Feedback Vertex Set on Graphs of Low Clique-width*

B.-M. Bui-Xuan[1] O. Suchý[2,†] J. A. Telle[1]
M. Vatshelle[1]

[1]Department of Informatics, University of Bergen, Norway
[buixuan,telle,vatshelle]@ii.uib.no

[2]Universität des Saarlandes, Saarbrücken, Germany
suchy@mmci.uni-saarland.de

Submitted: Dec 20, 2009; Accepted: Aug 8, 2011; Published: XX
Mathematics Subject Classification: 05C85, 68R10

Abstract

The Feedback Vertex Set problem asks whether a graph contains q vertices meeting all its cycles. This is not a local property, in the sense that we cannot check if q vertices meet all cycles by looking only at their neighbors. Dynamic programming algorithms for problems based on non-local properties are usually more complicated. In this paper, given a graph G of clique-width cw and a cw-expression of G, we solve the Minimum Feedback Vertex Set problem in time $O(n^2 2^{O(cw \log cw)})$. Our algorithm applies dynamic programming on a so-called k-module decomposition of a graph, as defined by Rao [30], which is easily derivable from a k-expression of the graph. The related notion of module-width of a graph is tightly linked to both clique-width and NLC-width, and in this paper we give an alternative equivalent characterization of module-width.

1 Introduction

The problem of finding a minimum Feedback Vertex Set (FVS) in a graph, i.e. the smallest set of vertices whose removal results in a graph that has no cycles, has many applications. For example to optical networks [22], circuit testing, deadlock resolution, analyzing manufacturing processes and computational biology (see [10] and its bibliography). It is one of the classical NP-complete problems from the 1972 list of Karp [21] and

*Supported by the Norwegian Research Council, project PARALGO.

†Supported by the DFG Cluster of Excellence on Multimodal Computing and Interaction, DFG project DARE (GU 1023/1-2) and by grant 1M0545 of the Czech Ministry of Education. Most of the work done while on Dep. of Applied Mathematics and Inst. for Theoretical Computer Science, Charles University, Prague, Czech Republic and while visiting the University of Bergen.

has been extensively studied from many viewpoints, including linear programming [7], approximation algorithms [2, 13, 16, 22], exact algorithms [14] and parametrized complexity [6, 10, 17, 25, 29].

The minimum FVS problem is 2-approximable in polynomial time [1]. The fastest exact algorithm has runtime $O(1.7548^n)$ [14]. The fastest FPT (Fixed Parameter Tractable) algorithm when parametrized by the size q of the FVS has runtime $O(5^q qn^2)$ [6]. These algorithmic results are quite strong, but are not useful for cases of input graphs having a large number of vertices n, and a large minimum FVS q, if we want the actual smallest FVS. For such cases we may instead hope that the input graph has a bounded width parameter. For example, if G is a planar graph of treewidth tw then Kloks et al [23] give a dynamic programming algorithm solving minimum FVS on G in time $O(2^{O(tw \log tw)}n)$. Recently a randomized algorithm with runtime $O(3^{tw}n^{O(1)})$ for minimum FVS with one-sided error of probability at most 0.5 was developed [9]. It is still an open problem whether there is a deterministic $O(2^{O(tw)}n^{O(1)})$ algorithm for FVS, even though such algorithms exist for a large variety of NP-hard problems. However, for minimum FVS it would require a small breakthrough to get such an algorithm. One reason for this is that FVS is not a locally checkable property, in the sense that given q vertices we cannot check that they form an FVS simply by looking at their neighbors. One also has to consider paths between pairs of vertices. The same issue arises when the problem is parametrized by q the size of the FVS, but Dehne et al. [10] gave an $O(2^{O(q)}n)$ algorithm using the technique of iterative compression and the running time for this parameterization was improved in a long series of papers. In this paper we do not aim to prolong this series. Instead, we consider the problem on graphs of clique-width cw. This graph class encompasses large classes of graphs of unbounded treewidth, and for which powerful algorithmic results are known. Note that bounded clique-width does not imply bounded treewidth, hence we can not directly translate FPT algorithms parametrized by treewidth to FPT algorithms parametrized by clique-width. For instance, we have that any graph problem expressible in MSO_1-logic, as is the case with minimum FVS, is FPT when parametrized by clique-width (roughly, apply [20], then [27, Proposition 6.3], then [8]). Since FVS can be expressed in MSO_1-logic it follows that FVS is FPT parameterized by clique-width, however the running time will contain a tower of 2's, and we are not aware of any MSO_1 formulation which would lead to a tower with less than 4 levels. In this paper we are interested in as low exponential dependency on cw as possible, and for this we need to use a specially designed dynamic programming algorithm.

The complement of a FVS is a vertex subset inducing a forest, and solving such tree-like problems using dynamic programming based on clique-width are usually more complicated than dynamic programming based on treewidth. An $O^*(2^{cw^2 \log cw})$ algorithm was given in [5], and an $O^*(2^{cw^2})$ algorithm was given in [15] (this algorithm also works for the lower parameter rank-width). The first algorithm is an extension of the treewidth algorithm with the key observation that we only need to consider cw^2 components of the complement of FVS. The second improves this by cleverly reducing the number of considered components to cw.

In this paper we give an $O^*(2^{cw \log cw})$ algorithm. To this end, we use the trick of

considering only *cw* components, but we also make use of a technique taking into account an "expectation from the outside" during the bottom-up computation of the dynamic programming. This technique was introduced in [3] where it was used for finding a minimum dominating set along a so-called H-join decomposition, and later also for FVS in [15]. Roughly, when operating on some reduced instance – *e.g.*, some subgraph $G[A]$ induced by vertex subset A – we not only compute solutions – *e.g.*, FVS, dominating sets, etc. – depending on $G[A]$, but also those solutions satisfying specific constraints depending on $G[V(G) \setminus A]$. As opposed to classical dynamic programming, a consequence of "expecting from the outside" will be that we no longer partition the set of possible solutions (of $G[A]$) into equivalence classes with a one-to-one correspondence between such classes and indices of the table (data-structure for dynamic programming). Instead, each possible solution now can influence several indices of the table.

The exponential dependency on clique-width of our algorithm matches asymptotically the current best known algorithms based on treewidth. More precisely, our algorithm finds a minimum FVS on a graph G of clique-width cw in time $O(2^{O(cw \log cw)} n^2)$, when given a cw-expression of G which is a decomposition of the graph showing that it has clique-width cw.

Clique-width is related to the notion of NLC-width of a graph [12] with which it shares most properties but we have chosen to use clique-width in this paper simply because that notion is more well known. More specifically, our algorithm applies dynamic programming on a so-called k-module decomposition of a graph, as defined by Rao [30], which is easily derivable from a k-expression of the graph. The related notion of module-width of a graph is tightly linked to both clique-width and NLC-width, and in this paper we give an alternative equivalent characterization of module-width.

2 Framework

Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. Consider the following unifying decomposition framework for several decomposition schemes. A binary tree is a rooted tree where every internal node has exactly two children.

Definition 2.1 (Decomposition tree)**.** A *rooted decomposition tree* of a graph G is a pair (T, δ) where T is a binary tree having $n = |V(G)|$ leaves and δ is a bijection between the vertices of G and the leaves of T.

Roughly, trees with their leaves in a bijection with the vertices of G are important for techniques like divide-and-conquer or dynamic programming since they show how to "divide" the graph instance into several sub-instances and recurse. Clearly, any tree with the right number of leaves and a bijection can be considered as a decomposition tree. Then, a common technique to select those that are more suited for some task is to use an evaluating function.

Definition 2.2 (Decomposition and width parameters)**.** Let G be a graph, $f : 2^{V(G)} \to \mathbb{R}$ a function assigning a non-negative real value to subsets of $V(G)$, and (T, δ) a rooted

decomposition tree of G. For every node u of T, let V_u denote the vertex subset of G induced by the leaves of the subtree of T rooted at u. The *f-width of (T, δ)* is the maximum value of $f(V_u)$, taken over every node u of T. An *optimal f-decomposition* of G is a rooted decomposition tree of G having minimum f-width. The *f-width of G* is the f-width of an optimal f-decomposition of G.

If f is also required to be symmetric, namely that $f(V_u) = f(V(G) \setminus V_u)$ for every V_u, then the above framework, up to unrooting the tree T and setting $f(V(G)) = f(\emptyset) = 0$, is equivalent to the one developed for the study of branch decomposition of symmetric and submodular functions (see, *e.g.*, [27, Section 2] for a short and recent introduction). This includes the branch-width [31], rank-width [27], and boolean-width [4] decompositions of graphs. On the other hand, rooted decomposition trees as defined here can be used for situations where the symmetry does not occur, for instance with a branch-like decomposition of a submodular function that is not necessarily symmetric, a clique-width or NLC-width expression, or a so-called k-module decomposition as will be presented below.

For an efficient complexity analysis of the algorithm that will be described in Section 4, we will be interested in the following definition of f-width, so-called module-width in [26, 32].

Definition 2.3. Let G be a graph and let $X \subseteq V(G)$ be a vertex subset. A subset $A \subseteq X$ is a *twin set of X* if, for every $z \in V(G) \setminus X$ and pair of vertices $x, y \in A$, we have x adjacent to z if and only if y adjacent to z. A twin set A is a *twin-class of X* if A is maximal. The set of all twin-classes of X forms a partition of X, that we call the *twin-class partition of X*.

Definition 2.4 (Module-width)**.** The function $\mu_G : 2^{V(G)} \to \mathbb{N}$ is defined such that $\mu_G(X)$ is the number of twin-classes of X in the graph G. The *module-width decompositions and parameters* of G refer to those of Definition 2.2 when $f = \mu_G$. The μ_G-width of G will be called the *module-width* of G and denoted by $\mu w(G)$.

The above terminology of module-width is according to the name given to an equivalent notion that was mentioned in [26, last two pages] and formalized in [32, Section 6.1.2]. Indeed, one can use a similar decomposition framework, so-called k-module decomposition, in order to result in the same parameter as follows.

Definition 2.5. ([26, 32]) Let G be a graph. A vertex subset $X \subseteq V(G)$ is a *k-module* if there exists a partition of X into k twin sets. G is a *k-module decomposable graph* if there is a rooted decomposition tree (T, δ) such that every vertex subset of G that is induced by the leaves of some subtree of T is also a k-module of G. The *module-width* of G is the minimum integer k such that G is k-module decomposable.

Definitions 2.4 and 2.5 both lead to the same notion of module-width thanks to the following simple observations. Firstly, if X is a k-module, then it is also a $(k+1)$-module as long as $k + 1 \leq |X|$. Secondly, the minimum number k such that X is a k-module is exactly $\mu_G(X)$.

Clique-width and NLC-width expressions are constructions of a graph using logic operations. For a proper introduction to clique-width and NLC-width refer to [8, 12]. The underlying graphs of clique-width and NLC-width expressions are rooted trees where every internal node has at most two children and where the leaves are in a bijection with the vertices of the graph. This, up to contracting one child nodes, can be seen as a rooted decomposition tree. The clique-width $cw(G)$ and the NLC-width $nlc\text{-}w(G)$ of a graph G are parameters of G having powerful algorithmic properties. For instance, we have that any graph problem expressible in MSO_1-logic is FPT when parametrized by one of these two parameters (roughly, apply [20], then [27, Proposition 6.3], then [8]). They are closely linked to module-width by the following property.

Theorem 2.6. *([32, Theorem 6.6]) We have for any graph G that*

$$\mu w(G) \leq nlc\text{-}w(G) \leq cw(G) \leq 2\mu w(G).$$

We now give an alternative viewpoint of these module-width decompositions, that will link module-width to the so-called H-join decomposition framework [3] in an unexpected way.

Definition 2.7. Let H be a bipartite graph with color classes V_1 and V_2, thus $V(H) = V_1 \cup V_2$. Let G be a graph and $X \subseteq V(G)$ a subset of its vertices. We say that G is an *H-join across the ordered cut* $(X, V(G) \setminus X)$ if there exists a partition of X with set of classes $\mathcal{P}$ and a partition of $V(G) \setminus X$ with set of classes $\mathcal{Q}$, and injective functions $f_1 : \mathcal{P} \to V_1$ and $f_2 : \mathcal{Q} \to V_2$, such that for any $x \in X$ and $y \in V(G) \setminus X$ we have x adjacent to y in G if and only if x belongs to a class P_i of $\mathcal{P}$ and y to a class Q_j of $\mathcal{Q}$ with $f_1(P_i)$ adjacent to $f_2(Q_j)$ in H.

We will abusively refer to ordered cuts simply by cuts. Twins in a bipartite graph are vertices in the same color class having exactly the same neighborhood. A *twin contraction* is the deletion of a vertex when it has a twin. Notice that H-joins are insensitive to twin contractions: if H' is obtained from H by a twin contraction then G is an H-join across some cut if and only if G is an H'-join across the same cut. Note also that we do allow a twin-free bipartite graph to have one isolated vertex in each color class. We model the joining in module-width decompositions by using the following graph.

Definition 2.8. For a positive integer k we define a bipartite graph Y_k having for each integer i of $\{1, 2, \ldots, k\}$ a vertex $a_i \in A$ and having for each subset S of $\{1, 2, \ldots, k\}$ a vertex $b_S \in B$, with $V(Y_k) = A \cup B$. This gives k vertices in A and 2^k vertices in B. A vertex a_i is adjacent to a vertex b_S if and only if $i \in S$.

Lemma 2.9. *Let k be an integer, let H be a bipartite graph over color classes $V_1 \cup V_2$ with $|V_1| \leq k$. Then, applying successive twin contractions in H until stability will always result in a graph that is isomorphic to an induced subgraph of Y_k.*

Proof. Just give an arbitrary ordering over the vertices of $V_1 = (v_1, v_2, \ldots, v_l)$, and map them to the l first vertices $a_1, a_2, \ldots, a_l$ of Y_k, respectively (note that $l \leq k$ by hypothesis).

Then, for every vertex $u \in V_2$ of H, let $S = \{i : v_i \in N(u)\}$, and map u to vertex b_S of Y_k. Hence, H is an induced subgraph of Y_k. Now, applying twin contractions on a subgraph of Y_k will always result in another induced subgraph of Y_k. □

Corollary 2.10. *The function μ_G of Definition 2.4 is exactly equal to the function η_G defined for all $X \subseteq V(G)$ by*

$$\eta_G(X) = \min\{k : G \text{ is a } Y_k\text{-join across the cut } (X, V(G) \setminus X)\}.$$

Proof. In Definition 2.7 of an H-join across $(X, V(G) \setminus X)$, if we consider as joining partition of X the twin-class partition of X, then H is a bipartite graph having exactly $\mu_G(X)$ vertices on one of its color class. Lemma 2.9 then allows to conclude. □

3 Computing the twin-classes

In the next section we will give a dynamic programming algorithm to solve the feedback vertex set problem on an input made by an n-vertex m-edge graph G and one of its rooted decomposition tree (T, δ). Note that the underlying graph of a clique-width expression of G is a rooted tree where each internal node has at most two children, and the leaves are in a bijection with the vertices of G. Contracting the internal nodes having one child will result in a rooted decomposition tree of G. Moreover, it can also be obtained from the proof of Theorem 2.6 that the module-width of this rooted decomposition tree is at most the clique-width of the clique-width expression. Consequently, if the input to our algorithm is the graph G and a clique-width k expression of G, we can transform them in a straightforward manner to an input made of G and one of its rooted decomposition tree of module-width at most the value of k.

For every internal node u of T with V_u being the vertex subset of G induced by the subtree of T rooted at u, we will need to compute the twin-classes of V_u as mentioned in the definition of μ_G in Definition 2.4. In this section, we will describe how to perform such a computation for every internal node u of T, in global running time $O(n^2)$.

We will use the so-called partition refinement algorithmic technique (refer to, *e.g.*, [18, 28] for details). Partitions will be represented by double-linked lists. A *refinement operation* of a partition $\mathcal{Q} = (Q_1, Q_2, \ldots, Q_k)$ of V_u using $A \subseteq V_u$ as pivot is the act of splitting every Q_i into $Q_i \cap A$ and $Q_i \setminus A$. The output of a refinement operation can be of two types. It can be made of one partition of V_u which is the result of removing all empty sets from $(Q_1 \cap A, Q_1 \setminus A, Q_2 \cap A, Q_2 \setminus A, \ldots, Q_k \cap A, Q_k \setminus A)$. We refer to these as one-to-one refinements. It can also be composed of two partitions (one of A and one of $V_u \setminus A$) which result from removing all empty sets from $(Q_1 \cap A, Q_2 \cap A, \ldots, Q_k \cap A)$ and $(Q_1 \setminus A, Q_2 \setminus A, \ldots, Q_k \setminus A)$. We refer to these as one-to-two refinements. With the appropriate data structure, all these types of refinement operations can be implemented to run in $O(|A|)$ time for each operation (refer to, *e.g.*, [18] for details).

A simple way to compute the twin-class partition of V_u is to initialize $\mathcal{Q} = (V_u)$ and, for every vertex $z \in V(G) \setminus V_u$, perform an one-to-one refinement of $\mathcal{Q}$ using the neighborhood $N(z)$ of z as pivot. The correctness follows directly from the definition of twin-classes.

This computation would have $O(m)$ runtime for each internal node u of T, hence a global $O(nm)$ runtime.

The main idea to reduce this runtime is to observe that, in the above operations, we can use $N(z) \cap V_u$ as pivot instead of $N(z)$ (for every $z \in V(G) \setminus V_u$) without modifying the refined partition of each step. However, the sum over every possible V_u and $z \in V(G) \setminus V_u$ of the value $|N(z) \cap V_u|$ might still be large. We will observe a second fact. For a partition $\mathcal{Q} = (Q_1, Q_2, \ldots, Q_k)$ of X and a subset $Y \subseteq X$, we denote by $\mathcal{Q}[Y]$ the partition of Y which results from removing all empty sets from $(Q_1 \cap Y, Q_2 \cap Y, \ldots, Q_k \cap Y)$.

Remark 3.1. *Let w be an internal node of T with children a and b. Let V_w, V_a, and V_b be the vertex subsets of G induced by the leaves of the subtrees of T rooted at w, a, and b, respectively. Let $\mathcal{Q}_w = (Q_w(1), Q_w(2), \ldots, Q_w(h_w))$ be the twin-class partition of V_w. Then, initializing $\mathcal{Q} = \mathcal{Q}_w[V_a]$ and refining $\mathcal{Q}$ using $N(z) \cap V_a$ as pivot for all $z \in V_b$ will result to the twin-class partition of V_a.*

Basically, the algorithmic difference given by the remark is that we can now be restricted to $z \in V_b$ instead of using all $z \in V(G) \setminus V_a$ as before. The main point is that the sum over every possible V_a and $z \in V_b$ of the value $|N(z) \cap V_a|$ will be at most twice the value $n + m$ (every edge of G appears at most twice in the sum). We now implement Remark 3.1.

First of all, the bottleneck of using $N(z) \cap V_a$ as pivot will be that, unlike the case with $N(z)$ which can be read simply in the adjacency list of G, we will need to compute $N(z) \cap V_a$ for every possible V_a and z. We do this as a preprocessing step as follows.

We prepare the tree T as described in [19] so that afterwards we can, given two leaves x and y of T, compute the lowest common ancestor w of x and y in T in $O(1)$ time. This can also be done in such a way that, if a and b denote the children of w, then we can in $O(1)$ time decide whether x is a descendent of a or it is a descendent of b. Then, for every internal node w of the tree T, with children a and b, we initialize two tables $N_w^{b \to a}$ and $N_w^{a \to b}$ that will contain, for every vertex z in V_b (resp. V_a), the neighborhood of z in V_a (resp. V_b). Now, we scan through every edge xy of G and compute the lowest common ancestor w of x and y, as well as the children a and b of w such that x is a descendent of a, and finally add x to $N_w^{b \to a}[y]$ and y to $N_w^{a \to b}[x]$. Clearly, after scanning all edges of G, we have that $N_w^{b \to a}[z] = N(z) \cap V_a$ for all w, a, b, and z. This preprocessing takes $O(m)$ time.

We come to the proper computation of the twin-class partitions. The twin-class partition associated to the root of T only has one class, which is $V(G)$. Suppose that we have computed the twin-class partition $\mathcal{Q}_w$ of an internal node w having children a and b. This partition $\mathcal{Q}_w$ is stored in a double-linked list w.r.t. the data structure used for partition refinement. Basically, the following operations can operate directly on this data structure, if we allow ourselves to modify the double-linked list. However, the information on the twin-classes of V_w would then be lost. For this reason, before continuing, we duplicate the data structure of $\mathcal{Q}_w$ so that we store the twin-classes of V_w in a private place of node w. Then, we can compute $\mathcal{Q}_w[V_a]$ and $\mathcal{Q}_w[V_b]$ simply by performing an one-to-two refinement of $\mathcal{Q}_w$ using either V_a or V_b as pivot (cf. $V_b = V_w \setminus V_a$) for each w. Duplication

and refinement using V_a (or V_b) as pivot take $O(n)$ time for every node w, hence an $O(n^2)$ global runtime.

We then initialize $\mathcal{Q} = \mathcal{Q}_w[V_a]$ and, for every entry z of the table $N_w^{b\to a}$, refine $\mathcal{Q}$ using $N_w^{b\to a}[z]$ as pivot. As mentioned before, the main point of all these procedures is that the sum of the size of all possible pivots will now be at most twice the value $n+m$. Hence, the global runtime of this step is in $O(n+m)$. We deduce the following lemma, whose proof is straightforward. Recall that from the input of a clique-width expression of G, we can derive a rooted decomposition tree simply by contracting all internal nodes having one child in the underlying graph of the clique-width expression. The module-width of this decomposition tree is at most the clique-width of the expression.

Lemma 3.2. *Given a graph G and either (T,δ) a rooted decomposition tree of G, or a clique-width expression tree of G. Then in $O(n^2)$ global runtime we can compute and store, for every internal node u of T with V_u being the vertex subset of G induced by the leaves of the subtree of T rooted at u, the partition of V_u into its twin-classes $Q_u(1), Q_u(2), \ldots, Q_u(h_u)$.*

4 Solving the Feedback Vertex Set Problem

Definition 4.1. A *feedback vertex set* of a graph G is a subset S of the vertices of G with $G[V(G)\setminus S]$ a forest. A *forest inducing set (FI-set)* of a graph G is a subset of vertices S with $G[S]$ a forest.

Fact 4.2. *If S is a FI-set of maximum cardinality then $V(G)-S$ is a feedback vertex set of minimum cardinality.*

We give dynamic programming algorithms that given a graph G and a rooted decomposition tree (T,δ) of G will find the size of a minimum Feedback Vertex Set of G, by computing the size of a maximum FI-set in G. Recall that in Section 3 V_a, V_b, V_w were the vertex subsets corresponding to subtrees rooted at nodes a, b, w of T. For simplicity we adopt $A = V_a, B = V_b, W = V_w$ for the rest of this section. The runtime of the algorithm will be expressed as a function of $\mu_G(A)$, i.e. the number of twin-classes of such vertex subsets $A = V_a$.

4.1 Definition of Tables

For $A \subseteq V(G)$ let $\mathcal{TC}_A = \{TC_A^1, TC_A^2, ...TC_A^k\}$ be the twin-classes of A, using $k = \mu w(V(G))$ and allowing some empty classes. The indices of table Tab_A will consist of pairs $(\mathcal{P},\mathcal{C})$ where $\mathcal{P}$ is a partition of $\mathcal{TC}_A$ and $\mathcal{C}$ is a partition of a subset of $\mathcal{TC}_A$. We denote the classes of $\mathcal{P}$ by $Q_A^0, Q_A^1, R_A^0, R_A^1, ..., R_A^k$, allowing some empty classes. $\mathcal{C}$ will be a partition of $\mathcal{TC}_A \setminus (Q_A^0 \cup R_A^0)$ and a coarsening of $R_A^1, R_A^2, ..., R_A^k$ in the sense that two twin-classes both in R_A^i, for some $1 \le i \le k$, must belong to the same class in $\mathcal{C}$.

Before defining the contents of the table formally, let us briefly give some intuition. An index $(\mathcal{P},\mathcal{C})$ will store a largest FI-set $S \subseteq A$ satisfying certain properties, e.g. where

Q^0_A and Q^1_A are those twin-classes containing exactly zero and one vertex from S and the remaining classes of $\mathcal{P}$ consist of twin-classes containing at least two vertices of S. Among these, the twin-classes of R^0_A are exactly those containing vertices that should not get *any* further FI-set neighbors as we progress up to the root of the decomposition tree. The partition of the remaining twin-classes into $R^1_A, R^2_A, ..., R^k_A$ is part of the 'expectation from the outside'. As we progress up the path to the root two nodes receive a new common FI-neighbor if and only if they are in twin-classes belonging to the same R^i_A. The partition class $\mathcal{C}$ is also part of the expectation and tells us about connected components of FI-sets.

Definition 4.3. For every partition $\mathcal{P}$ of $\mathcal{TC}_A$ into $k+3$ parts $Q^0_A, Q^1_A, R^0_A, R^1_A, ..., R^k_A$ (allowing empty classes) and every partition $\mathcal{C}$ of $\mathcal{TC}_A \setminus (Q^0_A \cup R^0_A)$ that is a coarsening of $R^1_A, R^2_A, ..., R^k_A$, we have an index $(\mathcal{P}, \mathcal{C})$ in Tab_A. The contents of $\mathsf{Tab}_A[\mathcal{P}, \mathcal{C}]$ will be a vertex subset $S \subseteq A$ of maximum cardinality among all $S \subseteq A$ satisfying the pair $(\mathcal{P}, \mathcal{C})$, where such S is said to *satisfy* $(\mathcal{P}, \mathcal{C})$ if

1. $Q^0_A = \{TC^i_A : |TC^i_A \cap S| = 0\}$
2. $Q^1_A = \{TC^i_A : |TC^i_A \cap S| = 1\}$
3. The graph $G(\mathcal{P}, S)$ is a forest, where $G(\mathcal{P}, S)$ is constructed from $G[S]$ by adding new intermediate vertices $\{v_1, v_2, ..., v_k\}$, and edges from v_i to all vertices in twin-classes belonging to R^i_A, for $1 \leq i \leq k$.
4. Two vertices $u \in TC^i_A \cap S$ and $v \in TC^j_A \cap S$ with TC^i_A, TC^j_A not in R^0_A belong to the same connected component of $G(\mathcal{P}, S)$ if and only if TC^i_A, TC^j_A are in the same class of $\mathcal{C}$.

If no set $S \subseteq A$ satisfying $(\mathcal{P}, \mathcal{C})$ exists then the contents of $\mathsf{Tab}_A[\mathcal{P}, \mathcal{C}]$ should be $\sharp$.

4.2 The dynamic programming algorithm

We are now ready to describe the algorithm computing a maximum FI-set of G. The algorithm starts by initializing all table entries of all tables of the tree T to $\sharp$.

At any leaf a of the tree T we have $A = \{\delta(a)\}$ and $\mathcal{TC}_A = \{\{\delta(a)\}\}$. Two entries of the table Tab_a will be updated to something other than $\sharp$ corresponding to the two choices for a set satisfying an index $(\mathcal{P}, \mathcal{C})$, namely $S = \{\delta(a)\}$ and $S = \emptyset$. The first choice gives $\mathsf{Tab}_a[\mathcal{P}, \mathcal{C}] = \{\delta(a)\}$ for $\mathcal{P}$ having empty classes except $Q^1_A = \{\{\delta(a)\}\}$ and $\mathcal{C} = \{\{\delta(a)\}\}$. The second choice gives $\mathsf{Tab}_a[\mathcal{P}, \mathcal{C}] = \emptyset$ for $\mathcal{P}$ having empty classes except $Q^0_A = \{\{\delta(a)\}\}$ and $\mathcal{C}$ the partition of the empty set.

In a bottom-up traversal of the tree T, when reaching an internal node w having children a and b we do the following dynamic programming:

For all index triples $(\mathcal{P}_A, \mathcal{C}_A), (\mathcal{P}_B, \mathcal{C}_B), (\mathcal{P}_W, \mathcal{C}_W)$
  If $\mathsf{Tab}_a[\mathcal{P}_A, \mathcal{C}_A] = S_A$ and $\mathsf{Tab}_b[\mathcal{P}_B, \mathcal{C}_B] = S_B$ (i.e. not $\sharp$ entries)
    and $S_A \cup S_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$
    and $\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W] = \sharp$ or $|\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W]| < |S_A \cup S_B|$
  Then update $\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W] := S_A \cup S_B$

After the bottom-up traversal filling all tables output the entry $\mathsf{Tab}_{root}[\mathcal{P}, \mathcal{C}]$ at the root of T, for $\mathcal{P}$ having empty classes except $R^0_{root} = \{V(G)\}$ and $\mathcal{C}$ the partition of the empty set.

5 Correctness and timing

Let us start by noting that at the root of T we have $\mathcal{TC}_{root} = \{V(G)\}$ and the value of $\mathsf{Tab}_{root}[\mathcal{P}, \mathcal{C}]$ for the partition where $R^0_{root} = \{V(G)\}$ and $\mathcal{C}$ the partition of the empty set will by Definition 4.3 be a maximum FI-set of G. A central part of the correctness argument is to show that the FI-set stored in a table entry will induce an acyclic graph. This will be done by contradiction, showing that a cycle in one graph can be replaced in another graph by a walk starting and ending in the same vertex and using some edge exactly once, and then applying the following easy observation.

Lemma 5.1. *Consider a graph H containing a walk starting and ending in the same vertex. If some edge uv appears an odd number of times in the walk then the subgraph H' of H induced by the edges in the walk contains a cycle.*

Proof. Note that H' is a connected graph. We first show that it remains connected also after removing the edge uv. Removing uv breaks the walk up in subwalks of three types, from u to u, from v to v and from u to v. Since uv is used an odd number of times and the original walk started and ended in same node, both u and v are used as endpoints an odd number of times. Therefore one of the subwalks will go from u to v showing that u and v are in the same connected component even after removing uv. Since adding a new edge to a connected graph will give a graph with a cycle the graph H' must have contained a cycle. □

Lemma 5.2. *The dynamic programming algorithm will correctly fill all tables.*

Proof. The lemma is proved by bottom-up induction on the tree T. Leaves of T are correctly updated since we try both subsets of nodes as FI-sets for the unique indices that they satisfy. Consider an internal node w of T with children a, b and assume inductively that Tab_a and Tab_b are correct. We show that Tab_w is then updated correctly. Recall that A, B, W are the vertex subsets corresponding to subtrees rooted at a, b, w.

For any index $(\mathcal{P}_W, \mathcal{C}_W)$ we must show that if there is a set satisfying $(\mathcal{P}_W, \mathcal{C}_W)$ then $\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W]$ is not equal $\sharp$ and that, if $\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W] = S_W$ then $S_W \subseteq W$ is a largest set satisfying $(\mathcal{P}_W, \mathcal{C}_W)$. First, note that in the latter case we know S_W satisfies $(\mathcal{P}_W, \mathcal{C}_W)$ as this was checked in the algorithm.

Thus, assume for contradiction that there is a set $F_W \subseteq W$ satisfying $(\mathcal{P}_W, \mathcal{C}_W)$ and that we have either $|F_W| > |\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W]|$ or we have $\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W] = \sharp$. From $\mathcal{P}_W, \mathcal{C}_W, F_W$ we first construct $(\mathcal{P}_A, \mathcal{C}_A), (\mathcal{P}_B, \mathcal{C}_B)$ such that $F_W = F_A \cup F_B$ and F_A satisfies $(\mathcal{P}_A, \mathcal{C}_A)$ and F_B satisfies $(\mathcal{P}_B, \mathcal{C}_B)$. Note that we will not be using $\mathcal{C}_W$ as the pair $\mathcal{P}_W, F_W$ uniquely defines $\mathcal{C}_W$. Let $F_A = F_W \cap A$ and $F_B = F_W \cap B$.

We now construct $\mathcal{P}_A = Q^0_A, Q^1_A, R^0_A, R^1_A, ..., R^k_A$. We set $Q^0_A = \{TC^i_A : |TC^i_A \cap F_A| = 0\}$ and $Q^1_A = \{TC^i_A : |TC^i_A \cap F_A| = 1\}$. Recall that $\mathcal{TC}_W$ is a coarsening of $\mathcal{TC}_A \cup \mathcal{TC}_B$, so

that a class of $\mathcal{TC}_W$ is the union of some classes of $\mathcal{TC}_A$ and some of $\mathcal{TC}_B$. For any R^i_W with $i > 0$ the twin-classes of $\mathcal{TC}_A$ that are subsets of R^i_W, but have not been assigned to $Q^0_A \cup Q^1_A$, will be assigned to R^i_A.

A twin-class TC^h_A of $\mathcal{TC}_A$ that has not been assigned yet (neither to Q^0_A, Q^1_A nor any R^i_A) must be a subset of R^0_W and must have at least two vertices in F_A. If TC^h_A does not have a neighbor in F_B then it is assigned to R^0_A. If TC^h_A does have a neighbor in F_B then since $F_A \cup F_B$ contains no cycle all vertices in TC^h_A have a single neighbor v_h in F_B common to them all. We arbitrarily pick indices $j \in \{1, 2, ..., k\}$ with R^j_A still empty (since $k = \mu w(V(G))$ we can find enough such j). We then assign remaining twin-classes using these indices such that two remaining twin-classes TC^h_A and TC^g_A both belong to the same R^j_A if and only if they have the same common neighbor in F_B. All remaining R^i_A should be empty. This completes the construction of $\mathcal{P}_A$. For $\mathcal{P}_B$ we do the analogous construction.

Claim 5.3. *The graph $G(\mathcal{P}_A, F_A)$ is isomorphic (respecting twin-classes) to the subgraph of $G(\mathcal{P}_W, F_A \cup F_B)$ induced on edges with either both endpoints in F_A or exactly one endpoint in a twin-class belonging to R^i_A for some i. Same holds for $G(\mathcal{P}_B, F_B)$.*

Proof. Note that $G(\mathcal{P}_A, F_A)$ and $G(\mathcal{P}_W, F_A \cup F_B)$ clearly induce the same graph on F_A. All remaining edges of $G(\mathcal{P}_A, F_A)$ are, by definition of $G(\mathcal{P}_A, F_A)$, accounted for by noting that two vertices in some twin-class of R^i_A have a common neighbor (not in A) if and only if their twin-classes belong to the same R^i_A for $i > 0$. But then they then have a common neighbor also in $G(\mathcal{P}_W, F_A \cup F_B)$, since by construction R^i_A is one of two types: either all twin-classes in R^i_A are subsets of some twin-class in R^i_W, in which case these vertices have a common neighbor in $G(\mathcal{P}_W, F_A \cup F_B)$, or all vertices of all twin-classes in R^i_A have a common neighbor in F_B. Since we are only showing that $G(\mathcal{P}_A, F_A)$ is (isomorphic to, while respecting twin-classes) a *subgraph* of $G(\mathcal{P}_W, F_A \cup F_B)$ this suffices. □

We now construct $\mathcal{C}_A$, following Definition 4.3. Consider the graph $G(\mathcal{P}_A, F_A)$ constructed from $G[F_A]$ by adding new vertices $\{v_1, v_2, ..., v_k\}$, and edges from v_i to all vertices in all twin-classes belonging to R^i_A, for $1 \leq i \leq k$. We define $\mathcal{C}_A$ to be the partition of $\mathcal{TC}_A \setminus (Q^0_A \cup R^0_A)$ such that TC^i_A, TC^j_A not in $(Q^0_A \cup R^0_A)$ are in the same class of $\mathcal{C}_A$ if and only if two vertices $u \in TC^i_A \cap F_A$ and $v \in TC^j_A \cap F_A$ belong to the same connected component of $G(\mathcal{P}_A, F_A)$. For $\mathcal{C}_B$ we do the analogous construction. We are thus done with construction of indices $(\mathcal{P}_A, \mathcal{C}_A)$ and $(\mathcal{P}_B, \mathcal{C}_B)$ in Tab_a and Tab_b.

Claim 5.4. *F_A satisfies $(\mathcal{P}_A, \mathcal{C}_A)$ and F_B satisfies $(\mathcal{P}_B, \mathcal{C}_B)$.*

Proof. We give the argument for F_A only since the argument for F_B is symmetric. It is obvious from the construction that F_A will satisfy the two first constraints in Definition 4.3 for $(\mathcal{P}_A, \mathcal{C}_A)$. By Claim 5.3 and the fact that $F_A \cup F_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$ the graph $G(\mathcal{P}_A, F_A)$ is a forest so that it satisfies the third constraint. By construction of $\mathcal{C}_A$ it is clear that F_A satisfies the fourth constraint. □

Based on the inductive assumption that Tab_a and Tab_b are correct we know that since F_A satisfies $(\mathcal{P}_A, \mathcal{C}_A)$ and F_B satisfies $(\mathcal{P}_B, \mathcal{C}_B)$ we have $\mathsf{Tab}_a[\mathcal{P}_A, \mathcal{C}_A] = S_A$ for some

largest $S_A \subseteq A$ satisfying $(\mathcal{P}_A, \mathcal{C}_A)$, $\mathsf{Tab}_a[\mathcal{P}_B, \mathcal{C}_B] = S_B$ for some largest $S_B \subseteq B$ satisfying $(\mathcal{P}_B, \mathcal{C}_B)$, and thus $|S_A| \geq |F_A|$, and $|S_B| \geq |F_B|$. Consider what happens when the algorithm considers the triple $(\mathcal{P}_A, \mathcal{C}_A)$, $(\mathcal{P}_B, \mathcal{C}_B)$, $(\mathcal{P}_W, \mathcal{C}_W)$. If $S_A \cup S_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$ then we are guaranteed that $|\mathsf{Tab}[\mathcal{P}_W, \mathcal{C}_W]| \geq |S_A \cup S_B| \geq |F_A \cup F_B| = |F_W|$ which will establish the contradiction.

Thus, it remains only to show that $S_A \cup S_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$, as in Definition 4.3. From Definition 4.3 we get that if TC^i_A belongs to Q^0_A or Q^1_A then $|S_A \cap TC^i_A| = |F_A \cap TC^i_A|$, same holds for B. A twin-class TC^i_W is a union of some twin-classes from $\mathcal{TC}_A$ and $\mathcal{TC}_B$. If TC^i_W belongs to Q^0_W or Q^1_W then it is a union of twin-classes belonging to Q^0_A, Q^1_A, Q^0_B or Q^1_B and hence $|(S_A \cup S_B) \cap TC^i_W| = |F_W \cap TC^i_W|$. Therefore by construction $S_A \cup S_B$ satisfies the first two constraints of Definition 4.3 for $(\mathcal{P}_W, \mathcal{C}_W)$. The following claims will be useful to show that the third constraint is satisfied.

Claim 5.5. *For any $i \geq 1$ all vertices in any non-empty twin-class of R^i_A have, in the graph $G(\mathcal{P}_W, S_A \cup S_B)$, exactly one neighbor not in A, which is common to them all. Same for R^i_B.*

Proof. We first show that no vertex x of a twin-class that is a subset of R^i_A can have more than one neighbor outside A in $G(\mathcal{P}_W, S_A \cup S_B)$. We do this by a case analysis showing that I: it cannot have any neighbor in an R^j_B-class, II: it cannot have two neighbors in Q^1_B-classes, III: it cannot have two new intermediate neighbors, and IV: it cannot have one intermediate neighbor and one neighbor in a Q^1_B-class.

Since by Claim 5.4 F_A, S_A satisfy $(\mathcal{P}_A, \mathcal{C}_A)$ and also F_B, S_B satisfy $(\mathcal{P}_B, \mathcal{C}_B)$ all twin-classes of R^i_A contain at least two vertices of S_A and of F_A, and R^j_B (for $j > 0$ an index of a non-empty R^j_B) contain at least two vertices of S_B and of F_B. Therefore, since all vertices in a twin-class of $\mathcal{TC}_A$ have the same neighbors outside A case I must hold since otherwise we would have a 4-cycle in $G(\mathcal{P}_W, F_A \cup F_B)$, contradicting that $F_A \cup F_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$. Likewise all twin-classes of Q^i_B contain one vertex of both S_B and of F_B. Therefore, case II must hold since otherwise we would again have a 4-cycle in $G(\mathcal{P}_W, F_A \cup F_B)$. By Definition 4.3 no vertex in the graph $G(\mathcal{P}_W, S_A \cup S_B)$ has more than one intermediate neighbor so case III holds. For case IV note first that the only way vertex x in a twin-class of R^i_A can have an intermediate neighbor in $G(\mathcal{P}_W, S_A \cup S_B)$ is if its twin-class is a subset of R^j_W for some $j > 0$. But then the vertex of F_A in the same twin-class will also have an intermediate neighbor in $G(\mathcal{P}_W, F_A \cup F_B)$. Since all twin-classes of Q^i_B contain one vertex of both S_B and of F_B we conclude that case IV must hold since otherwise we would again have a 4-cycle in $G(\mathcal{P}_W, F_A \cup F_B)$.

We now show that every vertex in any twin-class of R^i_A has a common neighbor outside A in $G(\mathcal{P}_W, S_A \cup S_B)$. Firstly, every vertex whose twin-class is a subset of R^j_W for $j > 0$ has a common intermediate neighbor. Secondly, for a vertex whose twin-class is a subset of R^0_W any neighbor it has outside A must be in S_B. Any two vertices of F_A in twin-classes of R^i_A have in $G(\mathcal{P}_A, F_A)$ a common neighbor, by Claim 5.4. By Claim 5.3 these two vertices of F_A have also in $G(\mathcal{P}_W, F_A \cup F_B)$ a common neighbor, which must be in F_B since R^i_A is a subset of R^0_W. If any two vertices x, y in F_A have a common neighbor in

F_B then any two vertices in S_A from the same twin-classes as x, y must have a common neighbor in F_B. This concludes the proof. □

Claim 5.6. *The graph $G(\mathcal{P}_A, S_A)$ is isomorphic (respecting twin-classes) to the subgraph of $G(\mathcal{P}_W, S_A \cup S_B)$ induced on edges with either both endpoints in S_A or exactly one endpoint in a twin-class belonging to R_A^i for some i. Same holds for $G(\mathcal{P}_B, S_B)$.*

Proof. Note that $G(\mathcal{P}_A, S_A)$ and $G(\mathcal{P}_W, S_A \cup S_B)$ clearly induce the same graph on S_A. For the edges with exactly one endpoint in a twin-class belonging to R_A^i Claim 5.5 implies that such an edge exists in both graphs or none of the two graphs. No other edges exist in $G(\mathcal{P}_A, S_A)$. □

To show that $S_A \cup S_B$ satisfies the third constraint of Definition 4.3 for $(\mathcal{P}_W, \mathcal{C}_W)$ we show that $G(\mathcal{P}_W, S_A \cup S_B)$ is a forest. We will prove this by contradiction, showing that if we have a cycle Ψ in $G(\mathcal{P}_W, S_A \cup S_B)$ then we also have a walk Ψ' containing a cycle in $G(\mathcal{P}_W, F_W)$ (which we know is a forest). We assume that the cycle Ψ is induced and break it into parts uniquely as follows (after first uniquely choosing parts of type 1 below and then of type 2 below the rest of the cycle will uniquely be of types 3 and 4 below):

1. maximal paths starting and ending in S_A containing at least one edge of $G[S_A]$ and otherwise only containing edges with exactly one endpoint in a twin-class belonging to R_A^i for some i.

2. maximal paths starting and ending in S_B containing at least one edge of $G[S_B]$ and otherwise only containing edges with exactly one endpoint in a twin-class belonging to R_B^i for some i.

3. crossings from S_A to S_B directly by one edge.

4. crossings using one intermediate new vertex v_i.

We assume additionally that the cycle Ψ has the smallest number of parts over all induced cycles. Each part starts and ends in a vertex of $S_A \cup S_B$ and these endpoints are called *special* vertices.

Claim 5.7. *Each twin-class of $\mathcal{TC}_A$ and $\mathcal{TC}_B$ contains at most one special vertex.*

Proof. Assume for contradiction that some twin-class contains two special vertices x, y. This twin-class must be some R_A^i (or R_B^i) since these are the only classes containing more than one vertex of S_A (or S_B). Any special vertex x in S_A (resp S_B) has two neighbors x_1, x_2 in the cycle Ψ. These two neighbors cannot both be in S_A (resp S_B), since x would then not be a special vertex. By Claim 5.5, x, y have exactly one neighbor not in A, thus wlog we can assume $x_1 = y_1 \notin A$ so that the cycle Ψ has consecutive vertices x, x_1, y. Their other neighbor(s) x_2, y_2 in the cycle are in S_A and thus x, y are not special vertices. □

Claim 5.8. *Ψ cannot have only one part.*

Proof. Note that Ψ must have at least two parts since if it had only one part this part would have to be of type 1 (or type 2) and be a cycle in which case Claim 5.6 would imply that $G(\mathcal{P}_A, S_A)$ (or $G(\mathcal{P}_B, S_B)$) had a cycle, and thus not satisfy $(\mathcal{P}_A, \mathcal{C}_A)$ (or $(\mathcal{P}_B, \mathcal{C}_B)$) contradicting Claim 5.4. □

We are ready to start constructing Ψ'. First, we choose for each special vertex $v \in S_A$ (resp S_B) of Ψ an arbitrary vertex of F_A (resp F_B) from the same twin-class as v. By Claim 5.7 we have thus chosen at most one vertex from each twin-class. We then replace each part of Ψ in $G(\mathcal{P}_W, S_A \cup S_B)$ by a path in $G(\mathcal{P}_W, F_W)$ between the two chosen vertices and call the resulting graph Ψ'.

Claim 5.9. *Parts of type 1 (resp 2) can be replaced by paths in $G(\mathcal{P}_W, F_A \cup F_B)$ containing at least one edge of $G[F_A]$ and otherwise only containing edges with exactly one endpoint in a twin-class belonging to R_A^i for some i (resp for type 2, replacing A by B).*

Proof. By Claim 5.6 a part of type 1 in Ψ, i.e. a path in $G(\mathcal{P}_W, S_A \cup S_B)$, gives a path in $G(\mathcal{P}_A, S_A)$. Since both S_A and F_A satisfy $(\mathcal{P}_A, \mathcal{C}_A)$ this gives a path in $G(\mathcal{P}_A, F_A)$. This means that for any part of type 1 between special vertices u, v there is a path in $G(\mathcal{P}_A, F_A)$ between the vertices $u'v'$ chosen from the same twin-classes as u, v, having edges in $G[F_A]$ and edges with exactly one endpoint in a twin-class belonging to R_A^i. We now show that there is such a path in $G(\mathcal{P}_A, F_A)$ between u', v' containing at least one edge of $G[F_A]$. Note that if u, v both belong to the same R_A^i then u, v have a common neighbor in $G(\mathcal{P}_W, S_A \cup S_B)$ and Ψ would have only one part and we apply Claim 5.8. Thus u', v' are in twin-classes belonging to different classes of the partition $\mathcal{P}_A$. Any path in $G(\mathcal{P}_A, F_A)$ between vertices in twin-classes belonging to different classes of $\mathcal{P}_A$ must contain an edge of $G[F_A]$.

Let us now argue that the existence of such a path from u' to v' in $G(\mathcal{P}_A, F_A)$ implies the existence of a similar path in $G(\mathcal{P}_W, F_A \cup F_B)$. Firstly, on F_A the induced subgraphs are the same in both graphs. Secondly, by construction of $\mathcal{P}_A$ from $\mathcal{P}_W$ any two vertices of F_A having in $G(\mathcal{P}_A, F_A)$ a common neighbor outside F_A also have in $G(\mathcal{P}_W, F_A \cup F_B)$ a common neighbor outside F_A.

We can similarly argue for parts of type 2. □

Parts of type 3 are easy to replace since if there is an edge in $G(\mathcal{P}_W, S_A \cup S_B)$ from a vertex in a twin-class TC_A^i to a vertex in a twin-class TC_B^j then in the graph $G(\mathcal{P}_W, F_W)$ any vertex in TC_A^i will be connected to all vertices in TC_B^j. Parts of type 4 are also easy to replace: such a part goes from a vertex in a twin-class TC_A^i via an intermediate new vertex to a vertex in twin-class TC_B^j with both twin-classes belonging to the same R_W^k and thus since F_W satisfies $\mathcal{P}_W, \mathcal{C}_W$ the same edges are present in $G(\mathcal{P}_W, F_W)$. This finishes the construction of Ψ' from Ψ.

Note that the subgraph Ψ' must be connected since we started with a cycle Ψ, then first replaced each special vertex v by some v' and then replaced paths of the cycle between special vertices u, v by paths connecting u' and v'. Ψ' is therefore a walk starting and ending in the same vertex and that is why Lemma 5.1 is useful to show that Ψ' contains a cycle.

Claim 5.10. *If Ψ contains an edge from a special vertex u in a twin-class of Q_A^1 (resp Q_B^1) to a vertex w not in A nor in a twin-class of R_B^i (resp not in B nor in a twin-class of R_A^i) then Ψ' contains a cycle.*

Proof. Note that for $u \in Q_A^1$ there are only two choices for w: it can either belong to a twin-class of Q_B^1 or it can be an intermediate vertex between u and a special vertex of B. The special vertex u of Ψ is replaced in Ψ' by u' in the same twin-class and no other special vertex of Ψ is replaced by u'. Similarly, if $w \in Q_B^1$ then it is replaced by a vertex w' from the same twin-class and it is the only such vertex. Otherwise we let $w' = w$. We will argue that the edge $u'w'$ appears exactly once in Ψ', and the statement will follow from Lemma 5.1. The edge $u'w'$ does not belong to $G[F_A]$ nor to $G[F_B]$ and neither u' nor w' belongs to some R_A^i or R_B^i. Therefore, by Claim 5.9 parts of type 1 or 2 in Ψ will never be replaced by a path containing the edge $u'w'$. A part of type 3 will be replaced by a single edge, and a part of type 4 by a path on two edges, from A to B. Two parts of type 3 cannot both be replaced by the same edge $u'w'$ since then the cycle Ψ would have contained the edge uw twice contradicting the fact that Ψ was chosen to be simple. Similarly, two parts of type 4 cannot both be replaced by a path containing the edge $u'w'$ since then the cycle Ψ would have contained the edge from u to its intermediate neighbor twice contradicting the fact that Ψ was chosen to be simple. Similarly we can argue for $u \in Q_B^1$. □

Claim 5.11. *If all edges of Ψ belong either to $G[S_A]$ or $G[S_B]$ or have at least one endpoint in R_A^i or R_B^i then Ψ' contains a cycle.*

Proof. By Claim 5.5 a vertex u in R_A^i (resp R_B^i) has a single neighbor outside B so that if the cycle Ψ contains a vertex u in a twin-class of R_A^i (resp R_B^i) then Ψ must contain an edge uv of $G[S_A]$ ($G[S_B]$). Thus Ψ has at least one part of type 1 or 2.

We now argue that the condition in the claim implies that the parts of Ψ must alternate between being: of type 1 (containing at least one edge of $G[S_A]$), then crossing parts of type 3 or 4 (containing no edges of $G[S_A]$ or $G[S_B]$), then of type 2 (containing at least one edge of $G[S_B]$), then again crossings, and so forth. Consider wlog a part of type 1 ending in a special vertex $u \in S_A$. The other part with special vertex u cannot be of type 1 or 2 since parts of type 1 are maximal and parts of type 2 have special vertices in S_B. We need to show that going around the cycle Ψ from u we must encounter a part of type 2 before we encounter a part of type 1 again (i.e. they alternate). When going around the cycle Ψ from u there are two cases, either we encounter a vertex in a twin-class of some R_B^i before encountering an edge of $G[S_A]$, or not. In the former case we would next encounter an edge of $G[S_B]$ by the observation at the start of the proof of this claim and would be done. In the latter case the condition in the claim implies that every edge of Ψ between u and the occurrence of the edge of $G[S_A]$ would have an endpoint in R_A^i, which would contradict the maximality of the type 1 part ending in u. We conclude that the parts alternate as described above.

Parts of type 3 and 4 in Ψ are replaced in Ψ' by crossings containing no edges of $G[A]$ or $G[B]$ and by Claim 5.9 parts of type 1 (resp 2) are replaced in Ψ' by paths containing at

least one edge of $G[A]$ and no edge of $G[B]$ (resp at least one edge of $G[B]$ and no edge of $G[A]$). Call an edge of Ψ' with both endpoints in $G[A]$ or both in $G[B]$ a one-sided edge. Firstly, if there is some one-sided edge uv of Ψ' that appears only once in the walk defined by Ψ' then by Lemma 5.1 the walk Ψ' contains a cycle. Otherwise, take a one-sided edge uv such that there are two appearances of it in the walk Ψ' with no other one-sided edge appearing twice in the part of the walk between these two appearances of uv. Note that there must be at least one special vertex between the two occurences.

As the types alternate, the subgraph induced by the edges between these two uses of uv in the walk Ψ' must contain a one-sided edge yz (on the other side). We therefore have a walk in Ψ' starting and ending in vertex u containing an edge yz used exactly once, so that Ψ' contains a cycle by Lemma 5.1. □

If Ψ does not fulfill the condition of Claim 5.11 then it contains an edge having one endpoint in a twin-class of Q^1_A (resp Q^1_B) and the other endpoint not in A (resp not in B) and not in a twin-class of R^i_B (resp R^i_A). But then Ψ fulfills Claim 5.10. Thus the existence of a cycle Ψ in $G(\mathcal{P}_W, S_A \cup S_B)$ implies a cycle in $G(\mathcal{P}_W, F_W)$ either by Claim 5.10 or 5.11, contradicting the fact that F_W satisfies $(\mathcal{P}_W, \mathcal{C}_W)$. Thus, we have shown that $S_A \cup S_B$ satisfies the third constraint of Definition 4.3 for $(\mathcal{P}_W, \mathcal{C}_W)$. We now show that it satisfies also the fourth constraint.

The argument for the fourth constraint is similar to the one for the third constraint but a bit simpler since we only need to replace paths by connected graphs and not cycles by connected graphs containing a cycle.

We show there exists a path Γ in $G(\mathcal{P}_W, S_A \cup S_B)$ from a vertex in $TC^i_W \notin R^0_W$ to a vertex in $TC^j_W \notin R^0_W$ if and only if there exists a path Γ' in $G(\mathcal{P}_W, F_W)$ from a vertex in TC^i_W to a vertex in TC^j_W. Since F_W satisfies the fourth constraint of Definition 4.3 for $(\mathcal{P}_W, \mathcal{C}_W)$, showing this will imply that also $S_A \cup S_B$ satisfies it. Given Γ we construct Γ' as follows. Break Γ into parts of 4 types in such a way that the endpoints of a part contains no vertex from R^0_W.

1. paths having edges in A only
2. paths having edges in B only
3. a single edge from S_A to S_B
4. a path of length two from S_A or S_B to S_A or S_B via a new vertex v_i

This can be done since vertices of R^0_W have no crossing edges. First replace each vertex $v \in TC^p_A$ (resp. $v \in TC^p_B$) that is the endpoint of a part of Γ by an arbitrary vertex v' of F_W in TC^p_A (resp. TC^p_B). A part (of type 1) with endpoints u, v containing only edges from A is a path also in $G(\mathcal{P}_A, S_A)$, and since both S_A and F_A satisfy $(\mathcal{P}_A, \mathcal{C}_A)$ such a part can be replaced by a path between u', v' in $G(\mathcal{P}_A, F_A)$, and since by Claim 5.3 $G(\mathcal{P}_A, F_A)$ is a subgraph of $G(\mathcal{P}_W, F_W)$ it can be replaced by a path between u', v' in $G(\mathcal{P}_W, F_W)$. The same holds for parts (of type 2) with endpoints u, v containing only edges from B. A part of type 3 is easy to replace: edges from S_A to S_B are replicated in

F_W since all vertices in a twin-class have the same neighbors on the other side. A part of type 4 is also easily replaced since they go from a vertex in a twin-class TC_A^j or TC_B^j to a vertex in a twin-class TC_A^p or TC_B^p both in R_W^i and thus in $G(\mathcal{P}_W, F_W)$ all vertices of these two twin-classes will be connected by such a path of length two via new vertex v_i. This concludes the construction of Γ' and shows that there is a path between u', v' in $G(\mathcal{P}_W, F_W)$. The opposite direction, given Γ' constructing Γ, is done in an analogous manner. Thus $S_A \cup S_B$ satisfies the fourth constraint of Definition 4.3 for $(\mathcal{P}_W, \mathcal{C}_W)$.

We have shown that $S_A \cup S_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$. Therefore we cannot have that $\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W] = \sharp$ and, since $|S_A \cup S_B| = |S_A| + |S_B| \geq |F_A| + |F_B| = |F_W|$, we cannot have $|F_W| > |\mathsf{Tab}_w[\mathcal{P}_W, \mathcal{C}_W]|$, finishing the proof. □

Theorem 5.12. *Given either a rooted decomposition tree (T, δ) of module-width k of a graph G, or a k-expression of a graph G of clique-width at most k, we can in $O(k^{5k}n^2)$ steps solve the Minimum Feedback Vertex Set problem on G.*

Proof. Consider first the case of input being a rooted decomposition tree. By Lemma 3.2 we can compute twin-classes for all nodes of the tree in time $O(n^2)$. Note that for any node a of the tree T the number of twin-classes of V_a is at most k. By Definition 4.3 and Lemma 5.2 the maximum value over all entries in the table at the root of our dynamic programming algorithm will correctly solve the problem.

The tables are indexed by $\mathcal{P}$, an ordered partition and $\mathcal{C}$ an unordered partition of a subset defined by $\mathcal{P}$. There are $O(k^{k+3})$ ordered partitions of $k+3$ elements. The number of unordered partitions of k elements is bounded by the number of ordered partitions. Note that the number of unordered partitions is so much smaller than the number of ordered partitions that it will cancel all factors polynomial in k, hence the size of the tables are $O(k^{2k})$. For the runtime, the bottleneck is the inner node update procedure which loops over all triples of table indexes $(\mathcal{P}_A, \mathcal{C}_A), (\mathcal{P}_B, \mathcal{C}_B), (\mathcal{P}_W, \mathcal{C}_W)$ and check if the union the two elements $S_A = \mathsf{Tab}_a[\mathcal{P}_A, \mathcal{C}_A]$ and $S_B = \mathsf{Tab}_b[\mathcal{P}_B, \mathcal{C}_B]$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$. This gives a total of $O(k^{6k})$ iterations, however $\mathcal{C}_W$ is uniquely defined by the other 5 elements hence only $O(k^{5k})$ iterations are needed. To check if $S_A \cup S_B$ satisfies $(\mathcal{P}_W, \mathcal{C}_W)$ we first make the union in $O(n)$ time and then check the four constraints of Definition 4.3 for $(\mathcal{P}_W, \mathcal{C}_W)$. The two first constraints are checked in $O(n)$ time, building the graph $G(\mathcal{P}_W, S_A \cup S_B)$ and checking if it is a forest is straight forward to do in $O(m)$ time. However if the two graphs $G[S_A]$ and $G[S_B]$ are stored from the previous step, then one can build the graph in $O(k^2 n)$ time, and since a forest has at most $n-1$ edges also check if $G(\mathcal{P}_W, S_A \cup S_B)$ is a forest. Checking that the connected components match $\mathcal{C}_W$ is also done in $O(n)$ time as long as the graph is a forest. In total the combine step is $O(k^{5k}n)$. The preprocessing is done in $O(n^2)$ time, initialization is done in $O(n \times k^{2k})$ time. Filling the tables requires n combine steps, hence the total running time is $O(n^2 k^{5k})$.

Note that within the same runtime we could instead have taken as input a k-expression of a graph G of clique-width at most k. This is since by Theorem 2.6 the module-width of G is no larger than the clique-width of G, and from the k-expression we easily derive a rooted decomposition tree of module-width at most k. □

6 Conclusion

The Feedback Vertex Set problem has a non-local property that does not lend itself to easy dynamic programming. Using the technique of 'expectation from the outside' in the definition of tables of the dynamic programming, and not the standard technique of partitioning the solution space into equivalence classes, we have given an FPT algorithm for FVS parametrized by the clique-width of a given decomposition. The exponential runtime of this algorithm matches the runtime of the best known deterministic algorithm when parametrizing by treewidth. Note that many graph classes have unbounded treewidth and bounded clique-width but the opposite cannot occur. It is already an open problem to solve FVS deterministically in exponential time $O^*(2^{O(tw)})$, so maybe $O^*(2^{O(cw)})$ is too much to hope for. Boolean-width and rank-width are parameters bounded on the same graph classes as clique-width, but their values can be exponentially smaller than clique-width. The best runtime known for FVS on graphs of rank-width rw is $O(2^{5rw^2}rw^3n)$ [15], and from this we can deduce an algorithm with runtime $O(2^{5(2^{2bw})+3bw}n)$ for graphs of boolean-width bw. Can we get runtime $O^*(2^{O(bw^2)})$ for boolean-width?

References

[1] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics* 12:289–297, 1999.

[2] R. Bar-Yehuda, D. Geiger, J. Naor, and R. Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. *SIAM Journal on Computing* 27:942–959, 1998.

[3] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. H-join decomposable graphs and algorithms with runtime single exponential in rank-width. *Discrete Applied Mathematics* 158(7):809–819, 2010. .

[4] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Fast FPT algorithms for vertex subset and vertex partitioning problems using neighborhood unions. `http://arxiv.org/abs/0903.4796`

[5] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Feedback Vertex Set on Graphs of Low Cliquewidth. In *Proceedings of IWOCA'09*, LNCS 5874, pages 113–124, 2009.

[6] J. Chen, F. Fomin, Y. Liu, S. Lu, and Y. Villanger. Improved Algorithms for the Feedback Vertex Set Problems. In *Proceedings of WADS'07*, LNCS 4619, pages 422–433, 2007.

[7] F. Chudak, M. Goemans, D. Hochbaum, and D. Williamson. A primal-dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Operations Research Letters* 22:111–118, 1998.

[8] B. Courcelle, J.A. Makowsky, U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

[9] M. Cygan, J. Nederlof, Ma. Pilipczuk, Mi. Pilipczuk, J. van Rooij and J. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. to appear in *Proceedings of FOCS'11*

[10] F. Dehne, M. Fellows, M. Langston, F. Rosamond, and K. Stevens. An $O(2^{O(k)}n^3)$ FPT Algorithm for the Undirected Feedback Vertex Set Problem. In *Proceedings of COCOON'05*, LNCS 3595, pages 859–869, 2005.

[11] R.Downey and M.Fellows. *Parameterized Complexity*, Springer-Verlag (1999)

[12] W. Espelage, F. Gurski, E. Wanke. How to Solve NP-hard Graph Problems on Clique-Width Bounded Graphs in Polynomial Time. In *Proceedings of WG'01*, LNCS 2204, pages 117–128, 2001.

[13] G. Even, J. Naor, B. Schieber, and L. Zosin. Approximating minimum subset feedback sets in undirected graphs with applications. *SIAM Journal on Discrete Mathematics* 13:255–267, 2000.

[14] F. Fomin, S. Gaspers, A. Pyatkin, and I. Razgon. On the Minimum Feedback Vertex Set Problem: Exact and Enumeration Algorithms. *Algorithmica*, 52:293–307, 2008.

[15] R. Ganian and P. Hliněný. On Parse Trees and Myhill-Nerode-type Tools for handling Graphs of Bounded Rank-width. *Discrete Applied Mathematics* 158(7):851–867, 2010.

[16] M. Goemans and D. Williamson. Primal-dual approximation algorithms for feedback problems in planar graphs. *Combinatorica* 18(1):37–59, 1998.

[17] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, and S. Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences* 72(8):1386–1396, 2006.

[18] M. Habib, C. Paul, L. Viennot. Partition Refinement Techniques: An Interesting Algorithmic Tool Kit. *International Journal of Foundations on Computer Science* 10(2):147–170, 1999.

[19] D. Harel and R. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[20] P. Hliněný, S. Oum. Finding branch-decompositions and rank-decompositions. *SIAM Journal on Computing* 38(3):1012–1032, 2008.

[21] R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[22] J. Kleinberg and A. Kumar. Wavelength conversion in optical networks. *Journal of Algorithms* 38:25–50, 2001.

[23] T. Kloks, C. Lee, J. Liu. New Algorithms for k-Face Cover, k-Feedback Vertex Set, and k-Disjoint Cycles on Plane and Planar Graphs. In *Proceedings of WG'02*, LNCS 2573, pages 282–295, 2002.

[24] D. Kobler, U. Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics* 126(2-3): 197–221, 2003.

[25] A. Koutsonas and D. Thilikos. Planar Feedback Vertex Set and Face Cover: Combinatorial Bounds and Subexponential Algorithms. In *Proceedings of WG'08*, LNCS 5344, pages 254–274, 2008.

[26] J.-M. Lanlignel. Autour de la décomposition en coupe. *Ph. D. thesis,* Université Montpellier II, 2001.

[27] S. Oum, P. Seymour. Approximating clique-width and branch-width. *J. Combin.Theory Ser. B* 96(4):514–528, 2006.

[28] R. Paige, R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing* 16(6):973–989, 1987.

[29] V. Raman, S. Saurabh, and C. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Transactions on Algorithms* 2(3):403–415, 2006.

[30] M. Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics* 308(24):6157–6165, 2008.

[31] N. Robertson, P. Seymour. Graph minors X: Obstructions to tree-decomposition. *Journal on Combinatorial Theory Series B,* 52:153–190, 1991.

[32] M. Rao. Décompositions de graphes et algorithmes efficaces. *Ph. D. thesis,* Université Paul Verlaine, Metz, 2006.