the input set:*
`input:` *a finite set A, and a nice decomposition $(D, \mathcal{X})$ for A that is compatible with Φ, and $\max_{t \in V(D)} |X_t| \leq k'$,* `output:` *decide if the q-branched Φ-width of A is at most k.*

Guideline of the algorithm.

In the following, we define the notion of *characteristic*, $Char((T, r, \sigma), X)$, of a partitioning-tree (T, r, σ) of A restricted to $X \subseteq A$ (Section 4). Roughly, characteristics are a compact data structure encoding the information necessary to build partitioning-trees. Following the framework in [BK96], we prove that the number of characteristics of q-branched partitioning-trees with Φ-width at most k, restricted to X, is bounded by a function of q, k and $|X|$ (this function does not depend on q when $q = \infty$). Then, we prove that the size of such characteristics is also bounded (Lemma 6). Finally, we define an ordering $\preceq$ on the characteristics that allows us to consider only some specific characteristics. More precisely, given $v \in V(D)$, a *set of characteristics* $Set(v)$ is a set of characteristics of q-branched partitioning-trees of A_v with Φ-width at most k, restricted to X_v. This set is said *full* if, roughly, all minimal such characteristics belong to it (see Section 4). By definition, there is a non empty full set of characteristics $FullSet(v)$ if and only if the q-branched Φ-width of A_v is at most k.

The algorithm proceeds by performing a dynamic programming. First, it computes a full set of characteristics $FullSet(v)$ for any start node (i.e., leaf) $v \in V(D)$. Then, for any $v \in V(D)$, a full set of characteristics $FullSet(v)$ is computed in constant time, starting from the full sets of characteristics of the children of v. This is the role of Procedures $IntroduceNode$ (Section 5.1.2), $ForgetNode$ (Section 5.2) and $JoinNode$ (Section 5.3). Therefore, in time $|V(D)| = O(|A|)$, our algorithm computes a full set of characteristics $FullSet(r_D)$ of the root of D. Since $A_{r_D} = A$, the q-branched Φ-width of A_v is at most k if and only if $FullSet(r_D) \neq \emptyset$.

In Sections 4 and 5, we present the main tools used in the design of our algorithm. First, in the next section, we present an important application of this theorem to the graphs.

3 Tractability of width-parameters of graphs.

This section is devoted to present an application of Theorem 1 in terms of graph's parameters. We first recall the definition of some graph's parameters, and establish their relationship with partition functions [AMNT07].

Let $G = (V, E)$ be a connected graph. Let Δ be the function that assigns, to any partition $\mathcal{X} = \{E_1, \cdots, E_r\}$ of E, the set of the vertices of G that are incident to edges in E_i and E_j, with $i \neq j$. Let δ be the partition function that assigns $|\Delta(\mathcal{X})|$ to any partition $\mathcal{X}$ of E.

Treewidth [RS86]: The treewidth of G is at most $k \geq 1$ if and only if there is a partitioning-tree of E with δ-width at most $k + 1$. Indeed, let (T, σ) be a partitioning-tree of E, then $(cp(T), (X_t)_{t \in V(cp(T))})$, with $X_t = \Delta(\mathcal{T}_t)$, is a tree-decomposition of G. Conversely, let $(T, \mathcal{X})$ be a tree-decomposition of G with width at most k. Then, for any edge $\{x, y\} \in E$, let us choose an arbitrary bag X_t that contains both x and y, add a leaf f adjacent to t in T, and let $\sigma(f) = \{x, y\}$. Finally, let S be the minimal subtree spanning all such leaves. The resulting tree (S, σ) is a partitioning-tree of E with δ-width at most $k + 1$ and $T = cp(S)$ [AMNT07].

Pathwidth [RS83]: The pathwidth of G is at most $k \geq 1$ if and only if there is a partitioning-tree (T, σ) of E with δ-width at most $k + 1$ and such that (T, σ) is 0-branched.

q-branched treewidth [FFN07]: More generally, the q-branched treewidth of G is at most $k \geq 1$ if and only if there is a q-branched partitioning-tree (T, σ) of E with δ-width at most $k + 1$. Recall that a partitioning-tree (T, σ) is q-branched if $cp(T)$ is q-branched.

Other partition functions defining branchwidth (br), linearwidth (lw), carvingwidth ($carw$), and cutwidth (cw) are described in Appendix F. The remaining part of this section is devoted to prove the Theorem 2 that is an important interpretation of Theorem 1 when width-parameters of graphs are concerned. We first need some lemmata. The following lemma is straightforward and its proof is thus omitted.

Lemma 1. *Aforementioned partition functions are monotone and closed under taking subset.*

Lemma 2. *Let G be a graph with maximum degree deg. Given a nice tree-decomposition $(T, \mathcal{Y})$ of G with width at most $k' \geq 1$, a nice decomposition $(D, \mathcal{X})$ of E, compatible with the partition functions corresponding to treewidth (resp., branchwidth), $q \geq 0$, and with $\max_{t \in V(D)} |X_t| \leq k' \cdot deg$ can be computed in linear time.*

Proof. Due to lack of space, we only prove the lemma for the partition function corresponding to treewidth. First, it is easy to obtain a nice decomposition $(D, \mathcal{X})$ of E from $(T, \mathcal{Y})$. For any $v \in V(T)$, let T_v denote the subtree of T rooted in v, and $A_v = \cup_{t \in V(T_v)} Y_t$, and let E_v be the set of edges belonging to the subgraph induced by the vertices contained in A_v that are incident to a vertex in Y_v. Any start node, resp., join node, Y_t of $(T, \mathcal{Y})$ corresponds to a start node, resp., join node, E_t of $(D, \mathcal{X})$. For any introduce node Y_t of $(T, \mathcal{Y})$, let $x \in V$ be the vertex such that $Y_t = Y_{t'} \cup \{x\}$, where t' is the single child of t in T. Let $e_1, \cdots, e_r$ be the edges that are incident to x and to some vertex in $Y_{t'}$. Then, Y_t is modified into a

path of introduce nodes $E(G[Y_{t'}]) \cup \{e_1\}, E(G[Y_{t'}]) \cup \{e_1, e_2\}, \cdots, E(G[Y_{t'}]) \cup \{e_1, e_2, \cdots, e_r\}$ in $(D, \mathcal{X})$. Finally, any forget node Y_t of $(T, \mathcal{Y})$ is modified into a path of forget nodes $E(G[Y_{t'}]) \setminus \{e_1\}, E(G[Y_{t'}]) \setminus \{e_1, e_2\}, \cdots, E(G[Y_{t'}]) \setminus \{e_1, e_2, \cdots, e_r\}$ in $(D, \mathcal{X})$, where t' is the unique child of t in T, and $e_1, \cdots, e_r$ are the edges that are incident to $x = Y_{t'} \setminus Y_t$ and to no other vertex in Y_t. The obtained decomposition of E is a nice decomposition and its width (i.e., the maximum number of edges in each bag) is at most the width of the tree-decomposition $(T, \mathcal{Y})$ times the maximum degree of G.

It remains to prove that $(D, \mathcal{X})$ is compatible with δ. Let F_δ be defined as follows.

Definition 1. *Let x be an integer, $\mathcal{P}$ be a partition of a subset E' of E and an edge $e \in E'$. Then, $F_\delta(x, \mathcal{P}, e) = x + |\{v \in e \mid v \in \Delta(\mathcal{P}) \setminus \Delta(\mathcal{P} \cap (E' \setminus \{e\}))\}|$.*

That is, F_δ adds to x the number of vertices incident to e that contribute to the border of the partition $\mathcal{P}$ because they are incident to e. F_δ is obviously strictly increasing in its first coordinate. Moreover, it can be computed in constant time when $|E'|$ is bounded by a constant.

For any $v \in V(D)$, let D_v denote the subtree of D rooted in v, and $A_v = \cup_{t \in V(D_v)} X_t$.

Let $v \in V(D)$ be an introduce node with child u, and let $\{e\} = X_v \setminus X_u$. Let $\mathcal{P}$ be a partition of A_v. We need to prove that $\delta_{A_v}(\mathcal{P}) = F_\delta(\delta_{A_u}(\mathcal{P} \cap A_u), \mathcal{P} \cap X_v, e)$. In other words, let us prove that $\delta_{A_v}(\mathcal{P}) = \delta_{A_u}(\mathcal{P} \cap A_u) + |\{v \in e \mid v \in \Delta(\mathcal{P} \cap X_v) \setminus \Delta(\mathcal{P} \cap X_v \cap (X_v \setminus \{e\}))\}|$.

$\delta_{A_v}(\mathcal{P})$ is the number of vertices in the subgraph induced by the set of edges A_v, that are incident to edges in different parts of $\mathcal{P}$. This set of vertices can be divided into two disjoint sets: (1) the set S_1 of vertices that are incident to two edges f and h that are different from e and that belong to different parts of $\mathcal{P}$, and (2) the set S_2 of vertices x incident to e and such that all other edges (different from e) incident to x belong to the same part of $\mathcal{P}$ that is not the part of e. S_1 is exactly the set of vertices belonging to $\Delta_{A_u}(\mathcal{P} \cap A_u)$, therefore $|S_1| = \delta_{A_u}(\mathcal{P} \cap A_u)$.

By definition of $(D, \mathcal{X})$ (because it has been built from a tree-decomposition), any edge of $A_u = A_v \setminus \{e\}$ that has a common end with e belongs to X_v. Therefore, any vertex in S_2 belongs to $\Delta(\mathcal{P} \cap X_v)$. It is easy to conclude that $|S_2| = |\{v \in e \mid v \in \Delta(\mathcal{P} \cap X_v) \setminus \Delta(\mathcal{P} \cap X_v \cap (X_v \setminus \{e\}))\}|$.

Therefore, the function F_δ satisfies the desired properties. Let H_δ be defined as follows.

Definition 2. *Let x and y be two integers, and let $\mathcal{P}$ be a partition of a subset E' of E. Then, $H_\delta(x, y, \mathcal{P}) = x + y - \delta(\mathcal{P})$.*

H_δ is obviously strictly increasing in its first and second coordinates. Moreover, it can be computed in constant time when $|E'|$ is bounded by a constant.

Let $v \in V(D)$ be a join node with children u and w, and let $\mathcal{P}$ be a partition of A_v, we must prove that $\delta_{A_v}(\mathcal{P}) = H_\delta(\delta_{A_u}(\mathcal{P} \cap A_u), \delta_{A_w}(\mathcal{P} \cap A_w), \mathcal{P} \cap X_v)$. That is, we prove that $\delta_{A_v}(\mathcal{P}) = \delta_{A_u}(\mathcal{P} \cap A_u) + \delta_{A_w}(\mathcal{P} \cap A_w) - \delta_{X_v}(\mathcal{P} \cap X_v)$.

First, note that $\Delta_{A_u}(\mathcal{P} \cap A_u) \cup \Delta_{A_w}(\mathcal{P} \cap A_w) \subseteq \Delta_{A_v}(\mathcal{P})$. Moreover, by definition of the nice decomposition $(D, \mathcal{X})$, an edge of $A_u \setminus X_v$ and an edge of $A_w \setminus X_v$ cannot be incident.

Indeed, X_v has been built by taking all edges incident to a vertex in a bag Y of the tree-decomposition $(T, \mathcal{Y})$. By the connectivity property of a tree-decomposition, if a vertex x would have been incident to an edge in $A_u \setminus A_w$ and to an edge in $A_w \setminus A_u$, then $x \in Y$ which would have implied that both these edges belong to $X_v = A_u \cap A_w$, a contradiction. Therefore, $\Delta_{A_v}(\mathcal{P}) \subseteq \Delta_{A_u}(\mathcal{P} \cap A_u) \cup \Delta_{A_w}(\mathcal{P} \cap A_w)$. To conclude, it is sufficient to observe that $\Delta_{A_u}(\mathcal{P} \cap A_u) \cap \Delta_{A_w}(\mathcal{P} \cap A_w) = \Delta_{X_v}(\mathcal{P} \cap X_v)$. □

Due to lack of space the proof of the following lemma is omitted and can be found in Appendix F.2.

Lemma 3. *Any nice tree-decomposition $(T, \mathcal{Y})$ of G is a nice decomposition of V that is compatible with the partition functions corresponding to carwingwidth (resp., cutwidth).*

Bodlaender designs a linear-time algorithm that decides if the treewidth of a graph G is at most k, and, if $tw(G) \leq k$ returns a tree-decomposition of width at most $O(k)$ [Bod96]. Moreover, a nice tree-decomposition of G can be computed in linear time from any tree-decomposition of G, and without increasing its width [BK96]. Finally, for any graph G and any $q \geq 0$, $tw(G) \leq tw_q(G) \leq pw(G)$ (By definition), $tw(G) \leq \frac{3}{2}bw(G)$ [RS91], $pw(G) \leq cw(G)$ [TSB00], $tw(G) \leq 3carw(G)$ [TSB00] and $pw(G) \leq lw(G)$ [BT04]. Therefore, as an application of Theorem 1, Lemmata 1,2 and 3 lead to:

Theorem 2. *Let k and q be two fixed parameters. There exists an algorithm that solves the following problem in time linear in the size of the input.*
`input:` *A graph G with degree bounded as a function of q and k,* `output:` *Decide if G has q-branched treewidth, resp., branchwidth, linearwidth, carvingwidth or cutwidth at most k.*

4 Characteristics of partitioning-trees.

This section is devoted to define the *characteristic* of any rooted partitioning-tree of some finite set A when we "restrict" it to a subset $B \subseteq A$. Let Φ be a monotone partition function.

4.1 Contraction of labeled path

One of the main tool that we use is the contraction of labeled paths. A *labeled path* is a path the vertices and edges of which are labeled by integers. In the following, any path of a partitioning tree of A will be considered as a labeled path, the vertices and edges being labeled by the Φ_A-width of the partition they correspond to. Note that, because Φ is monotone, the label of any edge is at most the minimum label of its ends.

Let $P = \{v_0, v_1, \cdots, v_n\}$ be a path where any vertex v_i is labeled with an integer $\ell(v_i)$, and any edge $e_i = \{v_{i-1}, v_i\}$ with an integer $\ell(e_i)$. To define the contraction of P, we revisit the notion of *typical sequence* of a sequence of integers [BK96] (see Appendix B.1). Roughly, the goal of the following operation is to contract some edges and vertices of P that are not "necessary" to remember the variations of the sequence $(\ell(v_0), \ell(e_1), \ell(v_1), \cdots, \ell(e_n), \ell(v_n))$.

The *contraction* $Contr(P)$ is the path obtained from P, with same ends, by contracting some edges and vertices obtained by the following procedure. Start with one integral variable $m = 1$. While $m \neq n$, do the following. Let i, $m \leq i \leq n-1$, be the greatest index such that, for any $m \leq j \leq i$, $\ell(e_m) \leq \ell(e_j)) \leq \ell(v_i)$ and $\ell(e_m) \leq \ell(v_j)) \leq \ell(v_i)$. Contract all vertices and edges between e and v_i. Then, set m to the greatest index such that, for any $i < j \leq m$, $\ell(v_i) \geq \ell(e_j) \geq \ell(e_m)$ and $\ell(v_i) \geq \ell(v_j) \geq \ell(e_m)$. Contract all vertices and edges between v_i and e_m. Edges and vertices of $Contr(P)$ keep their initial label (cf. Figure 2 in Appendix A).

The crucial property of $Contr(P) = \{v_0 = v'_0, v'_1, \cdots, v'_{p-1}, v'_p = v_n\}$ is that the sequence $S' = (\ell(e'_1), \ell(v'_1), \cdots, \ell(e'_{p-1}), \ell(v'_{p-1}), \ell(e'_p))$ (where $e'_i = \{v'_{i-1}, v'_i\}$) is "almost" the typical sequence of the sequence $S = (\ell(e_1), \ell(v_1), \cdots, \ell(e_n))$. More precisely, if $\ell(e'_1) \neq \ell(v'_1)$, then $S' = \tau(S)$, otherwise $(\ell(v'_1), \ell(e'_2), \cdots, \ell(v'_{p-1}), \ell(e'_p)) = \tau(S)$ ($\tau(S)$ denotes the typical sequence of S). Moreover, it is important to note that any $v \in V(Contr(P))$ ($e \in E(Contr(P))$) *represents* a unique $v^* \in V(P)$ ($e^* \in E(P)$).

We define $\max(P)$ as the maximum integer labeling an edge or a vertex of a labeled path P. Similarly, we define $\min(P)$. The following lemma is straightforward when using the fact that the number of different typical sequences of integers in $\{0, 1, \cdots, k\}$ is at most $\frac{8}{3}2^{2k}$ [BK96].

Lemma 4. *Let P be a labeled path.*
1. $\min(Contr(P)) = \min(P)$ *and* $\max(Contr(P)) = \max(P)$.
2. The number of contractions of paths P with $\max(P) \leq k$ *is bounded by a function of k.*

In the following, we need to order the labeled paths. An *extension* of a labeled path P is any path obtained by subdividing some edges of P an arbitrary number of times. Both edges and the vertex resulting from the subdivision of an edge e are labeled with $\ell(e)$. Given two labeled paths P and Q, we say that $P \preceq Q$ if there is an extension $P^* = \{p_1, \cdots, p_r\}$ of P and an extension $Q^* = \{q_1, \cdots, q_r\}$ of Q with same length, and such that $\ell(p_i) \leq \ell(q_i)$ and $\ell(\{p_i, p_{i+1}\}) \leq \ell(\{q_i, q_{i+1}\})$ for any $i \leq r$.

4.2 Restriction of a partitioning-tree

Let (T, r, σ) be a partitioning-tree of A. Any internal vertex $v \in V(T)$ is labeled by $\ell(v) = \Phi_A(\mathcal{T}_v)$ and any edge $e \in E(T)$ is labeled by $\ell(e) = \Phi_A(\mathcal{T}_e)$. To avoid technicality, we assume that T is not restricted to an edge, and r is not a leaf of T. Therefore, the corpse $cp(T)$ (T without its leaves) can be rooted in r. The *restriction* $Char((T, r, \sigma), B)$ of (T, r, σ) to B is a rooted partitioning-tree (T^*, r^*, σ^*) of B, together with a labeling function $\ell^* : V(cp(T^*)) \cup E(T^*) \to \mathbb{N}$, an integer $dist^*$, a subset K^* of vertices of T^* such that any $v \in K^*$ has extra label $(out^*(v), branch^*(v)) \in \mathbb{N} \times \{0, 1\}$. $Char((T, r, \sigma), B)$ is computed as follows.

1. Let T^* be the smallest subtree spanning the leaves of T that map elements of B. Let r^* be the vertex of T^* that is closest to r in T. From now on, T^* is rooted in r^*. For any leaf f of T^*, let $\sigma^*(f) = \sigma(f)$.

2. $dist^*$ is set to the number of branching nodes, in $cp(T)$, on the path between r and r^* (including r, and excluding r^*).
3. Let K^* be the set of vertices of T^* that are either a leaf of T^*, or the parent of a leaf of T^*, or a branching node of (T^*, r^*), or a branching node of $cp(T)$ in $V(T^*)$ (rooted in r).
4. For any vertex v of K^*, $branch^*(v) = 1$ if v is a branching node of $cp(T)$, and $branch^*(v) = 0$ otherwise. $out^*(v)$ is set to the maximum number of branching nodes on any path between v and a leaf in $A \setminus B$ all internal vertices of which are different from r^* and in $T \setminus T^*$.
5. Any internal vertex $v \in V(T^*)$ (resp., any edge $e \in E(T^*)$) keeps the same label than in T: $\ell^*(v) = \Phi_A(\mathcal{T}_v)$ (resp., $\ell^*(e) = \Phi_A(\mathcal{T}_e)$).
6. Then, for any two vertices v, w in K^* such that no internal vertices of the path P between v and w are in K^*, replace P by $Contr(P)$.

An example is illustrated in Figure 3 in Appendix A.

The key point for the understanding of the relationship between the partitioning-tree (T, r, σ) of A and its restriction $((T^*, r^*, \sigma^*), \ell^*, K^*, dist^*, out^*, branch^*)$ to B is based on the following. Any vertex of K^* represents a specific vertex of T that is either a leaf of T that maps an element of B, or the parent of such a leaf in T, or a branching node of $cp(T)$ or a vertex of T that defines a partition of B with at least three part. Any path P between two vertices v, w in K^* such that no internal vertices of P between v and w are in K^*, represents a path $P(v, w)$ in T the internal vertices of which have degree two in T. Moreover, by definition of the operation $P = Contr(P(v, w))$, any vertex (resp., edge) of P represents a specific vertex (resp., edge) of T. Beside, by Lemma 4, the maximum (minimum) label over the vertices and edges of T^* is the maximum (minimum) label over the vertices and edges of T. In particular, $\ell^*(v) \leq k$ and $\ell^*(e) \leq k$ for any $v \in V(T^*)$ and $e \in E(T^*)$ if and only if (T, r, σ) has Φ-width at most k.

Finally, the labels *out* and *branch* are sufficient to remember if (T, r, σ) is q-branched. Indeed, let the *br-height* of v, denoted by $brheight(v)$, in $cp(T)$ be the maximum number of branching nodes in a path from v to a leaf of the subtree of $cp(T)$ rooted in v. With this definition, (T, r, σ) is q-branched if and only if the br-height of r is at most q. If v is a leaf of T^*, it is a leaf of T, then $brheight(v) = 0$. Otherwise, the br-height of v can be computed recursively by $\max\{out(v), height\} + branch(v)$, where $height$ is the maximum of the br-height among the children of v. In particular, if (T, r, σ) is q branched, $out^*(v) \leq q$ for any $v \in K^*$. Finally, the br-height of r equals the br-height of r^* plus $dist^*$.

4.3 Characteristic of A restricted to B

Let $((T^*, r^*, \sigma^*), \ell^*, K^*, dist^*, out^*, branch^*)$ be such that (T^*, r^*, σ^*) is a rooted partitioning-tree of $B \subseteq A$, $\ell^* : V(cp(T^*)) \cup E(T^*) \to \mathbb{N}$, $K^* \subseteq V(T^*)$ that contains at least all leaves, parents of leaves, the root and vertices with degree at least three of T^*, $dist^* \in \mathbb{N}$, $out^* : K^* \to \mathbb{N}$, $branch^* : K^* \to \{0, 1\}$, and for any $v, w \in K^*$ such that no internal vertices of the path P between v and w are in K^*, $P = Contr(Q)$ (i.e., P results from some contraction).

Definition 3. *$((T^*, r^*, \sigma^*), \ell^*, K^*, dist^*, out^*, branch^*)$ is a characteristic of A restricted to B if it exists a partitioning-tree (T, r, σ) of A, such that $((T^*, r^*, \sigma^*), \ell^*, K^*, dist^*, out^*, branch^*) = Charac((T, r, \sigma), B)$. $((T^*, r^*, \sigma^*), \ell^*, K^*, dist^*, out^*, branch^*)$ is a (k, q)-characteristic of A restricted to B if, moreover, $\ell^* : V(T^*) \cup E(T^*) \to [0, k]$, and $dist^* + brheight(r^*) \leq q$. Note that the latter assumption implies that $out^* : K^* \to [0, q]$.*

Lemma 5. *$((T^*, r^*, \sigma^*), \ell^*, K^*, dist^*, out^*, branch^*)$ is a (k, q)-characteristic of A restricted to B if and only it exists a q-branched partitioning-tree (T, r, σ) of A with Φ-width at most k, such that $((T^*, r^*, \sigma^*), \ell^*, K^*, dist^*, out^*, branch^*) = Charac((T, r, \sigma), B)$.*

Lemma 6. *The number of (k, q)-characteristic of A restricted to B is bounded by $f(k, q, |B|)$.*

Proof. Let $((T^*, r^*, \sigma^*), \ell^*, K^*, dist^*, out^*, branch^*)$ be a (k, q)-characteristic of A restricted to B. T^* is a tree with $|B|$ leaves, $|K^*| \leq kq + 2k$, any path between two vertices in K^* has length at most $2k + 1$ (Lemma 4 and [BK96]), $dist^* \leq q$, and for any vertex $v \in V(cp(T^*))$ and edge $e \in E(T^*)$, $\ell^*(v) \leq k$, $\ell^*(e) \leq k$, and $out(v) \leq q$. □

Case $q = \infty$. Note that, if q is unbounded, the number of characteristics of (k, ∞)-characteristic of B is bounded by a function k and $|B|$. Indeed, if q is unbounded, we don't need to take the variables $dist^*$, out^* and $branch^*$ into account. More precisely, the items 2 and 4 of the previous procedure can be removed and K^* must be the set of vertices of T^* that are either a leaf of T^*, or a branching node of (T^*, r^*).

The *skeleton* $Sk(C)$ of $C = ((T^*, r^*, \sigma^*), \ell^*, K^*, dist^*, out^*, branch^*)$ is the tree obtained from T^* by contracting all vertices that are not in K^* (these vertices have degree two, thus the notion of contraction is well defined). Therefore, $V(Sk(C)) = K^*$. Two partitioning-trees (T, r, σ) and (T', r', σ') are *isomorphic* if there is an one-to-one function $\varphi : V(T) \to V(T')$ preserving the edges, such that $\varphi(r) = r'$, and moreover, $\sigma'(\varphi(f)) = \sigma(f)$ for any leaf f of T.

Definition 4. *Given two characteristics $C^* = ((T^*, r^*, \sigma^*), \ell^*, K^*, dist^*, out^*, branch^*)$ and $C = ((T, r, \sigma), \ell, K, dist, out, branch)$ of A restricted to B, $C^* \preceq C$ if $Sk(C^*)$ and $Sk(C)$ are isomorphic, $dist^* \leq dist$, for any $v, w \in K^* = K$, $out^*(v) \leq out(v)$, $branch^*(v) \leq branch(v)$, $\ell^*(v) \preceq \ell(v)$, and $P^*(v, w) \preceq P(v, w)$ where $P^*(v, w)$ is the path between v and w in T^*, and $P(v, w)$ is the path between v and w in T.*

Definition 5. *A set F of (k, q)-characteristics of A restricted to B is full if for any q-branched partitioning-tree (T, r, σ) of A with Φ-width at most k, there is a q-branched partitioning-tree (S, u, μ) of A, such that $Char((S, u, \mu), B) \preceq Charac((T, r, \sigma), B)$ and $Char((S, u, \mu), B) \in F$.*

5 Decision algorithm.

This section is devoted to the presentation of Procedures used in our main algorithm. Notations are those defined in Section 2 for Theorem 1. Let $(D, \mathcal{X})$ be a nice decomposition

for A that is compatible with Φ a monotone partition function, and $\max_{t\in V(D)} |X_t| \leq k'$. This section presents procedures that compute a full set $FSC(t)$ of (k,q)-characteristics of A_t restricted to X_t, for any $t \in V(T)$. recall that, our algorithm proceeds by dynamic programming from the leaves of D to its root. Due to lack of space, the proofs of Lemmata 7,8, and 9 are omitted and can be found in Appendix C,D and E.

If v is a leaf, i.e., a start node of D, $A_v = X_v$, and $|X_v| \leq k'$. $FSC(v)$ consists of all (k,q)-characteristics of X_v. By Lemma 6, $|FSC(v)|$ is bounded by a function of k', k and q. It can be computed in constant time.

5.1 Case of an introduce node.

We first explain how any labeled partitioning-tree (T, r, σ) of A can be turned into a partitioning-tree of $A \cup \{a\}$, $a \notin A$. The following procedure (cf. Figure 4 in Appendix A) will be useful for a better understanding of Procedure *IntroduceNode*, and will be used in the proof of its correctness.

5.1.1 Insertion of a new element in a partitioning-tree.

1. Let us choose either an internal vertex $v \in V(T)$ (Case 1) or an edge $e \in E(T)$ (Case 2) . In Case 1, we add a new leaf f adjacent to v, f mapping a and set $\ell(\{f,v\}) = \Phi(A,\{a\})$. In Case 2, let us subdivide e into two edges, both new edges and the new vertex w receive label $\ell(e)$. Then, add a new leaf f adjacent to w, f mapping a and set $\ell(\{f,w\}) = \Phi(A,\{a\})$. Let (T^*, r^*, σ^*) be the partitioning-tree of $A \cup \{a\}$ obtained in this way.

2. For any internal vertex $v \in V(T^*)$, let $\mathcal{T}_v$ be the partition of $A \cup \{a\}$ defined by v. At this step, note that $\ell(v) = \Phi(\mathcal{T}_v \cap A)$. We modify this label to $\ell(v) = \Phi(\mathcal{T}_v)$.

   Similarly, for any edge $e \in E(T^*)$ but the edge incident to f, $\ell(e) = \Phi(\mathcal{T}_e \cap A)$. We modify this label to $\ell(v) = \Phi(\mathcal{T}_e)$.

3. In Case 1, $cp(T) = cp(T^*)$, therefore, the branching nodes and the br-height of the vertices of $cp(T)$ do not change. In Case 2, let $e = \{x,y\}$ be the chosen edge, and x the parent of y. Obviously, all branching nodes of $cp(T)$ are branching nodes of $cp(T^*)$. Moreover, x is the single vertex of $cp(T^*)$ that may be a branching node of $cp(T^*)$ while it was not a branching node of $cp(T)$. This occurs iff y is a leaf and x has exactly one non-leaf child in T.

5.1.2 Procedure *IntroduceNode*.

Let v be an introduce node of D, u its child, and $\{a\} = X_v \setminus X_u$. Let $FSC(u)$ a full set of (k,q)-characteristics of A_u restricted to X_u. For any characteristic $C = ((T,r,\sigma), \ell, K, dist, out, branch) \in FSC(u)$, Procedure *IntroduceNode* proceeds as follows, repeating the five steps below, for any possible execution of Step 1. Roughly, it tries all possible ways to insert a in C.

1. **Update of T:**

   There are two ways of inserting a in C. Either choose an internal vertex v_{att} of $V(T)$, add a leaf v_{leaf} adjacent to v_{att} (**Case 1**), or choose an edge $f = \{v_{top}, v_{bottom}\}$ (with v_{top} closer to the root than v_{bottom}), subdivide it into $e_{top} = \{v_{top}, v_{att}\}$ and $e_{bottom} = \{v_{att}, v_{bottom}\}$ and add a new leaf v_{leaf} adjacent to v_{att} (**Case 2**). In both cases, set $\sigma(v_{leaf}) = a$ and let $K \leftarrow K \cup \{v_{att}, v_{leaf}\}$. Note that, now, T is a partitioning-tree of X_v.

2. **Labels of the new vertex (vertices) and edge(s):**

   First, $e_{new} = \{v_{leaf}, v_{att}\}$ receives label $\ell(e_{new}) = \Phi_{A_v}(\{A_u, \{a\}\})$.

   In Case 2, $\ell(v_{att}) = \ell(e_{top}) = \ell(e_{bottom}) = \ell(f)$, and $out(v_{att}) = branch(v_{att}) = 0$.

3. **Update of labels $\ell(e)$ and $\ell(v)$:**

   $\forall e \in E(T)$, $e \neq e_{new}$, let $\mathcal{T}_e$ be the partition of X_v defined by e. $\ell(e) \leftarrow F_\Phi(\ell(e), \mathcal{T}_e, a)$.

   $\forall t \in V(T)$, let $\mathcal{T}_t$ be the partition of X_v defined by t. $\ell(t) \leftarrow F_\Phi(\ell(t), \mathcal{T}_t, a)$.

   $\forall x, y \in K$ with no internal vertices of the path P between x and y are in K, $P \leftarrow Contr(P)$.

4. **Creation of a new branching node:**

   The variable *dist* and the variables $out(v)$ and $branch(v)$ ($v \in K$) are not modified.

   If initially v_{bottom} is a leaf of T, $branch(v_{top}) = 0$ and v_{top} has a child $x \neq v_{bottom}$ in T such that x is not a leaf of T. Then, $branch(v_{top}) = 1$. (this condition must be understood in contrast with the last item of the procedure of insertion of an element in a partitioning-tree).

5. **Update of FSC(v):** Let *height* be the br-height of r (computable thanks to the variables *out* and *branch*). If $dist + height \leq q$ and $\ell(v) \leq k$ for any internal vertex $v \in V(T)$, and $\ell(e) \leq k$ for any edge $e \in E(T)$, then $FSC(v) \leftarrow FSC(v) \cup \{C\}$.

Lemma 7. *IntroduceNode computes a full set of (k, q)-characteristics of A_v restricted to X_v.*

5.2 Procedure *ForgetNode*

Let v be a forget node of D, u its child and $FSC(u)$ a full set of (k, q)-characteristics of A_u restricted to X_u. For any characteristic $C = ((T, r, \sigma), \ell, K, dist, out, branch) \in FSC(u)$, Procedure *ForgetNode* proceeds as follows. Roughly, it restricts C to $X_v = X_u \setminus \{a\}$.

1. Let v_{leaf} be the leaf of T that maps $\{a\}$, let v_{att} be the vertex of T with degree at least three that is closest to v_{leaf} (if no such a vertex exists, T is a path and v_{att} is set to the other leaf). Let w be the neighbour of v_{att} in the path between v_{leaf} and v_{att}, and let $height(w)$ be the br-height of w. Let p be the number of vertex $v \in V(T)$ with $branch(v) = 1$ in the path between v_{att} and r (excluding v_{att}).

2. Remove the path between v_{leaf} and v_{att} (but the vertex v_{att}) from T and K.

   If $r \neq v_{att}$ and r is belongs to the path P between v_{leaf} and v_{att}, $r \leftarrow v_{att}$ and $dist \leftarrow dist + p$. Otherwise, i.e., $r = v_{att}$ or r does not belong to P, $out(v_{att}) \leftarrow \max\{out(v_{att}), height(w)\}$.

3. If v_{att} has degree 2 (after the removal), and v_{att} is not the parent of a leaf neither the root in the current tree, and $branch(v_{att}) = 0$, then $K \leftarrow K \setminus \{v_{att}\}$. Let $w_1, w_2 \in K$ be such that no internal vertices of the path P between w_1 and w_2 are in K and $v_{att} \in V(P)$. Then, $P \leftarrow Contr(P)$.

4. Add C to $FSC(v)$.

Lemma 8. *ForgetNode computes a full set of (k,q)-characteristics of A_v restricted to X_v.*

5.3 Case of an join node.

We first explain how a labeled partitioning-tree (T, r, σ) of $A \cup B$ and a labeled partitioning-tree (S, r_S, σ_S) of $A \cup C$ can be turned into a partitioning-tree of $A \cup B \cup C$, A, B and C being three pairwise disjoint sets. The procedure merging two partitioning-trees (cf. Figure 5 in Appendix A) will be useful for a better understanding of Procedure $JoinNode$, and will be used in the proof of its correctness.

5.3.1 Merging of two labeled paths.

First, we present an operation that merges two labeled paths $P = \{v_1, \cdots, v_n\}$ and $Q = \{w_1, w_2, \cdots, w_m\}$ with common ends, i.e., $v_1 = w_1$ and $w_m = v_n$, and vertex-disjoint otherwise. For any $i < n$, $e_i = \{v_i, v_{i+1}\}$, and for any $i < m$, $f_i = \{w_i, w_{i+1}\}$. Let $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Let $P^* = \{p_1, \cdots, p_h\}$ be an extension of P and $Q^* = \{q_1, \cdots, q_h\}$ be an extension of Q with same length. A *merging of P and Q using F* is the labeled path $\{r_1, \cdots, r_h\}$, where r_i is labeled by $F(\ell(p_i), \ell(q_i))$ and $\{r_i, r_{i+1}\}$ is labeled by $F(\ell(\{p_i, p_{i+1}\}), \ell(\{q_i, q_{i+1}\}))$ (cf. Figure 2 in Appendix A).

5.3.2 Merging of two partitioning-trees.

The key point in the operation of merging is that T and S must have some structure in common. More precisely, let (T', r', σ') (resp., (S', r'_S, σ'_S)) be the partitioning-tree of A obtained by taking the smallest subtree spanning the leaves of T (resp., S) that map the elements of A and contracting all vertices of which with degree two and different from the root (T', resp., S', is rooted in its vertex that is closest to r, resp., r_S). We impose that (T', r', σ') and (S', r'_S, σ'_S) are isomorphic and that $r'_S = r_S$.

Any (T^*, r^*, σ^*) built by the following procedure is a merging of (T, r, σ) and (S, r_S, σ_S) in A. (T^*, r^*, σ^*) is built as follows. Start with a copy of (T, r, σ) and a copy of (S, r_S, σ_S). For any vertex $v \in V(T') = V(S')$, let v_1 be the corresponding vertex in T, and v_2 be the corresponding vertex in S. Identify v_1 and v_2, that is, replace both these vertices by a new one $v^* \in V(T^*)$ adjacent to the neighbours of v_1 and v_2. For any $\{v, w\} \in E(T')$, let v^* and w^* be the vertices built as above. Currently in T^*, there are two paths P_T (initially a path of T) and P_S (initially a path of S) between v^* and w^* and that are vertex-disjoint but in

v^* and w^*. Compute a merging of P_T and P_S. Any vertex, resp., edge , of the resulting path defines a partition $\mathcal{P}$ of $A \cup B \cup C$, label it with $\Phi_{A\cup B\cup C}(\mathcal{P})$. Let r^* be the vertex that results from r'. Note that, a vertex $v \in V(T^*)$ is a branching node in $cp(T^*)$ if and only if v has been obtained from a branching node of $cp(T)$ or $cp(S)$, or v has been obtained from a vertex of T with one non-leaf child and from a vertex of S with one non-leaf child.

5.3.3 Procedure *JoinNode*.

Let v be a join node of D, let u, w be its children, let $FSC(u)$ be a full set of characteristics of A_u restricted to X_u, and $FSC(w)$ a full set of characteristics of A_w restricted to X_w.

The *structure* $Struct(C)$ of a tree C is the tree obtained from $Sk(C)$ by contracting all its vertices with degree two, different from r^*. For any characteristic $C = ((T, r, \sigma), \ell, K, dist, out, branch) \in FSC(u)$ and $C_S = ((S, r_S, \sigma_S), \ell_S, K_S, dist_S, out_S, branch_S) \in FSC(w)$, with isomorphic structures and $dist_S = 0$, Procedure *JoinNode* proceeds as follows. Roughly, it merges C and C_S to obtain $C^* = ((T^*, r^*, \sigma^*), \ell^*, K^*, dist^*, out^*, branch^*)$.

1 **Identifying the structures:** For any vertex $t \in V(Struct(T)) = V(Struct(S))$, let t_1 be the corresponding vertex in T, and t_2 be the corresponding vertex in S. Identify t_1 and t_2, let t^* be the resulting vertex. Note that r and r_S are identified, let r^* be the resulting vertex. Let $dist^* = dist$.

2 **Merging the paths:** For any $\{x, y\} \in E(Struct(T))$, let x^* and y^* be the vertices built as above. Currently in T^*, there are two paths P_T (initially a path of T) and P_S (initially a path of S) between x^* and y^* and that are vertex-disjoint but in x^* and y^*. Any internal vertex, resp., edge , of both these paths defines the same partition $\mathcal{P}$ of A. Compute a merging of P_T and P_S using the function $F : (i, j) \to H_\Phi(i, j, \mathcal{P})$.

3 **Updating of K^*:** Any vertex x^* in T^* results from $x_1 \in V(T)$ and $x_2 \in V(S)$. Let K^* be the set of vertices that result from $x_1 \in V(T)$ and $x_2 \in V(S)$ with, either $x_1 \in K$ or $x_2 \in K_S$. For any $x^* \in K^*$, $branch^*(x^*) = branch(x_1)$ *or* $branch_S(x_2)$ *or* $(out(x_1) > 0$ & $out_S(x_2) > 0)$ and $out^*(x^*) = \max\{out(x_1), out_S(x_2)\}$.

4 **Contracting the paths:** $\forall x, y \in K^*$ with no internal vertices of the path P between x and y are in K^*, $P \leftarrow Contr(P)$.

5 **Update of FSC(v):** Let *height* be the br-height of r^* (computable thanks to the variables out^* and $branch^*$). If $dist^* + height \leq q$ and $\ell^*(v) \leq k$ for any internal vertex $v \in V(T)$, and $\ell^*(e) \leq k$ for any edge $e \in E(T)$, then $FSC(v) \leftarrow FSC(v) \cup \{C^*\}$.

Lemma 9. *$FSC(v)$ is a full set of (k, q)-characteristics of $A_v = A_u \cup A_w$ restricted to X_v.*

References

[ACP87] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, 1987.

[AMNT07] Omid Amini, Frédéric Mazoit, Nicolas Nisse, and Stéphan Thomassé. Submodular partition functions. Accepted in SIAM J. discrete Maths. Tech. Report, LABRI RR-1427-07, Univ. Bordeaux, Apr., 2007.

[Bie91] Daniel Bienstock. Graph searching, path-width, tree-width and related problems (a survey). *DIMACS Ser. in Discrete Mathematics and Theoretical Computer Science*, 5:33–49, 1991.

[BK96] Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

[BKK95] Hans L. Bodlaender, Ton Kloks, and Dieter Kratsch. Treewidth and pathwidth of permutation graphs. *SIAM J. Discrete Math.*, 8(4):606–616, 1995.

[BM93] Hans L. Bodlaender and Rolf H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Discrete Math.*, 6(2):181–188, 1993.

[Bod93] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *STOC*, pages 226–234, 1993.

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

[BT97] Hans L. Bodlaender and Dimitrios M. Thilikos. Constructive linear time algorithms for branchwidth. In *ICALP*, pages 627–637, 1997.

[BT04] Hans L. Bodlaender and Dimitrios M. Thilikos. Computing small search numbers in linear time. In *IWPEC*, pages 37–48, 2004.

[FFN07] Fedor V. Fomin, Pierre Fraigniaud, and Nicolas Nisse. Nondeterministic graph searching: From pathwidth to treewidth. *Algorithmica*, 2007. http://dx.doi.org/10.1007/s00453-007-9041-6.

[FT08] Fedor V. Fomin and Dimitrios M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.

[MN08] Frédéric Mazoit and Nicolas Nisse. Monotonicity of non-deterministic graph searching. *Theoretical Computer Science*, 399(3):169–178, 2008.

[Ree92] Bruce A. Reed. Finding approximate separators and computing tree width quickly. In *STOC*, pages 221–228. ACM, 1992.

[RS83] Neil Robertson and Paul D. Seymour. Graph minors. i. excluding a forest. *J. Comb. Theory, Ser. B*, 35(1):39–61, 1983.

[RS86] Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of treewidth. *J. Algorithms*, 7(3):309–322, 1986.

[RS91] Neil Robertson and Paul D. Seymour. Graph minors. x. obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991.

[RS94] Neil Robertson and Paul D. Seymour. Graph minors. xi. circuits on a surface. *J. Comb. Theory, Ser. B*, 60(1):72–106, 1994.

[RS04] Neil Robertson and Paul D. Seymour. Graph minors. xx. wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004.

[ST94] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[TSB00] Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Constructive linear time algorithms for small cutwidth and carving-width. In *ISAAC*, pages 192–203, 2000.

APPENDIX

A Figures

- In Figure 1, a 2-branched partitioning-tree (T, R, σ) of the set $\{a, b, \cdots, k, l\}$ is represented. The vertex $V \in V(T)$ defines the partition $\mathcal{T}_V = \{abfghijkl, c, de\}$, $R \in V(T)$ defines $\mathcal{T}_R = \{abcde, fg, hijkl\}$, and the edge $E \in E(T)$ defines the bi-partition $\mathcal{T}_E = \{ab, cdefghijkl\}$. The black vertices are the branching nodes of $cp(T)$.

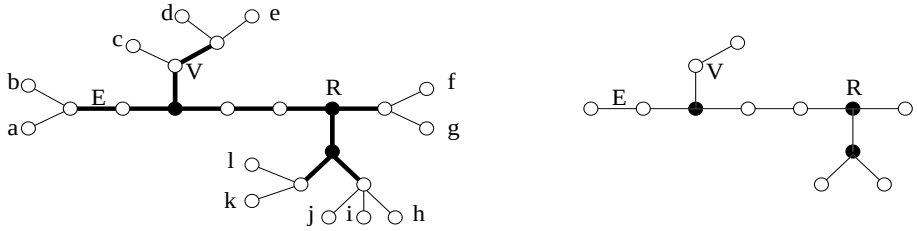

Figure 1: Partitioning-tree of $\{a, b, \cdots, k, l\}$ (left) and its corpse (right).

- In Figure 2, we illustrate two labeled paths P and Q. The black vertices and bold edges of P and Q are those that are represented in $Contr(P)$ and $Contr(Q)$. The vertices of $Contr(P)$ and $Contr(Q)$ are named as the vertices they represent in P and Q. We also illustrate an extension P^* of P and an extension Q^* of Q. The black vertices and bold edges of P^* and Q^* are those in $Init(P^*)$ and $Init(Q^*)$. Finally, we represent a merging of P and Q using the function $F : (x, y) \to x + y$. When merging P and Q, we assume they have same ends, i.e., $a = a'$ and $h = e'$. The black vertices and bold edges the merging R are those in $Matching(R)$.

- In Figure 3, we illustrate the building of the restriction of a partitioning-tree (T, R, σ) of $A = \{abcdefghijkl\}$ to $B = \{bcf\}$. T is represented to the left of Figure 3. The black vertices are the branching nodes of $cp(T)$. We omit the labels of edges for a better readability.

  The top-right figure represents the smallest subtree spanning the leaves of T that map elements of B (Step 1 of the building of $Char((T, R, \sigma), B)$). At Step 2, $dist^* = 1$. The vertices represented by a square are the vertices of K^* (Step 3). During Step 4, we set $branch^*(Y) = branch^*(Z) = 1$ and $branch^*(R^*) = branch^*(X) = branch^*(W) = 0$, and $out^*(Z) = 1$ and $out^*(R^*) = out^*(Y) = out^*(X) = out^*(W) = 0$.

  Finally, during Step 6, any path P between two vertices in K^*, and with no internal vertex in K^* is replaced by $Contr(P)$. The bottom-right figure represents the final result $Char((T, R, \sigma), B)$.

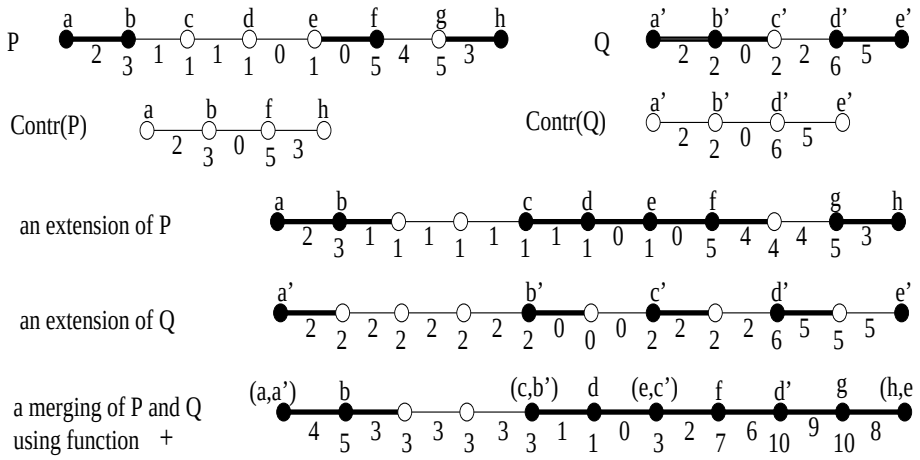

Figure 2: Labeled paths: contraction, extension and merging.

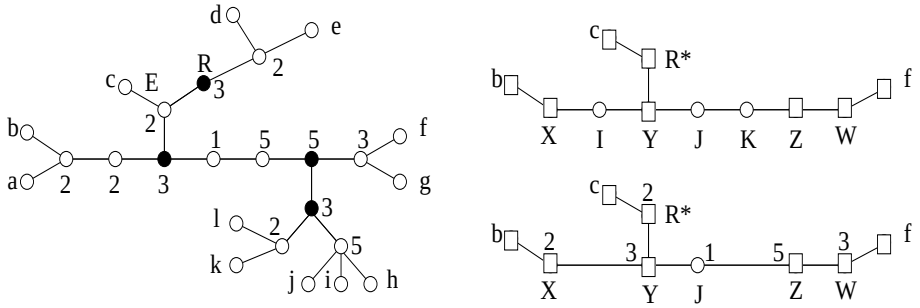

Figure 3: Partitioning-tree (T, R, σ) of $\{abcdefghijkl\}$ (left) and $Char((T, R, \sigma), B)$, its characteristic restricted to $\{bcf\}$ (bottom-right).

- Figure 4 (left) illustrates the procedure of insertion that turns the partitioning-tree (T, R, σ) (Figure 3 left) into a partitioning-tree (S, R_S, σ_S) of $A \cup \{m\}$. We choose to illustrate the case when the insertion is performed by subdividing an edge $E = \{x, y\} \in E(T)$ (x is the parent of y). The bold edges are the new edges. Moreover, we are in the case when y is a leaf and x has exactly one non-leaf child in T. Therefore, x becomes a branching node of $cp(S)$ in the new partitioning-tree.

  Figure 4 (right) illustrates the same operation in $Char((T, R, \sigma), B)$ (Figure 3 bottom-right). That is, Figure 4 (right) illustrates Procedure $IntroduceNode$ executed on an introduce node v of a nice decomposition D, such that, v has one child u, $X_v = \{bcfm\} = X_u \cup \{m\}$ and $A_v = \{abcdefghijklm\} = A_u \cup \{m\}$.

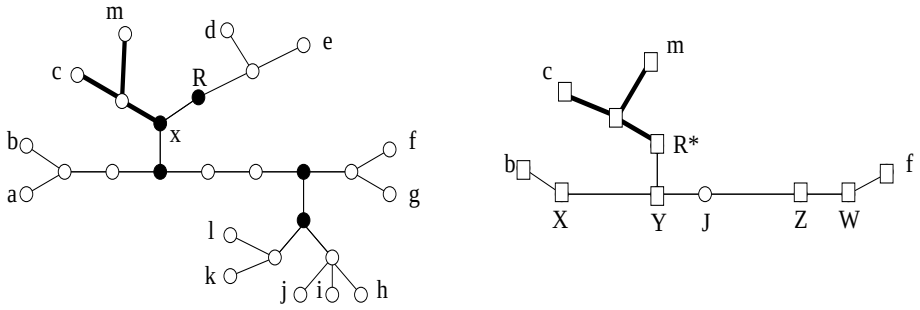

Figure 4: Insertion of $\{m\}$ in (T, R, σ) by subdividing E (left). Same operation in $Char((T, R, \sigma), B)$ (right).

- Figure 5 represents the merging of a partitioning-tree (T, R, σ) of $A \cup B = \{abcdefghijkl\}$ (top-left) and a partitioning-tree (S, R_S, σ_S) of $A \cup C = \{bcfgmnopqrst\}$ (top-right), with $A = \{bcfg\}$. The bold edges are the edges of the smallest subtrees T' and S' (of T and S) spanning the leaves that map elements of A. The black vertices are the vertices of the **common** structure *Struct* of T and S obtained by contracting all degree-2 vertices of T' and S'. Recall we also impose that $R_S \in V(T')$.

  The graph in the middle results of the identifying of the vertices in this common structure. Now, any edge in *Struct* corresponds to two paths in the current graph. We compute a merging of these paths. A possible result is illustrated Figure 5 (bottom).

B On labeled paths.

Let Φ be a monotone partition function. Any partitioning-tree (T, σ) can be viewed as a labeled graph, where any $v \in V(T)$ is labeled with $\Phi(\mathcal{T}_v)$ and any $e \in E(T)$ is labeled with $\Phi(\mathcal{T}_e)$. Because Φ is monotone, the label of an edge is at most the label of its ends. In this section, we detail operations over labeled paths, that will serve as subroutine in the forthcoming sections.

B.1 Labeled paths.

A labeled path P is a path $\{v_0, v_1, \cdots, v_n\}$ where any vertex v_i is labeled with an integer $\ell(v_i)$, and any edge $e_i = \{v_{i-1}, v_i\}$ with an integer $\ell(e_i)$. We assume moreover that the label of any edge is at most the minimum label of its ends.

A vertex $v_i \in V(P)$ or an edge $\{v_i, v_{i+1}\}$ is smallest than $v_j \in V(P)$ if $i < j$. Similarly v_i (resp., $\{v_i, v_{i+1}\}$) is smallest than $\{v_j, v_{j+1}\}$ if $i < j$. We define $\max(P)$ as the maximum integer labeling an edge or a vertex of P. Similarly, we define $\min(P)$. An *extension* of a labeled path P is any path obtained by subdividing some edges of P an arbitrary number of times. Both edges and the vertex resulting from the subdivision of an edge e are labeled with $\ell(e)$. Let P^* be an extension of P. The initial elements $Init(P^*) \subseteq V(P^*) \cup E(P^*)$ of P^* are

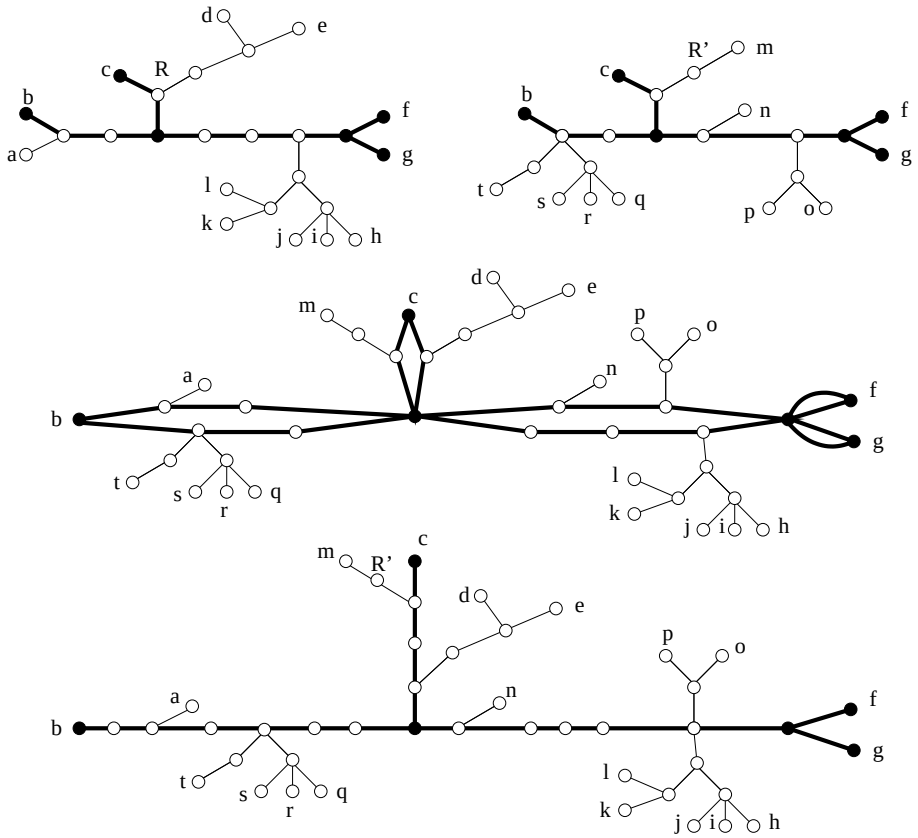

Figure 5: Merging of a partitioning-tree (T, R, σ) of $A \cup C = \{abcdefghijkl\}$ and a partitioning-tree (S, R_S, σ_S) of $B \cup C = \{bcfgmnopqrst\}$, with $C = \{bcfg\}$.

the vertices of P^* that do not result from the subdivision of an edge in P, and the edges of P^* the smallest end of which are in $Init(P^*)$. Given two labeled paths P and Q, we say that $P \preceq Q$ if there is an extension $P^* = \{p_1, \cdots, p_r\}$ of P and an extension $Q^* = \{q_1, \cdots, q_r\}$ of Q with same length, and such that $\ell(p_i) \leq \ell(q_i)$ and $\ell(\{p_i, p_{i+1}\}) \leq \ell(\{q_i, q_{i+1}\})$ for any $i \leq r$. For any function $F : \mathbb{N} \to \mathbb{N}$, let $F(P)$ denote the path $\{v_0, v_1, \cdots, v_n, v_{n+1}\}$ where any label ℓ has been replaced by $F(\ell)$. If $P = \{v_1, \cdots, v_n\}$ and $Q = \{w_1, \cdots, w_m\}$ are two labeled paths with a common end $v_n = w_1$ and vertex disjoint otherwise, their *concatenation* $P \odot Q$ is the labeled path $\{v_1, \cdots, v_n, w_2, \cdots, w_m\}$.

B.2 Contraction of a labeled path.

In this section, we define an operation on labeled paths that will be widely used in the next sections. For this purpose, we revisit the notion of *typical sequence* of a sequence of integers [BK96]. Roughly, the goal of the following operation is to contract some edges and vertices of P that are not "necessary" to remember the variations of the sequence $(\ell(v_0), \ell(e_1), \ell(v_1), \cdots, \ell(e_n), \ell(v_n))$.

First, let us recall the definition of the typical sequence of a sequence of integers [BK96]. Let $S = (s_i)_{i \leq 2n-1}$ be a sequence of integers. Its typical sequence $\tau(S)$ is obtained by iterating the following operations while it is possible: (1) if there is $i < |S|$ such that $s_i = s_{i+1}$, remove s_{i+1} from S, and (2) if there are $i < j - 1 < |S|$, and either, for any $i \leq k \leq j$, $s_i \leq s_k \leq s_j$, or, for any $i \leq k \leq j$, $s_i \geq s_k \geq s_j$, remove s_k from S for any $i < k < j$. Note that the order in which the operations are executed is not relevant, therefore $\tau(S)$ is uniquely defined.

The *contraction* $Contr(P)$ is the path obtained from P, with same ends, by contracting some edges and vertices obtained by the following procedure. Start with one integral variable $m = 1$. While $m \neq n$, do the following. Let i, $m \leq i \leq n-1$, be the greatest index such that, for any $m \leq j \leq i$, $\ell(e_m) \leq \ell(e_j)) \leq \ell(v_i)$ and $\ell(e_m) \leq \ell(v_j)) \leq \ell(v_i)$. Contract all vertices and edges between e and v_i. Then, set m to the greatest index such that, for any $i < j \leq m$, $\ell(v_i) \geq \ell(e_j) \geq \ell(e_m)$ and $\ell(v_i) \geq \ell(v_j) \geq \ell(e_m)$. Contract all vertices and edges between v_i and e_m. Edges and vertices of $Contr(P)$ keep their initial label (cf. Figure 2 in Appendix A).

The crucial property of $Contr(P) = \{v_0 = v'_0, v'_1, \cdots, v'_{p-1}, v'_p = v_n\}$ is that the sequence $S' = (\ell(e'_1), \ell(v'_1), \cdots, \ell(e'_{p-1}), \ell(v'_{p-1}), \ell(e'_p))$ (where $e'_i = \{v'_{i-1}, v'_i\}$) is "almost" the typical sequence of the sequence $S = (\ell(e_1), \ell(v_1), \cdots, \ell(e_n))$. More precisely, if $\ell(e'_1) \neq \ell(v'_1)$, then $S' = \tau(S)$, otherwise $(\ell(v'_1), \ell(e'_2), \cdots, \ell(v'_{p-1}), \ell(e'_p)) = \tau(S)$ ($\tau(S)$ denotes the typical sequence of S). Moreover, it is important to note that any $v \in V(Contr(P))$ ($e \in E(Contr(P))$) *represents* a unique $v^* \in V(P)$ ($e^* \in E(P)$).

Lemma 1. *Let P and Q be two labeled paths. Let P^* be an extension of P.*
1. $Contr(P^*) = Contr(P) = Contr(Contr(P))$.
2. Let $F : \mathbb{N} \to \mathbb{N}$ be any strictly increasing function. $Contr(F(Contr(P))) = Contr(F(P))$.
3. If $P \preceq Q$, then $Contr(P) \preceq Contr(Q)$.
4. $Contr(P \odot Q) = Contr(Contr(P) \odot Contr(Q))$

Lemma 2. *Let $P = \{v_0, \cdots, v_n\}$ be a labeled path and $Contr(P) = \{w_0, \cdots, w_p\}$. Let $i \leq p$. Let $Contr'(P) = \{w_0, \cdots, w_i, x, w_{i+1}, \cdots, w_p\}$ be the extension of $Contr(P)$ obtained by subdividing once $e_i = \{w_i, w_{i+1}\}$. Let $e^*_i = \{v_j, v_{j+1}\}$ be the edge of P represented by e_i, and $P' = \{v_0, \cdots, v_j, y, v_{j+1}, \cdots, v_n\}$ be the extension of P obtained by subdividing once e^*_i. Let $Q_1 = \{w_0, \cdots, x\}$, $Q_2 = \{x, \cdots, w_p\}$, $P'_1 = \{v_0, \cdots, y\}$ and $P'_2 = \{y, \cdots, v_n\}$.*

Then $Contr(P'_1) = Contr(Q_1)$ and $Contr(P'_2) = Contr(Q_2)$.

B.3 Merging of labeled paths.

Now, we present an operation that merges two labeled paths $P = \{v_1, \cdots, v_n\}$ and $Q = \{w_1, w_2, \cdots, w_m\}$ with common ends (i.e., $v_1 = w_1$ and $w_m = v_n$) and vertex-disjoint otherwise. For any $i < n$, $e_i = \{v_i, v_{i+1}\}$, and for any $i < m$, $f_i = \{w_i, w_{i+1}\}$. Let $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Let $P^* = \{p_1, \cdots, p_h\}$ be an extension of P and $Q^* = \{q_1, \cdots, q_h\}$ be an extension of Q with same length. A *merging R of P and Q using F* is the labeled path $\{r_1, \cdots, r_h\}$, where r_i is labeled $F(\ell(p_i), \ell(q_i))$ and $\{r_i, r_{i+1}\}$ is labeled $F(\ell(\{p_i, p_{i+1}\}), \ell(\{q_i, q_{i+1}\}))$. We say that r_i matches $p_i = Peer(q_i) \in V(P^*)$ and $q_i = Peer(p_i) \in V(Q^*)$, and $\{r_i, r_{i+1}\}$ matches $\{p_i, p_{i+1}\} = Peer(\{q_i, q_{i+1}\}) \in E(P^*)$ and $\{q_i, q_{i+1}\} = Peer(\{p_i, p_{i+1}\}) \in E(Q^*)$. Let $Matching(R)$ be the set of pairs (p_i, q_i) such that $p_i \in Init(P^*)$ or $q_i \in Init(Q^*)$, and the pairs $(\{p_i, p_{i+1}\}, \{q_i, q_{i+1}\})$ such that $\{p_i, p_{i+1}\} \in Init(P^*)$ or $\{q_i, q_{i+1}\} \in Init(Q^*)$.

Let R^* be a merging of $Contr(P)$ and $Contr(Q)$ using F. We say that a merging R of P and Q *respects* R^* if, for any $(a, b) \in Matching(R^*)$, $(a', b') \in Matching(R)$ where a' is the element of P represented by a in $Contr(P)$, and b' is the element of Q represented by b in $Contr(P)$.

Let us assume $P = P_1 \odot P_2 \odot \cdots \odot P_r$, i.e., $P = \{v_1, \cdots, v_n\}$, $1 = t_1 < t_2 < \cdots < t_r = n$ are r integers, and $P_i = \{v_{t_i}, \cdots, v_{t_{i+1}}\}$, $i < r$. Moreover, $Q = Q_1 \odot Q_2 \odot \cdots \odot Q_q$, that is, $Q = \{w_1, \cdots, w_m\}$, $1 = t'_1 < t'_2 < \cdots < t'_q = m$ are q integers, and $Q_i = \{w_{t'_i}, \cdots, w_{t'_{i+1}}\}$, $i < q$. Any merging M of P and Q will be noted $M = M_1 \odot M_2 \odot \cdots \odot M_r$, where $M = \{y_1, \cdots, y_k\}$, $k_1 < \cdots < k_r$ are the indexes such y_{k_i} matches a vertex v_{t_j}, $j \leq r$, with $Peer(v_{t_j})$, or y_{k_i} matches a vertex $w_{t'_j}$, $j \leq q$, with $Peer(w_{t'_j})$, and $M_i = \{y_{k_i}, \cdots, y_{k_{i+1}}\}$ for any $i < u$.

Lemma 3. *Let $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ strictly increasing on both coordinates. Let $P = P_1 \odot P_2 \odot \cdots \odot P_r$ and $P' = Contr(P_1) \odot Contr(P_2) \odot \cdots \odot Contr(P_r)$. Let $Q = Q_1 \odot Q_2 \odot \cdots \odot Q_q$ and $Q' = Contr(Q_1) \odot Contr(Q_2) \odot \cdots \odot Contr(Q_q)$. Let $M' = M'_1 \odot M'_2 \odot \cdots \odot M'_r$ be a merging of P' and Q' using F, and let $M = M_1 \odot M_2 \odot \cdots \odot M_r$ be a merging of P and Q using F that respects M'. Then, $Contr(M'_1) \odot \cdots \odot Contr(M'_r) = Contr(M_1) \odot \cdots \odot Contr(M_r)$.*

Moreover, if $P'' = P''_1 \odot P''_2 \odot \cdots \odot P''_r$, with $P''_i \preceq Contr(P_i)$ for any $i \leq r$, and $Q'' = Q''_1 \odot Q''_2 \odot \cdots \odot Q''_q$, with $Q''_i \preceq Contr(Q_i)$ for any $i \leq q$. Then, there is a merging $M'' = M''_1 \odot \cdots \odot M''_r$ of P'' and Q'' using F, such that $M''_i \preceq Contr(M'_i)$ for any $i \leq r$.

C Proof of Lemma 7: Introduce node

Since Φ is closed under taking subset, A_v admits a q-branched partitioning-tree with Φ-width at most k only if A_u does. Therefore, we can assume that $FSC(u) \neq \emptyset$, otherwise, A_v does not admit a q-branched partitioning-tree with Φ-width at most k, and $FSC(v) = \emptyset$. The proof is twofold. We first prove that the set $FSC(v)$ returned by Procedure *IntroduceNode* is a set of characteristics of A_v restricted to X_v, then we prove it is full. The proof is illustrated in Figure 6.

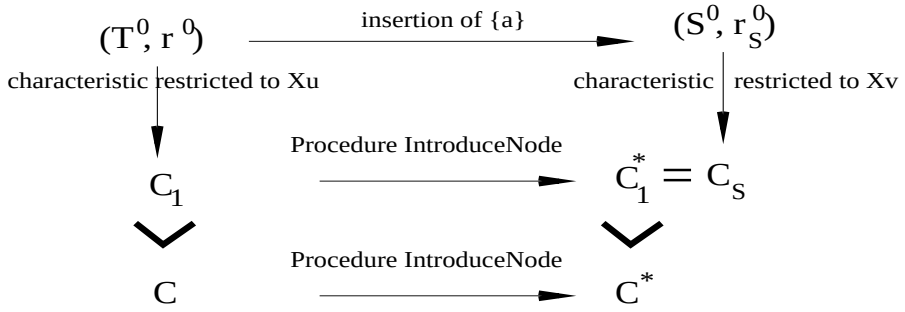

Figure 6: Proof of correctness of Procedure $IntroduceNode$

C.1 $FSC(v)$ is a set of characteristics of A_v restricted to X_v

Let $C_1^* = ((T_1^*, r_1^*, \sigma_1^*), \ell_1^*, K_1^*, dist_1^*, out_1^*, branch_1^*) \in FSC(v)$.
Let $C_1 = ((T_1, r_1, \sigma_1), \ell_1, K_1, dist_1, out_1, branch_1) \in FSC(u)$ be the characteristic transformed by procedure $IntroduceNode$ to compute C_1^*. Since $C_1 \in FSC(u)$, it is the characteristic of a partitioning-tree (T^0, r^0, σ^0) of A_u restricted to X_u. We prove that C_1^* is the characteristic of the partitioning-tree (S^0, r_S^0, σ_S^0) of A_v restricted to X_v, where (S^0, r_S^0, σ_S^0) is obtained from (T^0, r^0, σ^0) by inserting $\{a\} = X_v \setminus X_u$.

We assume that C_1^* is obtained from C_1 by inserting $\{a\}$ in the edge$f = \{v_{top}, v_{bottom}\}$. This corresponds to Case 2 of Step 1 of Procedure $IntroduceNode$. Case 1 can be proved in a similar way, and then we omit the proof here.

$f \in E(T_1)$, therefore, by definition of a characteristic, it represents an edge $f^0 \in E(T^0)$. Let $(S^0, r_S^0, \sigma_S^0) = ins((T^0, r^0, \sigma^0), f^0, a)$, and let $C_S = ((T_S, r_S, \sigma_S), \ell_S, K_S, dist_S, out_S, branch_S) = Char((S^0, r_S^0, \sigma_S^0), X_v)$ be its characteristic restricted to X_v. We prove that $C_1^* = C_S$.

We first prove that $K_1^* = K_S$. Indeed, let S' be the smallest subtree of S^0 the leaves of which map X_v, and let r'_S be the vertex in S' that is closest to r_S^0. K_S is the set of vertices that are leaves in (S', r'_S), parents of leaves, branching nodes of (S', r'_S) or branching nodes of $cp(S^0)$ in $V(S')$. Moreover, let T' be the smallest subtree of T^0 the leaves of which map X_u, and let r' be the vertex in T' that is closest to r^0. Since S^0 is obtained from T^0 by subdividing f^0 and adding a new leaf v_{leaf} adjacent to the new vertex v_{att}, and moreover, v_{leaf} maps $\{a\}$, S' is obtained from T' by subdividing f^0. Therefore, K_S is the set of vertices that are leaves in (T', r'), or parents of leaves, or branching nodes of (T', r') or branching nodes of $cp(T^0)$ in $V(T')$, or v_{att} or v_{leaf}. In other words, $K_S = K_1 \cup \{v_{att}, v_{leaf}\} = K_1^*$ (Step 1 of Procedure $IntroduceNode$).

Then, we prove that $T_1^* = T_S$ and $\ell_1^* = \ell_S$. T_S is obtained in the following way. Any $t \in V(S^0)$ defines a partition $\mathcal{S}_t^0$ of A_v and receives label $\ell^0(t) = \Phi_{A_v}(\mathcal{S}_t^0)$. Similarly, any $e \in E(S^0)$ defines a partition $\mathcal{S}_e^0$ of A_v and receives label $\ell^0(e) = \Phi_{A_v}(\mathcal{S}_e^0)$. Then, T_S is obtained from T'_S by replacing any path $P(x, y)$ between two vertices $x, y \in K_S$ by $Contr(P(x, y))$. Because $K_S = K_1^*$, $x, y \in K_1^*$. Let us describe the path $P_1^*(x, y)$ between x and y in T_1^*. We prove that $P_1^*(x, y) = Contr(P(x, y))$.

If $P(x,y) = \{v_{att}, v_{leaf}\}$, the result is obvious by Step 2 of Procedure *IntroduceNode*. Now, let us assume that $P(x,y) \neq \{v_{att}, v_{leaf}\}$ and $v_{att} \notin \{x,y\}$. Then, $P(x,y)$ represents a path P in T^0, and more precisely in T'. Any $t \in V(P)$ defines a partition $\mathcal{T}_t^0$ of A_u and receives label $\ell_1(t) = \Phi_{A_u}(\mathcal{T}_t^0) = \Phi_{A_v}(\mathcal{S}_t^0) \cap A_u$. Similarly, any $e \in E(P)$ defines a partition $\mathcal{T}_e^0$ of A_u and receives label $\ell_1(e) = \Phi_{A_u}(\mathcal{T}_e^0) = \Phi_{A_v}(\mathcal{S}_e^0) \cap A_u$. When computing $Char((T^0, r^0, \sigma^0), X_u)$, P is replaced by $Contr(P)$. The key point is that any internal vertex and edge of $Contr(P)$ defines the same partition $\mathcal{P}$ of X_v. Then, Procedure *IntroduceNode* modifies the labels of edges and vertices of $Contr(P)$ by applying the strictly increasing function $F_{\Phi,\mathcal{P}} : x \to F_\Phi(x, \mathcal{P}, a)$ (Step 3 of Procedure *IntroduceNode*). Let $F_{\Phi,\mathcal{P}}(Contr(P))$ be the path obtained in this way. Finally, Procedure *IntroduceNode* replaces $F_{\Phi,\mathcal{P}}(Contr(P))$ by $Contr(F_{\Phi,\mathcal{P}}(Contr(P)))$. By definition of F_Φ, $F_\Phi(\Phi_{A_u}(\mathcal{T}_t^0), \mathcal{P}, a) = F_\Phi(\Phi_{A_u}(\mathcal{S}_t^0 \cap A_u), \mathcal{S}_t^0 \cap X_v, a) = \Phi_{A_v}(\mathcal{S}_t^0)$ and $F_\Phi(\Phi_{A_u}(\mathcal{T}_e^0), \mathcal{P}, a) = F_\Phi(\Phi_{A_u}(\mathcal{S}_e^0 \cap A_u), \mathcal{S}_e^0 \cap X_v, a) = \Phi_{A_v}(\mathcal{S}_e^0)$ for any vertex $v \in V(P)$ and edge $e \in V(P)$. Therefore, $F_{\Phi,\mathcal{P}}(P) = P(x,y)$ By Item 2 of Lemma 1, $P_1^*(x,y) = Contr(F_\Phi(Contr(P))) = Contr(F_{\Phi,\mathcal{P}}(P)) = Contr(P(x,y))$.

Let x and y be the vertices in K_1 such that the path P of T^0 between x and y contains the edge f^0. It remains to prove that $P_1^*(x, v_{att}) = Contr(P(x, v_{att}))$ and $P_1^*(v_{att}, y) = Contr(P(v_{att}, y))$. In S^0, the edge f^0 is subdivided in P. Let R be the obtained path between x and y and containing v_{att}. $R = P(x, v_{att}) \odot P(v_{att}, y)$. By computing $Char((S^0, r_S^0, \sigma_S^0), X_v)$, $P(x, v_{att})$ is replaced by $Contr(P(x, v_{att}))$ and $P(v_{att}, y)$ is replaced by $Contr(P(v_{att}, y))$. By computing $Char((T^0, r^0, \sigma^0), X_u)$, the vertices t (resp., edges e) of P that defines are labeled with $\Phi_{A_u}(\mathcal{P}_t)$ (resp., $\Phi_{A_u}(\mathcal{P}_e)$). In T^0, let J be the path obtained from P by subdividing f^0. Let J_1 be the subpath of R between x and v_{att}, and let J_2 be the subpath of R between v_{att} and y. Then, P is replaced by $Contr(P)$. By Procedure *IntroduceNode*, the edge f (that represents f^0 in $Contr(P)$) is subdivided (inserting the new vertex v_{att}). Let Q be the obtained path between x and y and containing v_{att}. Let Q_1 be the subpath of Q between x and v_{att}, and let Q_2 be the subpath of Q between v_{att} and y. $Q = Q_1 \odot Q_2$. By Lemma 2, $Contr(J_1) = Contr(Q_1)$ and $Contr(J_2) = Contr(Q_2)$. Let $\mathcal{P}$ be the partition of X_v defines by the vertices and edges of Q_1, and let $\mathcal{P}'$ be the partition of X_v defines by the vertices and edges of Q_2. Procedure *IntroduceNode* then replaces Q_1 by $P_1^*(x, v_{att}) = Contr(F_{\Phi,\mathcal{P}}(Q_1))$ and Q_2 by $P_1^*(v_{att}, y) = Contr(F_{\Phi,\mathcal{P}}(Q_2))$. By Item 2 of Lemma 1 and by definition of $F_{\Phi,\mathcal{P}}$, $Contr(F_{\Phi,\mathcal{P}}(Q_1)) = Contr(F_{\Phi,\mathcal{P}}(Contr(Q_1))) = Contr(F_{\Phi,\mathcal{P}}(Contr(J_1))) = Contr(F_{\Phi,\mathcal{P}}(J_1)) = Contr(P(x, v_{att}))$. Therefore, $P_1^*(x, v_{att}) = Contr(P(x, v_{att}))$. Similarly, $P_1^*(v_{att}, y) = Contr(P(v_{att}, y))$.

Since $K_1^* = K_S$, and for any $x, y \in K_S$, $P_1^*(x,y) = Contr(P(x,y))$, we get that $T_1^* = T_S$ and $\ell_1^* = \ell_S$. It remains to prove that $(dist_1^*, branch_1^*, out_1^*) = (dist_S, branch_S, out_S)$. Recall that S' (resp., T') is the smallest subtree of S^0 (resp., T^0) the leaves of which map X_v (resp., X_u), and r'_S (resp., r') is the vertex in S' (resp., T') that is closest to r_S^0 (resp., r^0).

By procedure *IntroduceNode*, $dist^* = dist_1$, i.e., the number of branching nodes in T^0 between r^0 and r' which is the number of branching nodes in S^0 between r_S^0 and r'_S, i.e., $dist_S$. Now, for any vertex t in K_1^*, $out_1^*(t)$ is the maximum number of branching nodes on a path between t and a leaf in $A_u \setminus X_u$ every internal vertices of which are different from

r^0 and in $T' \setminus T^0$. It is also the maximum number of branching nodes on a path between t and a leaf in $A_v \setminus X_v = A_u \setminus X_u$ every internal vertices of which are different from r^0_S and in $S' \setminus S^0$, i.e., $out_S(t)$. For any vertex t in K^*_1, $branch^*_1(t) = 1$ if and only if $branch_1(t) = 1$ or t is the parent-end of f, has exactly one non-leaf child that is not the other end of f (Step 4 of Procedure *IntroduceNode*). That is, $branch^*_1(t) = 1$ if and only if t is a branching node of $cp(S^0)$, i.e., $branch^*_1(t) = branch_S(t)$.

Therefore, we proved that $C^*_1 = Char((S^0, r^0_S, \sigma^0_S), X_v)$. By Step 5 of Procedure *IntroduceNode*, $dist + height \leq q$ (where *height* is the br-height of r^*_1) and $\ell(v) \leq k$ for any internal vertex $v \in V(T)$, and $\ell(e) \leq k$ for any edge $e \in E(T)$. Therefore, by Lemma 5, (S^0, r^0_S, σ^0_S) is a q-branched partitioning-tree with Φ-width at most k.

C.2 $FSC(v)$ is full

Let (S^0, r^0_S, σ^0_S) be a q-branched partitioning-tree of A_v restricted to X_v with Φ-width at most k. Let v_{leaf} be the leaf of S^0 that maps $\{a\} = X_v \setminus X_u$. Let v_{att} be the parent of v_{leaf}. Let (T_1, r_1, σ_1) be the partitioning-tree of A_u restricted to X_u such that $(S^0, r^0_S, \sigma^0_S) = ins((T^0, r^0, \sigma^0), v_{att}, a)$.

Because $FSC(u)$ is a full set of characteristic of A_u restricted to X_u, there is $C \in FSC(u)$ such that $C \preceq C_1 = Char((T^0, r^0, \sigma^0), X_u)$. In the previous subsection, we proved that there is an execution of Procedure *IntroduceNode* on C_1 that computes $C^*_1 = Char((S^0, r^0_S, \sigma^0_S), X_v)$. We prove that a similar execution of Procedure *IntroduceNode* on C computes $C^* \preceq C^*_1$.

Let $C_1 = ((T_1, r_1, \sigma_1), \ell_1, K_1, dist_1, out_1, branch_1)$ and $C = ((T, r, \sigma), \ell, K, dist, out, branch)$. By definition of the ordering over the characteristics, (T_1, r_1, σ_1) and (T, r, σ) are isomorphic and $K_1 = K$. Let us assume that C^*_1 is obtained from C_1 by subdividing an edge f. For any $x, y \in K_1$ such that the path $P'(x, y)$ does not contain f, let $P(x, y)$ be the path between x and y in C. Let $\mathcal{P}$ be the partition of X_v defined by the vertices and edges of $P(x, y)$. The same partition is defined by the internal vertices and edges of $P'(x, y)$. By definition of the ordering, $P(x, y) \preceq P'(x, y)$. Procedure *IntroduceNode* replaces $P(x, y)$ by $Contr(F_{\Phi,\mathcal{P}}(P(x, y)))$ and $P'(x, y)$ by $Contr(F_{\Phi,\mathcal{P}}(P'(x, y)))$. By Item 3 of Lemma 1 and because $F_{\Phi,\mathcal{P}}$ is strictly increasing, $Contr(F_{\Phi,\mathcal{P}}(P(x, y))) \preceq Contr(F_{\Phi,\mathcal{P}}(P'(x, y)))$. Now, let $x, y \in K$ be the vertices such that $P'(x, y)$ does contain f, and let $P(x, y)$ be the corresponding path in C. By definition of the ordering, there is an extension $\{v_1, \cdots, v_n\}$ of $P(x, y)$ and an extension $\{v'_1, \cdots, v'_n\}$ of $P'(x, y)$ with $\ell(\{v_i, v_{i+1}\}) \leq \ell_1(\{v'_i, v'_{i+1}\})$ for any $i \leq n$. Let $i \leq n$ such that $\{v'_i, v'_{i+1}\} = f$, and let f' be the edge of $P(x, y)$ corresponding to $\{v_i, v_{i+1}\}$. We consider the execution of *IntroduceNode* that inserts a in C by subdividing f'. In C_1, by subdividing f, Procedure *IntroduceNode* divides $P'(x, y)$ into two paths P'_1 and P'_2. In C, by subdividing f', Procedure *IntroduceNode* divides $P(x, y)$ into two paths $P_1 \preceq P'_1$ and $P_2 \preceq P'_1$. Then, Procedure *IntroduceNode* applies the function F_Φ on each of these paths and contracts them. By Item 3 of Lemma 1 and because $F_{\Phi,\mathcal{P}}$ is strictly increasing, we get the result.

The fact that $dist^* \leq dist^*_1$ and $out^*(t) \leq out^*_1(t)$ and $branch^*(t) \leq branch^*_1(t)$ for any $t \in K$ holds trivially. Therefore, $C^* \preceq C^*_1$.

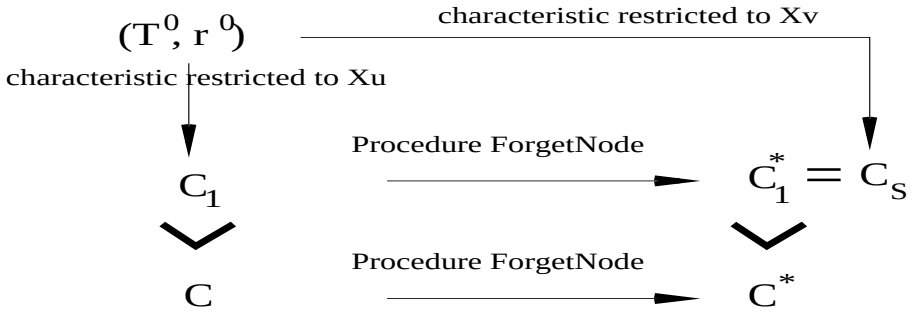

Figure 7: Proof of correctness of Procedure $ForgetNode$

D Proof of Lemma 8: Forget node

Since $A_v = A_u$, A_v admits a q-branched partitioning-tree with Φ-width at most k only if A_u does. Therefore, we can assume that $FSC(u) \neq \emptyset$, otherwise, A_v does not admit a q-branched partitioning-tree with Φ-width at most k, and $FSC(v) = \emptyset$. The proof is twofold. We first prove that the set $FSC(v)$ returned by Procedure $ForgetNode$ is a set of characteristics of A_v restricted to X_v, then we prove it is full. The proof is illustrated in Figure 7.

D.1 $FSC(v)$ is a set of characteristics of A_v restricted to X_v

Let $C_1^* = ((T_1^*, r_1^*, \sigma_1^*), \ell_1^*, K_1^*, dist_1^*, out_1^*, branch_1^*) \in FSC(v)$.
Let $C_1 = ((T_1, r_1, \sigma_1), \ell_1, K_1, dist_1, out_1, branch_1) \in FSC(u)$ be the characteristic transformed by Procedure $ForgetNode$ to compute C_1^*. Since $C_1 \in FSC(u)$, it is the characteristic of a partitioning-tree (T^0, r^0, σ^0) of A_u restricted to X_u. We prove that C_1^* is the characteristic of (T^0, r^0, σ^0) of $A_v = A_u$ restricted to $X_v = X_u \setminus \{a\}$.

Let $C_S = Char((T^0, r^0, \sigma^0), X_v) = ((T_S, r_S, \sigma_S), \ell_S, K_S, dist_S, out_S, branch_S)$. Let us show that $C_S = C_1^*$.

Let R be the smallest subtree of T^0 the leaves of which map X_u, and let r_R be the vertex in R that is closest to r^0. K_1 is the set of vertices that are leaves in (R, r_R), parents of leaves, branching nodes of (R, r_R) or branching nodes of $cp(T^0)$ in $V(R)$. C_1 is obtained by replacing any path $P(x, y)$ $(x, y \in K_1)$ by $Contr(P(x, y))$.

Let T' be the smallest subtree of T^0 the leaves of which map the elements of X_v. Let r' be the vertex of T' that is closest to r^0. Let v_{att}^0 be the vertex of T' that is closest to the leaf v_{leaf}^0 that maps $\{a\} = X_u \setminus X_v$. Let P^0 be the path between these two vertices. Note that, v_{att}^0 has degree at least two in T' or $T' = \{v_{att}^0\}$. In the latter case, T_1^* and T_S are reduced to a single vertex $r_S = r_1^*$ representing v_{att}^0 and with same labels. Let us assume that v_{att}^0 has degree at least two in T'. Note that $T' = R \setminus (P \setminus v_{att}^0)$.

There are three cases to be considered. In any case, there is a vertex v_{att} of T_1 that represents v^0_{att} (this is because v^0_{att} has degree at least three in the smallest subtree of T^0 the leaves of which map the elements of X_u), and a path P between v_{att} and v_{leaf} (the vertex that maps $\{a\}$ in T_1) representing P^0.

- First assume that v^0_{att} has degree at least three in T' or is a branching node of $cp(T^0)$, or is the parent of a leaf in T', and r^0 does not belong to the subtree W obtained from T^0 by removing v^0_{att} and that contains v^0_{leaf}. In this case, r does not belong to $P \setminus v_{att}$, v_{att} has degree at least three in T_1, or $branch(v_{att}) = 1$ or v_{att} is the root of T_1.

  By the definition of a characteristic, K_S is the set of vertices that are leaves in (T', r'), parents of leaves, branching nodes of (T', r') or branching nodes of $cp(T^0)$ in $V(T')$. Note that $v_{att} \in K_S$. That is, $K_S = K_1 \setminus (P \setminus v_{att})$. Therefore, T_S is obtained from T_1 by removing the path $P \setminus v_{att}$. No labels are modified but $out_S(v_{att})$ that is set to the maximum number of branching nodes on a path between v_{att} and a leaf in $A_v \setminus X_v$ every internal vertices of which are different from r^0 and in $T^0 \setminus T'$.

  Now, Procedure $ForgetNode$ does the same: T^*_1 is obtained from T_1 by removing $P \setminus v_{att}$, and $K^*_1 = K_1 \setminus (P \setminus v_{att})$. Moreover, Procedure $ForgetNode$ set $out^*_1(v_{att})$ to $\max\{out_1(v_{att}), height(w)\}$, where w is the neighbour of v_{att} in P. To conclude the proof, it is sufficient to remark that this value is exactly the maximum number of branching nodes on a path, not containing r^0, between v_{att} and a leaf in $A_v \setminus X_v$ every internal vertices of which are in $T^0 \setminus T'$.

- Assume that r^0 belongs to the subtree W obtained from T^0 by removing v^0_{att} and that contains v^0_{leaf}. In this case, r_1 belongs to P and v_{att} has degree at least three in T.

  By the definition of a characteristic, T_S can be obtained from T_1 by removing the path $P \setminus v_{att}$. Moreover, the root of T_S is set to v_{att}. Again, $K_S = K_1 \setminus (P \setminus v_{att})$, and no labels are modified, because the partitioning-tree (T^0, r^0, σ^0) that we consider remains unchanged.

  Again, Procedure $ForgetNode$ does the same: T^*_1 is obtained from T_1 by removing $P \setminus v_{att}$ and setting v_{att} as the new root and $K^*_1 = K_1 \setminus (P \setminus v_{att})$. Moreover, Procedure $ForgetNode$ set $dist^*_1$ to $dist^*_1 = dist_1 + p$, with p being the number of nodes of T_1 in the path between v_{att} and r_1 (excluding v_{att}). Therefore, $dist^*_1$ is set to the number of branching nodes between v^0_{att} and r^0 (excluding v^0_{att}).

  $out^*_1(v_{att})$ is set to $\max\{out_1(v_{att}), height(w)\}$, where w is the neighbour of v_{att} in P. To conclude the proof, it is sufficient to remark that this value is exactly the maximum number of branching nodes on a path, not containing r^0, between v^0_{att} and a leaf in $A_v \setminus X_v$ every internal vertices of which are in $T^0 \setminus T'$.

- Finally, let us assume that v^0_{att} has degree exactly two in T', v^0_{att} is not a branching node of $cp(T^0)$, nor the parent of a leaf in T', and r^0 does not belong to the subtree

W obtained from T^0 by removing v^0_{att} and that contains v^0_{leaf}. In this case, $v_{att} \notin K_S$, that is, $K_S = K_1 \setminus P$. Therefore, T_S is obtained from T', by replacing any path $P(x,y)$ $(x,y \in K_S)$ by $Contr(P(x,y))$.

Let $w_1, w_2 \in K_S$ such that $v_{att} \in P(w_1,w_2)$. The key point is that, in C_1, there are two paths $P(w_1,v_{att})$ and $P(v_{att},w_2)$ that are replaced by $Contr(P(w_1,v_{att}))$ and $Contr(P(v_{att},w_2))$, whereas in C_S, $P(w_1,w_2)$ is replaced by $Contr(P(w_1,w_2))$.

In this case, Procedure $ForgetNode$ proceeds as follow. T_1^* is obtained from T_1 by removing $P \setminus v_{att}$ and replacing the paths $Contr(P(w_1,v_{att})) \odot Contr(P(v_{att},w_2))$ by $Contr(Contr(P(w_1,v_{att})) \odot Contr(P(v_{att},w_2)))$. By Item 4 of Lemma 1, this is exactly $Contr(P(w_1,w_2))$.

D.2 $FSC(v)$ is full

Let (T^0,r^0,σ^0) be a q-branched partitioning-tree of A_v with Φ-width at most k. We need to prove that there is a characteristic $C^* \in FSC(v)$ such that $C^* = ((T^*,r^*,\sigma^*),\ell^*,K^*,dist^*,out^*,branch^*) \preceq Char((T^0,r^0,\sigma^0),X_u)$.

Because $FSC(u)$ is a full set of characteristic of A_u restricted to X_u, there is $C \in FSC(u)$ such that $C \preceq C_1 = Char((T^0,r^0,\sigma^0),X_u)$. In the previous subsection, we proved that the execution of Procedure $ForgetNode$ on C_1 computes $C_1^* = Char((T^0,r^0,\sigma^0),X_v)$. We prove that the execution of Procedure $ForgetNode$ on C computes $C^* \preceq C_1^*$.

Let $C_1 = ((T_1,r_1,\sigma_1),\ell_1,K_1,dist_1,out_1,branch_1)$ and $C = ((T,r,\sigma),\ell,K,dist,out,branch)$.

By considering the process of the execution of Procedure $ForgetNode$, that we have detailed in the previous section, (T^*,r^*,σ^*) and (T_1^*,r_1^*,σ_1^*) are isomorphic.

The only labels that are modified (when C is transformed in C^*, and C_1 is transformed in C_1^*) are the variable out of v_{att}, and possibly (last case considered in the previous subsection) the paths $Contr(P(w_1,v_{att}))$ and $Contr(P(v_{att},w_2))$ in C (resp., $Contr(P_1(w_1,v_{att}))$ and $Contr(P_1(v_{att},w_2))$ in C_1) are contracted into $Contr(Contr(P(w_1,v_{att}))\odot Contr(P(v_{att},w_2)))$ (resp., $Contr(Contr(P_1(w_1,v_{att})) \odot Contr(P_1(v_{att},w_2))))$. Note that, because (T^*,r^*,σ^*) and (T_1^*,r_1^*,σ_1^*) are isomorphic, if this latter transformation is executed, it is executed in both C and C_1.

Because $C \preceq C_1$, we get that $out^*(v_{att}) = \max\{out(v_{att}),height(w)\} \leq out_1^*(v_{att}) = \max\{out_1(v_{att}),height_1(w)\}$. Moreover, $Contr(P(w_1,v_{att})) \preceq Contr(P_1(w_1,v_{att}))$ and $Contr(P(v_{att},w_2)) \preceq Contr(P_1(v_{att},w_2))$, therefore $Contr(Contr(P(w_1,v_{att}))\odot Contr(P(v_{att},w_2))) \preceq Contr(Contr(P_1(w_1,v_{att}))\odot Contr(P_1(v_{att},w_2)))$. Then, $C^* \preceq C_1^*$, which concludes the proof.

E Proof of Lemma 9: Join node

Since $A_v = A_u \cup A_w$, A_v admits a q-branched partitioning-tree with Φ-width at most k only if A_u and A_w do. Therefore, we can assume that $FSC(u) \neq \emptyset$ and $FSC(w) \neq \emptyset$, otherwise, A_v does not admit a q-branched partitioning-tree with Φ-width at most k, and $FSC(v) = \emptyset$. The proof is twofold. We first prove that the set $FSC(v)$ returned by Procedure $JoinNode$

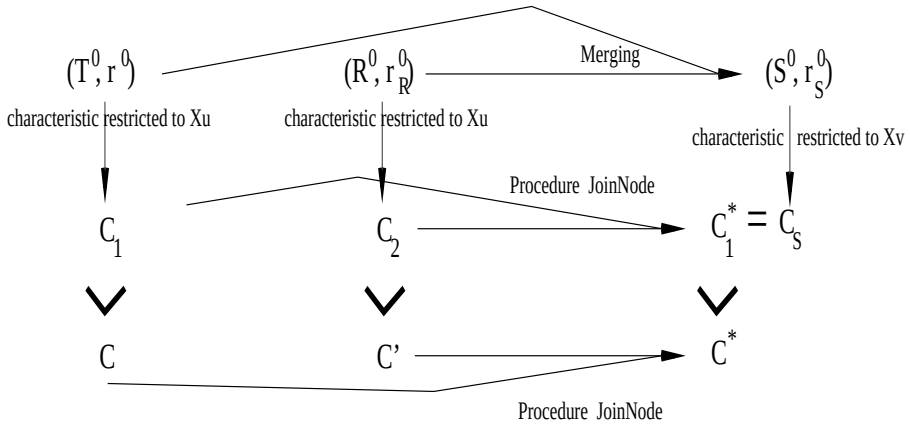

Figure 8: Proof of correctness of Procedure $JoinNode$

is a set of characteristics of A_v restricted to X_v, then we prove it is full. The proof is illustrated in Figure 8.

E.1 $FSC(v)$ is a set of characteristics of A_v restricted to X_v

Let $C_1 = ((T_1, r_1, \sigma_1), \ell_1, K_1, dist_1, out_1, branch_1) \in FSC(u)$, and let $C_1^* = ((T_2, r_2, \sigma_2), \ell_2, K_2, dist_2, out_2, branch_2)$ $FSC(w)$, such that $Struct(C_1)$ and $Struct(C_2)$ are isomorphic, and $dist_2 = 0$.

By definition of $FSC(u)$ and $FSC(w)$, $C_1 = Char((T^0, r^0, \sigma^0), X_u)$ and $C_2 = Char((R^0, r_R^0, \sigma_R^0), X_u)$ where (T^0, r^0, σ^0) is a q-branched partitioning-tree of A_u, and (R^0, r_R^0, σ_R^0) is a q-branched partitioning-tree of A_w

Let $C^* = ((T_1^*, r_1^*, \sigma_1^*), \ell_1^*, K_1^*, dist_1^*, out_1^*, branch_1^*) \in FSC(v)$ be the result of an execution of the Procedure $JoinNode$ on C_1 and C_2.

Our purpose here is to build a partitioning-tree (S^0, r_S^0, σ_S^0) of $A_v = A_u \cup A_w$ obtained by merging (T^0, r^0, σ^0) and (R^0, r_R^0, σ_R^0) in a particular way, such that $Char((S^0, r_S^0, \sigma_S^0), X_v) = C^*$. Roughly, we want to merge (T^0, r^0, σ^0) and (R^0, r_R^0, σ_R^0) respecting the way C^1 and C^2 are merged by Procedure $JoinNode$.

- Let T_1' be the smallest subtree of T^0 mapping the elements of X_v with r_1' being the vertex of T_1' that is closest to r^0, and T_2' is the smallest subtree of R^0 mapping the elements of X_v with r_2' being the vertex of T_2' that is closest to r_R^0 (note that, by definition of a characteristic, and because $dist_2 = 0$, $r_2' = r_R^0$).

  By definition of a characteristic, K_1 is the set of vertices that are a leaf, the parent of a leaf in (T_1', r_1'), a branching node of T_1', or a branching node of $cp(T^0)$ in T_1'. Moreover, $Struct(C_1)$ is the tree T_1'' obtained by contracting all vertices of degree 2 in T_1'. By definition, $V(Struct(C_1))$ is a subset of K_1. Similarly, K_2 is the set of vertices that are a leaf, the parent of a leaf in (T_2', r_2'), a branching node of T_2', or a branching

node of $cp(R^0)$ in T_2'. Moreover, $Struct(C_2)$ is the tree T_2'' obtained by contracting all vertices of degree 2 in T_2', and $V(Struct(C_2))$ is a subset of K_2.

Since $Struct(C_1)$ and $Struct(C_2)$ are isomorphic, T_1'' and T_2'' are isomorphic, and the operation of merging (T^0, r^0, σ^0) and (R^0, r_R^0, σ_R^0) is well defined.

Step 1 of Procedure *JoinNode* first identify the vertices of $V(Struct(C_1))$ in C_1 and $V(Struct(C_2))$ in C_2 and set r_1^* to r_1. Let G be the resulting graph.

The merging of (T^0, r^0, σ^0) and (R^0, r_R^0, σ_R^0) starts by identifying the vertices of $V(T_1'') = V(Struct(C_1))$ in T^0 and $V(T_2'') = V(Struct(C_2))$ in R^0 and set r_S^0 to r^0. Let H be the resulting graph. For any $x_1 \in V(Struct(C_1))$ and $x_2 \in V(Struct(C_2))$ that are identified, let x^* be the result given by identifying x_1 and x_2.

- Let $x^*, y^* \in V(G)$ such that $\{x_1, y_1\} \in E(Struct(C_1))$ (hence, $\{x_2, y_2\} \in E(Struct(C_2))$ by isomorphism). In G, there are two paths between x^* and y^* that are vertex-disjoint (but in their ends): $P_1(x^*, y^*)$ that results from the path $P_1(x_1, y_1) \in T_1$, and $P_2(x^*, y^*)$ that results from the path $P_2(x_2, y_2) \in T_2$.

  Because $\{x_1, y_1\} \in E(Struct(C_1))$, any internal vertex of $P_1(x_1, y_1)$ has degree 2 in T_1, which implies that any internal vertex or edge in $P_1(x_1, y_1)$ defines the same partition $\mathcal{P}$ of X_v. Similarly, any internal vertex or edge in $P_2(x_2, y_2)$ defines the partition $\mathcal{P}$. Step 2 of Procedure *JoinNode* merges $P_1(x^*, y^*) = P_1(x_1, y_1)$ and $P_2(x^*, y^*) = P_2(x_2, y_2)$ using the strictly increasing (for both coordinates) function $H_{\Phi,\mathcal{P}} : x, y \to H_\Phi(x, y, \mathcal{P})$. Let $P(x^*, y^*)$ be the resulting path.

  By definition of a characteristic, the path $P_1(x_1, y_1) \in T_1$ represents a path $P_1^0(x_1, y_1) \in T^0$, the internal vertices of which have degree 2 in T_1'. Note that some internal vertices of $P_1^0(x_1, y_1)$ may be branching nodes in $cp(T^0)$. We set $P_1^0(x_1, y_1) = Q_1^0 \odot Q_p^0$ such that the common ends between Q_i^0 and Q_{i+1}^0 are the branching nodes of $cp(T^0)$ in $P_1^0(x_1, y_1)$. By definition of a characteristic, $P_1(x_1, y_1) = Contr(Q_1^0) \odot Contr(Q_p^0)$. Similarly, we define the path $P_2^0(x_2, y_2) \in R^0$, and set $P_2^0(x_2, y_2) = Q_1'^0 \odot Q_{p'}'^0$, where $P_2(x_2, y_2) = Contr(Q_1'^0) \odot Contr(Q_{p'}'^0)$.

  We choose a merging of $P_1^0(x_1, y_1)$ and $P_2^0(x_2, y_2)$ using $H_{\Phi,\mathcal{P}}$ that respects the merging of $P_1(x^*, y^*) = P_1(x_1, y_1)$ and $P_2(x^*, y^*) = P_2(x_2, y_2)$.

- It is easy to see that after the merging of all paths $P_1^0(x_1, y_1)$ and $P_2^0(x_2, y_2)$, for any x^*, y^*such that $\{x_1, y_1\} \in E(Struct(C_1))$, and $\{x_2, y_2\} \in E(Struct(C_2))$ by isomorphism, then H is a tree, the leaves of which maps $A_u \cup A_w$.

  We prove it is a partitioning-tree (S^0, r_S^0, σ_S^0) of $A_v = A_u \cup A_w$. Especially, we prove that the labels of its edges and vertices are the Φ-width of the partitions they define.

  Any vertex t^* of the resulting path merges a vertex or an edge t of $P_1^0(x_1, y_1)$, that defines a partition $\mathcal{P}_t$ of A_u in T^0, and a vertex or an edge t' of $P_2^0(x_2, y_2)$, that defines a partition $\mathcal{P}_{t'}$ of A_w in R^0. Moreover, t^* defines a partition $\mathcal{P}_{t^*}$ of $A_v = A_u \cup A_w$ in S^0. The key point is that: $\mathcal{P}_t = \mathcal{P}_{t^*} \cap A_u$, $\mathcal{P}_{t'} = \mathcal{P}_{t^*} \cap A_w$ and $\mathcal{P} = \mathcal{P}_{t^*} \cap X_v$. By definition of

H_Φ, it results that t^* is labeled with $\Phi_{A_v}(\mathcal{P}_{t^*}) = H_\Phi(\Phi_{A_u}(\mathcal{P}_t), \Phi_{A_w}(\mathcal{P}_{t'}), \mathcal{P})$. Similarly, we prove that the labels of the edges in (S^0, r_S^0, σ_S^0) are those wanted.

It remains to show that $C_1^* = C_S$.

- K_S is the set of the leaves in T_S (which are exactly the leaves of T_1', i.e., the leaves of T_2'), parents of leaves, branching nodes of T_S, or branching nodes of $cp(S^0)$ in T_S. By definition of the merging of two trees, a vertex x^* belongs to K_S if it merges x_1 in T^0 and $x_2 \in R^0$ where x_1 or x_2 is a leaf in T_1' (i.e., a leaf of T_2'), the parents of leaves in T_1' or T_2', a branching node in T_1' or T_2', or a branching node of $cp(T^0)$ in T_1' or of $cp(R^0)$ in T_2'. That is K_S is the set of vertices that match x_1 in T^0 and $x_2 \in R^0$, where $x_1 \in K_1$ or $x_2 \in K_2$.

  By Step 3 of Procedure *JoinNode*, $K_S = K_1^*$.

- For any x^*, y^* such that we have merged $P_1^0(x_1, y_1)$ and $P_2^0(x_2, y_2)$ in a path $J(x^*, y^*)$ in S^0, respecting the merging of $P_1(x^*, y^*) = P_1(x_1, y_1)$ and $P_2(x^*, y^*) = P_2(x_2, y_2)$ in a path $J'(x^*, y^*)$ in C_1^*.

  $J(x^*, y^*) = J_1 \odot J_h$ such that the common ends between J_i and J_{i+1} are the vertices of K_S in $J(x^*, y^*)$. By definition of a characteristic, to obtain C_S, $J(x^*, y^*)$ is replaced by $Contr(J_1) \odot Contr(J_h)$.

  Let $J'(x^*, y^*) = J_1' \odot J_{h'}'$ such that the common ends between J_i' and J_{i+1}' are the vertices of K_1^* in $J'(x^*, y^*)$. Because, $J(x^*, y^*)$ has been built with respect to $J'(x^*, y^*)$ and $K_S = K_1^*$, we have that $h = h'$ and J_i is the merging between two subpaths of $P_1^0(x_1, y_1)$ and $P_2^0(x_2, y_2)$ that is realized with respect to J_i'.

  By Step 4 of Procedure *JoinNode* replaces $J'(x^*, y^*)$ by $Contr(J_1') \odot Contr(J_h')$.

  Now let us put the pieces together. In C_1^*, $P_1(x_1, y_1) = Contr(Q_1^0) \odot Contr(Q_p^0)$ is merged with $P_2(x_2, y_2) = Contr(Q_1'^0) \odot Contr(Q_{p'}'^0)$ using $H_{\Phi,\mathcal{P}}$ a function that is strictly increasing in both its coordinates, and the resulting path is $J'(x^*, y^*) = J_1' \odot J_h'$. In C_S, $P_1^0(x_1, y_1) = Q_1^0 \odot Q_p^0$ is merged with $P_2^0(x_2, y_2) = Q_1'^0 \odot Q_{p'}'^0$ using $H_{\Phi,\mathcal{P}}$ with respect to the merging between $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$, and the resulting path is $J(x^*, y^*) = J_1 \odot J_h$. By Lemma 3, $Contr(J_1') \odot Contr(J_h') = Contr(J_1) \odot Contr(J_h)$.

  Therefore, $((T_1^*, r_1^*, \sigma_1^*), \ell_1^*) = ((T_S, r_S, \sigma_S), \ell_S)$.

- It is easy to conclude by proving that $dist_1^* = dist_S = dist_1$, and that $branch_1^*(t) = branch_S(t)$ and $out_1^*(t) = out_S(t)$ for any $t \in K_S = K_1^*$ (because of Step 3 of Procedure *JoinNode*).

E.2 $FSC(v)$ is full

Let (S^0, r_S^0, σ_S^0) be a q-branched partitioning-tree of $A_v = A_u \cup A_w$ with Φ-width at most k. We need to prove that there is a characteristic $C^* \in FSC(v)$ such that $C^* \preceq Char((S^0, r_S^0, \sigma_S^0), X_v)$.

Let T^0 be the subtree of S^0 the leaves of which map A_u and r^0 be the vertex of T^0 that is closest to the root r_S^0. Let R^0 be the subtree of S^0 the leaves of which map A_w, and r_R^0 be the vertex of R^0 that is closest to the root r_S^0.

Note that either the distance between r^0 and r_S^0 is null, or the distance between r_R^0 and r_S^0 equals 0. W.l.o.g., we choose $r_R^0 = r_S^0$. Moreover, the intersection between T^0 and R^0 is S' the subtree of S^0 the leaves of which map $X_v = A_u \cap A_w$. Therefore, (S^0, r_S^0, σ_S^0) is a possible merging of (T^0, r^0, σ^0) and (R^0, r_R^0, σ_R^0) in X_v.

Because $FSC(u)$ is a full set of characteristics, there is $C \in FSC(u)$ with $C \preceq C_1 = Char((T^0, r^0, \sigma^0), X_u)$. Similarly, there is $C' \in FSC(w)$ with $C' \preceq C_2 = Char((R^0, r_R^0, \sigma_R^0), X_w)$.

In the previous subsection, we proved that there is an execution of Procedure $JoinNode$ on C_1 and C_2 that computes $C_1^* = Char((S^0, r_S^0, \sigma_S^0), X_v)$. We prove that the corresponding execution of Procedure $JoinNode$ on C and C' (i.e., an execution of Procedure $JoinNode$ that respects the matchings between C_1 and C_2) computes $C^* \preceq C_1^*$.

Then, the result essentially follows Lemma 3.

F Partition functions and Width parameters

In this section, we first derive several width-parameters in terms of partition functions. Then, we prove that these partition functions satisfy Lemmata 1 and 2. Together with Theorem 1, this allow to prove Theorem 2.

F.1 The considered parameters

Recall that in Section 3, we already define the treewidth (resp., pathwidth and q-branched treewidth) in terms of the partition function δ that associates to any partition $\mathcal{X} = \{E_1, \cdots, E_r\}$ of E, the number of the vertices of G that are incident to edges in E_i and E_j, with $i \neq j$.

The partition function $max\delta$-width is the function that assigns $\max_{i \leq n} \delta(E_i, E \setminus E_i)$ to any partition $(E_1, \cdots, E_n)$ of E. The branchwidth and the linear-width of a graph may be expressed in terms of this partition function:

- **branchwidth [BT97]:** By definition, the branchwidth of G, denoted by $bw(G)$, is at most $k \geq 1$ if and only if there is a partitioning-tree (T, σ) of E with $max\delta$-width at most k and such that the internal vertices of T have degree three.

- **linearwidth [BT04]:** The linear-width of G, denoted by $lw(G)$ is defined as the smallest integer k such that E can be arranged in a linear ordering $(e_1, \cdots, e_m)$ such that for every $i = 1, \cdots, m-1$ there are at most k vertices both incident to an edge that belongs to $\{e_1, \cdots, e_i\}$ and to an edge in $\{e_{i+1}, \cdots, e_m\}$. The linearwidth of G is at most $k \geq 2$ if and only if there is a partitioning-tree (T, σ) of E with $max\delta$-width at most k, such that the internal vertices of T have degree three, and (T, σ) is 0-branched. This result easily follows from the trivial correspondence between such a partitioning-tree of E and an ordering of E.

Let $Edge\delta$ be the function that assigns, for any partition $\mathcal{X} = \{V_1, \cdots, V_r\}$ of V, the cardinal of the set of the edges of G that are incident to vertices in V_i and V_j, with $i \neq j$. The partition function $maxEdge\delta$-width is the function that assigns $\max_{i \leq n} Edge\delta(V_i, V \setminus V_i)$ to any partition $(V_1, \cdots, V_n)$ of V. The partition function 3-$maxEdge\delta$-width is the function that assigns $\max\{Edge\delta(V_1, V \setminus V_1), Edge\delta(V_2, V \setminus V_2)\}$ to any partition (V_1, V_2, V_3) of V, with $|V_3| = 1$. The carvingwidth and the cutwidth of a graph may be expressed in terms of these partition functions:

- **carvingwidth [ST94, TSB00]:** By definition, the carwingwidth of G, $carw(G)$, is at most $k \geq 1$ if and only if there is a partitioning-tree of V with $maxEdge\delta$-width at most k, and such that the internal vertices of T have degree three.
- **cutwidth [TSB00]:** The cutwidth of G, denoted by $cw(G)$, is defined as the smallest integer k such that V can be arranged in a linear ordering $(v_1, \cdots, v_n)$ such that for every $i = 1, \cdots, n-1$ there are at most k edges both incident to a vertex that belongs to $\{v_1, \cdots, v_i\}$ and to a vertex in $\{v_{i+1}, \cdots, v_n\}$. The cutwidth of G is at most $k \geq 1$ if and only if there is a partitioning-tree (T, σ) of V with 3-$maxEdge\delta$-width at most k, and (T, σ) is 0-branched. This result easily follows from the trivial correspondence between such a partitioning-tree of V and an ordering of V.

F.2 Compatibility of the partition functions

In this section, we present some ideas in order to show the compatibility of the above mentioned partition functions with the nice tree decomposition.

F.2.1 Partition functions $Edge\delta$ and $maxEdge\delta$

It is easy to see that the partition function $Edge\delta$ behaves as the δ function but the role of vertices and edges being reversed. In the following, $Edge\Delta$ is the function that assigns, for any partition $\mathcal{X} = \{V_1, \cdots, V_r\}$ of V, the set of the edges of G that are incident to vertices in V_i and V_j, with $i \neq j$.

First note that any nice tree-decomposition $(T, \mathcal{Y})$ of G is a nice decomposition of V. To prove that $(T, \mathcal{Y})$ is compatible with the partition function $Edge\delta$, we follow the proof of Lemma 2 in Section 3. For this purpose, the functions $F_{Edge\delta}$ and $H_{Edge\delta}$ are defined as follows.

- Let x be an integer, $\mathcal{P}$ be a partition of a subset V' of V and a vertex $v \in V'$. Then, $F_{Edge\delta}(x, \mathcal{P}, v) = x + |\{e \in E \mid v \in e \text{ and } e \in Edge\Delta(\mathcal{P}) \setminus Edge\Delta(\mathcal{P} \cap (V' \setminus \{v\}))\}|$.
- Let x and y be two integers, and let $\mathcal{P}$ be a partition of a subset V' of V. Then, $H_{Edge\delta}(x, y, \mathcal{P}) = x + y - Edge\delta(\mathcal{P})$.

Therefore, the partition function $Edge\delta$ is compatible with any nice tree-decomposition. To prove the compatibilty of the partition function $maxEdge\delta$ with any nice tree-decomposition, we use the framework described below.

F.2.2 Partition function $maxf$ of a compatible partition function f

In this section we prove that, for any partition function f compatible with a nice decomposition of some set A, the partition function $maxf$ defined as follows is also compatible. For technicality, we however need to slightly modify Procedures $IntroduceNode$ and $JoinNode$ in this case. In particular, this section allows to prove that Theorem 2 is valid for the branchwidth, linearwidth, carwingwidth and cutwidth of a graph.

Let f be any partition function and let $maxf$ be the partition function that associates $\max_{i\leq n} f(A_i, E\setminus A_i)$ to any partition $(A_1, \cdots, A_n)$ of A. Because f is compatible with any nice decomposition of A, there exist two function F_f and H_f that satisfy the properties defining the notion of compatibility. With the notation of Section, we must have

$$f_{A_v}(\mathcal{P}) = F_f(f_{A_u}(\mathcal{P}\cap A_u), \mathcal{P}\cap X_v, A_v\setminus A_u),\ and$$

$$f_{A_v}(\mathcal{P}) = H_f(f_{A_u}(\mathcal{P}\cap A_u), f_{A_w}(\mathcal{P}\cap A_w), \mathcal{P}\cap X_v).$$

The key point is that if $\mathcal{P}$ is a bipartition of some set A, then $f_A(\mathcal{P}) = maxf_A(\mathcal{P})$. Therefore, when considering a bipartition, the functions F_{maxf} and H_{maxf} can be defined similarly to F_f and H_f.

Now, let us consider Step 3 of Procedure $IntroduceNode$. The first instruction consists in updating the weight of the edges of the characteristic of A_v. Since an edge of a characteristic of A_v corresponds to a bipartition of A_v, by the remark above, we can use the function F_f. Then, the second instruction consists in updating the weight of the internal vertices of the characteristic of A_v. It is easy to see that, for any vertex x of the characteristic, it is sufficient to set the weight of x to the maximum of its current weight (the weight it had in the characteristic of A_u) and the weight of its incident edges.

In Step 2 of Procedure $JoinNode$, we proceed similarly. When two edges of the paths P_T and P_S are merged together, they correspond to a bipartition of A_v and the function H_f can be used. When, the procedure merges a vertex with weight x of one path with an edge of the other path, the weight of the resulting vertex is set to the maximum between x and the weight of the incident edges. Finally, when the procedure merges a vertex with weight x of P_T with a vertex with weight y of P_S, the weight of the resulting vertex is set to the maximum between x, y and the weight of the incident edges.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399