Feedback vertex set on graphs of low cliquewidth[†]

B.-M. Bui-Xuan J. A. Telle M. Vatshelle

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

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 cliquewidth cw and a cw-expression of G, we solve the Minimum Feedback Vertex Set problem in time $O(n^2 2^{2cw^2 \log cw})$. Our algorithm applies a non-standard dynamic programming on a so-called k-module decomposition of a graph, as defined by Rao [28], 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 cliquewidth 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 [20], circuit testing, deadlock resolution, analyzing manufacturing processes and computational biology (see [8] and its bibliography). It is one of the classical NP-complete problems from the 1972 list of Karp [19] and has been extensively studied from many viewpoints, including linear programming [6], approximation algorithms [2, 11, 14, 20], exact algorithms [12] and parameterized complexity [5, 8, 15, 23, 27].

The minimum FVS problem is 2-approximable in polynomial time [1]. The fastest exact algorithm has runtime $O(1.7548^n)$ [12]. The fastest FPT (Fixed Parameter Tractable) algorithm when parameterized by the size q of the FVS has runtime $O(5^q qn^2)$ [5]. 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 [21] give a dynamic programming algorithm solving minimum FVS on G in time $O(2^{O(tw \log tw)}n)$. A similar algorithm can be devised also for non-planar G of treewidth tw, given with an optimal tree-decomposition, but it is an open problem if algorithms with runtime $O(2^{O(tw)}n)$ exist for minimum 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 if given q vertices we cannot check that they form an FVS simply by looking at the neighbors of these q vertices. In this paper we consider instead graphs of cliquewidth cw, that

[†]Supported by the Norwegian Research Council, project PARALGO.

encompasses large classes of graphs of unbounded treewidth, and for which powerful algorithmic results are known. For instance, we have that any graph problem expressible in MSO_1-logic, as is the case with minimum FVS, is FPT when parameterized by cliquewidth (roughly, apply [18], then [25, Proposition 6.3], then [7]).

In this paper we will be interested in as low exponential dependency on cw as possible, and for this we need to use a specially designed dynamic programming algorithm. Dynamic programming based on decompositions having small cliquewidth are usually more complicated than dynamic programming on decompositions having small treewidth. The algorithm we give solves minimum FVS on a graph G of cliquewidth cw in time $O(2^{2cw^2 \log cw} n^2)$, when given a cw-expression of G which is a decomposition of the graph showing that it has cliquewidth cw. The only other problems for which $O(poly(n)2^{poly(cw)})$ algorithms exist are for problems based on domination-type properties, like Dominating Set in [22] and a class of vertex partitioning problems as in [4].

Cliquewidth is related to the notion of nlc-width of a graph [10] with which it shares most properties but we have chosen to use cliquewidth in this paper simply because that notion is more famous. Our algorithm applies a non-standard dynamic programming on a so-called k-module decomposition of a graph, as defined by Rao [28], 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 cliquewidth and nlc-width, and in this paper we give an alternative equivalent characterization of module-width. Our dynamic programming algorithm is non-standard in the sense that we index tables by the classes of one equivalence relation on the set of possible solutions, but store optimal solutions at these indices that are related to another equivalence relation which is a coarsening of the first one. We need to do this in order to achieve the stated runtime.

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 a set function over $V(G)$, 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 $f(V(G)) = f(\emptyset) = -\infty$, is equivalent to the one developed for the study of branch decomposition of symmetric and submodular

functions (see, *e.g.*, [25, Section 2] for a short and recent introduction). This includes the branch-width [29], rank-width [25], 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 cliquewidth 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 [24, 28].

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 terminology of module-width is according to the name given to an equivalent notion that was mentioned in [24, last two pages] and formalized in [28]. More precisely, Lanlignel and Rao defined so-called k-module decomposition, leading to the same parameter as follows. 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. This results in exactly the same notion of module-width of Definitions 2.4 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 cliquewidth and NLC-width refer to [7, 10]. The underlying graphs of cliquewidth 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 parameterized by one of these two parameters (roughly, apply [18], then [25, Proposition 6.3], then [7]). They are closely linked to module-width by the following property.

Theorem 2.5 *([28]) 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 2.6 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 P and a partition

of $V(G) \setminus X$ with set of classes Q, and injective functions $f_1 : P \to V_1$ and $f_2 : Q \to V_2$, 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 P and y to a class Q_j of 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 2.7 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 $i \in S$.

The following lemma captures the crucial property of module-width decompositions, from an H-join decomposition point of view. Its proof is straightforward.

Lemma 2.8 *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 $N(u) = \bigcup_{i \in S} v_i$, 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 2.9 *The function μ_G of Definition 2.4 is exactly equal to the function η_G defined by*

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

Proof Let $k = \eta_G(X)$. Since G is a Y_k-join across $(X, V(G) \setminus X)$, we have that X is a k-module and hence $\mu_G(X) \leq k$.

Now let H be the bipartite graph over color classes X and $V(G) \setminus X$ such that $xy \in E(H) \Leftrightarrow xy \in E(G)$ for all $x \in X$ and $y \notin X$. Clearly, G is an H-join across $(X, V(G) \setminus X)$. Now we do twin contractions on "the X side" of H until stability, and obtain H'. Clearly, G is an H'-join across $(X, V(G) \setminus X)$. Besides, the number of vertices on "the X side" of H' is exactly $l = \mu_G(X)$, namely the number of twin classes of X. We then apply Lemma 2.8 and deduce that G is an H''-join across $(X, V(G) \setminus X)$ for H'' being some induced subgraph of Y_l. In particular, this means G is a Y_l-join across $(X, V(G) \setminus X)$, and $l = \mu_G(X) \geq k$. □

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 having at most two children, and the leaves are in a bijection with the vertices of G. By contracting the internal nodes having one child, we will result in a rooted decomposition tree of G. Moreover, it can also be obtained from the proof of Theorem 2.5 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 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. For this, the algorithm given in [25] for transforming a rank decomposition into a cliquewidth expression can be used for a global runtime in $O(n^2 2^{2rw(G)})$. In this section, we will describe such a computation for every internal node u of T, with global runtime $O(n^2)$.

We will use the so-called partition refinement algorithmic technique (refer to, *e.g.*, [16, 26] 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.*, [16] 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 [17] 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 $O(n)$ 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 cliquewidth 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 cliquewidth expression. The module-width of this decomposition tree is at most the cliquewidth of the expression.

Lemma 3.2 *Given a graph G and either (T, δ) a rooted decomposition tree of G, or a cliquewidth 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 of vertices S 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. For a node a of T, let V_a be the vertices of G mapped to leaves in the subtree of T rooted at a. The runtime of the algorithm will be expressed as a function of $\mu_G(V_a)$, i.e. the number of twin-classes of such vertex subsets V_a.

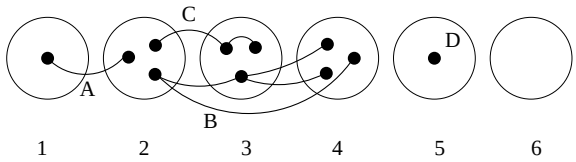

Figure 1: In the above figure is an example containing 6 twin-classes named 1,2,...,6 and a FI-set X having 11 vertices. T(X) contains 4 trees named A,B,C,D. The FI-pattern of X will be the 5-tuple $< S, W, S', W', P(S) >$, where $S = \{12, 24, 34, 44, 33\}$, and $W = \{23\}$ and $S' = \{1, 5\}$ and $W' = \{2, 3, 4\}$ and $P(S) = \{\{12\}, \{24, 34, 44\}, \{33\}\}$.

4.1 Two equivalence relations on FI-sets: by FI-classes and by FI-patterns

Let node a of T have $h_a = \mu_G(V_a)$ twin-classes $Q_a(1), ..., Q_a(h_a)$. We first consider how a FI-set $X \subseteq V_a$ interacts with twin-classes. We characterize X by a 5-tuple that we call $Pat_a(X)$ that records how the trees $T(X)$ in the forest induced by the FI-set X interact with twin-classes. Note that $T(X)$ may contain trees having only one vertex.

We first consider interaction of X on the set of pairs of twin-classes. Define $Z(X)$ to be the pairs (i, j) such that there is no path in $G[X]$ between a vertex in twin-class $Q_a(i)$ and a vertex in twin-class $Q_a(j)$. Define $S(X)$ to be the pairs (i, j) such that there is exactly one tree in $T(X)$ having a path between a vertex in twin-class $Q_a(i)$ and a vertex in twin-class $Q_a(j)$. Define $W(X)$ to be the pairs (i, j) such that there are at least two trees in $T(X)$ having a path between a vertex in twin-class $Q_a(i)$ and a vertex in twin-class $Q_a(j)$. Note that $Z(X), S(X), W(X)$ is a partition of the set of pairs of indices of twin-classes $\{(i, j) : 1 \leq i \leq j \leq h_a\}$, and thus given $S(X), W(X)$ we can uniquely identify $Z(X)$. Define also $\mathcal{P}(S(X))$ to be the partition of $S(X)$ into distinct trees, i.e. with two pairs (i, j) and (i', j') belonging to the same class of $\mathcal{P}(S(X))$ if and only if the same tree in $T(X)$ has both a path between vertices in twin-classes $Q_a(i)$ and $Q_a(j)$ and a path between vertices in twin-classes $Q_a(i')$ and $Q_a(j')$.

Now we consider interaction of X on twin-classes. Define $Z'(X), S'(X), W'(X)$ to be the partition of the set of indices $\{1, 2, ..., h_a\}$ of twin-classes such that if $i \in Z'(X)$ there is no vertex in X belonging to $Q_a(i)$, and if $i \in S'(X)$ there is exactly one vertex in X belonging to $Q_a(i)$, and if $i \in W'(X)$ there are at least two vertices in X belonging to $Q_a(i)$.

Definition 4.3 *For a FI-set $X \subseteq V_a$ define the FI-pattern $Pat_a(X)$ to be the 5-tuple $< S(X), W(X), S'(X), W'(X), \mathcal{P}(S(X)) >$.*

Our dynamic programming algorithm computing the size of a maximum FI-set in graph G will store a table Tab_a at each node a of the rooted decomposition tree T of G. For the computation of maximum FI-sets by a bottom-up traversal of T the following equivalence relation on FI-sets contained in V_a is clearly important.

Definition 4.4 *For two FI-sets $X, Y \subseteq V_a$ we define $X \equiv_a Y$ if for any FI-set $Z \subseteq V(G) \setminus V_a$ the set $X \cup Z$ is a FI-set if and only if $Y \cup Z$ is a FI-set. The equivalence classes of $\equiv_a$ are called FI-classes and for a FI-set $X \subseteq V_a$ we denote by $FIclass_a(X)$ the FI-class containing X.*

The table Tab_a will be indexed by FI-patterns, and the following lemma will be crucial for correctness of the algorithm.

Lemma 4.5 *If $Pat_a(X) = Pat_a(X')$ for two FI-sets $X, X' \subseteq V_a$ then $X \equiv_a X'$, i.e. $FIclass_a(X) = FIclass_a(X')$.*

Proof Consider a FI-set $Z \subseteq V(G) \setminus V_a$. We show that if $G[X \cup Z]$ contains a cycle then $G[X' \cup Z]$ contains a cycle, and the statement in the lemma will follow by symmetry. Let C be a chordless cycle in $G[X \cup Z]$ that has a minimum number of edges between a vertex of X and a vertex of Z. We call these crossing edges, and note that there are at least two crossing edges in C.

Let u_1v_1 be a crossing edge of C with $u_1 \in X, v_1 \in Z$. Let the cycle C continue from v_1 by a path $P(v_1, v_2)$ in $G[B]$ to vertex $v_2 \in Z$ and then a crossing edge v_2u_2, then a path $P(u_2, u_3)$ from u_2 to u_3 in $G[X]$ and again a crossing edge u_3v_3, etc. We thus get a total ordering $u_1v_1, u_2v_2, ..., u_{2k}v_{2k}$ on all crossing edges of C, with u_is in X and v_is in Z.

In this way we get k paths of cycle C belonging to $G[X]$, i.e. $k-1$ paths $P(u_{2i}, u_{2i+1})$ for $1 \leq i \leq k-1$, and also the path $P(u_{2k}, u_1)$. Note that any such path will contain edges and vertices from a single tree of $T(X)$, even if such a tree may be trivial, i.e. with $u_{2i} = u_{2i+1}$ or $u_{2k} = u_1$. Moreover, since the cycle C has the minimum number of crossing edges no tree of $T(X)$ contains a vertex belonging to two of these k paths.

Consider now $C_T = \{j : Q_a(j) \cap \{u_i : 1 \leq i \leq 2k\} \neq \emptyset\}$, i.e. the indices of twin-classes containing a vertex involved in a crossing edge. Note that $|Q_a(j) \cap \{u_i : 1 \leq i \leq 2k\}|$ can be at most two. Thus, to be able to exchange these k paths in $G[X]$ by equivalent paths in $G[X']$ we only need that the set of twin-classes containing respectively zero, one or at least two vertices of X are the same as the set of twin-classes containing containing respectively zero, one or at least two vertices of X', and this indeed holds since $Pat_a(X) = Pat_a(X')$ implies that $S'(X) = S'(X')$ and $W'(X) = W'(X')$ and hence also $Z'(X) = Z'(X')$. Thus, since $Pat_a(X) = Pat_a(X')$ we can exchange the set of k paths in $G[X]$, named $P(u_{2i}, u_{2i+1})$ for $1 \leq i \leq k-1$ and $P(u_{2k}, u_1)$, by k paths in $G[X']$ named $P(u'_{2i}, u'_{2i+1})$ for $1 \leq i \leq k-1$ and $P(u'_{2k}, u'_1)$, in such a way that vertex u_i, for any $1 \leq i \leq 2k$, is in the same twin class as u'_i, and also such that all the new paths belong to distinct trees in $T(Y)$. The latter statement is guaranteed by the fact that $\mathcal{P}(S(X)) = \mathcal{P}(S(X'))\}$ and $W(X) = W(X')$. Since any two vertices in the same twin class have the same neighbors in Z we have that also $G[X' \cup Z]$ contains a cycle and the lemma follows. □

Corollary 4.6 *The relation on FI-sets $\{X \subseteq V_a : X$ is a FI-set$\}$ given by $Pat_a(X) = Pat_a(Y)$ is an equivalence relation that is a refinement of $\equiv_a$.*

4.2 Tables and algorithm

The table Tab_a at node a of the rooted decomposition tree T of G, is defined as follows:

Definition 4.7 *The table Tab_a will be indexed by FI-patterns. For a FI-pattern P_a, let $FIclass_a(P_a)$ be the FI-class of the FI-sets having FI-pattern P_a, or let it be undefined if no such FI-set exists. Thus $FIclass_a(P_a) = FIclass_a(X)$ for any FI-set $X \subseteq V_a$ having $Pat_a(X) = P_a$. For a FI-class C let $max(C)$ be the maximum cardinality of a FI-set in C. If $FIclass_a(P_a)$ is undefined then define $max(FIclass_a(P_a))$ to be $-\infty$. The table Tab_a is correct when it satisfies the following two conditions:*

1. For any FI-pattern P_a we have $Tab_a[P_a] \leq max(FIclass_a(P_a))$

2. For any FI-class C there is FI-pattern P_a with $FIclass_a(P_a) = C$ and $Tab_a[P_a] = max(C)$

Whenever $Tab_a[P_a] \geq 0$ we also store with this table index some arbitrary FI-set F_a with $Pat_a(F_a) = P_a$. We are now ready to describe the algorithm computing the cardinality of a maximum FI-set of G. Let us start by noting that based on the above definition, the maximum entry over all table entries at the root of T will correctly solve the problem.

The algorithm starts by initializing all table entries to $-\infty$.

At any leaf a of the tree T we have $V_a = \{\delta(a)\}$ and set $Tab_a[\emptyset, \emptyset, \emptyset, \emptyset, \emptyset] = 0$ (and store the empty FI-set F_a) and $Tab_a[\emptyset, \emptyset, \{1\}, \emptyset, \emptyset] = 1$ (and store the FI-set $\{\delta(a)\}$ consisting of $\delta(a)$ belonging to the twin-class $Q_a(1)$).

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

For all pairs of patterns P_a from Tab_a and P_b from Tab_b with $Tab[P_a] \geq 0$ and $Tab[P_b] \geq 0$
  Let F_a and F_b be the FI-sets stored with these indices
  If $G[F_a \cup F_b]$ is acyclic (i.e. if $F_a \cup F_b$ is a FI-set)
    Compute $P_w = Pat_w(F_a \cup F_b)$
    If $Tab_w[P_w] = -\infty$ store $F_a \cup F_b$ with this index
    $Tab_w[P_w] = max(Tab_w[P_w], Tab_a[P_a] + Tab_b[P_b])$

4.3 Correctness and runtime

We consider correctness of the algorithm and start by a lemma showing that the equivalence relation given by FI-classes are well-behaved with respect to the child-parent relation of the rooted decomposition tree.

Lemma 4.8 *Let w be an inner node of T with children a and b. If $A \equiv_a A'$ and $B \equiv_b B'$ then $A \cup B \equiv_w A' \cup B'$.*

Proof It suffices to show that if $C \subseteq V(G) \setminus V_w$ is a FI-set such that $A \cup B \cup C$ is a FI-set, then also $A' \cup B' \cup C$ is a FI-set, and the lemma will follow by symmetry. We simply apply Definition 4.4 twice. Firstly, since $A \equiv_a A'$ and $A \cup (B \cup C)$ is a FI-set, we have that $A' \cup (B \cup C)$ is a FI-set. Secondly, since $B \equiv_b B'$ and $B \cup (A' \cup C)$ is a FI-set, we have that $B' \cup (A' \cup C)$ is a FI-set. □

Lemma 4.9 *Based on correct tables of children Tab_a, Tab_b, the algorithm will correctly update the parent table Tab_w.*

Proof We first show that Tab_w will satisfy condition 1 in Definition 4.7. If after updating we have $Tab_w[P_w] = k$ for some integer k then there must be a pair P_a, P_b with $Tab_a[P_a] = k_a$ and $Tab_b[P_b] = k_b$ with $k = k_a + k_b$, and since condition 1 holds for the children tables we have $max(FIclass_a(P_a)) \geq k_a$ and $max(FIclass_b(P_b)) \geq k_b$. Thus, there exists FI-sets $X_a \subseteq V_a$ and $X_b \subseteq V_b$ with $FIclass_a(X_a) = FIclass_a(P_a)$ and $FIclass_b(X_b) = FIclass_b(P_b)$ and $|X_a| \geq k_a$ and $|X_b| \geq k_b$. Moreover, we have stored FI-sets F_a and F_b, with $FIclass_a(F_a) = FIclass_a(P_a)$ and $FIclass_b(F_b) = FIclass_b(P_b)$, by Lemma 4.5, and in the algorithm we have checked that $F_a \cup F_b$ is a FI-set. We now apply Lemma 4.8 to conclude that also $X_a \cup X_b$ is a FI-set. By Lemma 4.8 we also get the stronger statement that $FIclass_w(F_a \cup F_b) = FIclass_w(X_a \cup X_b)$. We have thus shown that if $Tab_w[P_w] = k$ then there must exist a FI-set $X_a \cup X_b$ with $|X_a \cup X_b| \geq k$ and $FIclass_w(X_a \cup X_b) = FIclass_w(P_w)$ so that condition 1 is satisfied.

We now show that Tab_w will satisfy condition 2 in Definition 4.7. Consider a FI-class C of $\equiv_w$. Let $Z \subseteq V_w$ be a FI-set in this class. Note that $Z_a = Z \cap V_a$ and $Z_b = Z \cap V_b$ must be FI-sets. Since condition 2 holds for the children tables we therefore have Pa and P_b with $Tab_a[P_a] \geq |Z_a|$ and $Tab_b[P_b] \geq |Z_b|$ and $FIclass_a(P_a) = FIclass_a(Z_a)$ and $FIclass_b(P_b) = FIclass_b(Z_b)$. When the algorithm considers the pair P_a, P_b we find stored FI-sets F_a and F_b which by Lemma 4.5 have the property that $FIclass_a(F_a) = FIclass_a(Z_a)$ and $FIclass_b(F_b) = FIclass_b(Z_b)$. Therefore, applying Lemma 4.8 we get that since $Z_a \cup Z_b$ is a FI-set then $F_a \cup F_b$ is a FI-set. Moreover, we also get the stronger statement that $C = FIclass_w(F_a \cup F_b) = FIclass_w(Z_a \cup Z_b)$. We have thus shown that for any FI-class C containing a FI-set $Z \subseteq V_w$ there will be a FI-pattern $P_w = Pat_w(F_a \cup F_b)$ with $FIclass_w(P_w) = C$ such that $Tab_w[P_w] \geq |Z_a| + |Z_b| = |Z|$. We conclude that also condition 2 will be satisfied. □

Theorem 4.10 *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 cliquewidth at most k, we can in $O(2^{2k^2 \log k} n^2)$ 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 4.7 and Lemma 4.9 the maximum value over all entries in the table at the root of our dynamic programming algorithm will correctly solve the problem.

For the runtime, the bottleneck is the inner node update procedure which loops over all *pairs* of patterns P_a, P_b. The number of patterns is bounded by the number of choices for the 5-tuples $S, W, S', W', \mathcal{P}(S)$. An upper bound on the number of choices for the third and fourth components jointly (3-partitions of the k twin-classes) is 3^k, while for the first and second components jointly (3-partitions of unordered pairs of twin-classes) it is $3^{\frac{k^2+k}{2}}$. The number of choices for the fifth component can be (loosely) upper bounded by the number of partitions of unordered pairs of indices of the k twin-classes. This gives an upper bound on the total number of patterns $2^{k^2 \log k}$. In the update procedure we spend for each pair of patterns time at most $O(nk^2)$ to check if the union of two FI-sets are a FI-set and to compute the new pattern, making use of the fact that two vertices in the same twin class have the same neighbors across the cut. Since there are at most n inner nodes the runtime in the theorem follows.

Note that within the same runtime we could instead have taken as input a k-expression of a graph G of cliquewidth at most k. This since by Theorem 2.5 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. □

Note: After submitting this paper we have learned that Ganian and Hliněný have a manuscript [13] with an algorithm solving FVS in time $O(rw^2|V(G)|^2 + 2^{5rw^2} rw^3|V(G)|)$ parameterized by the rankwidth rw of G, when a rank decomposition of width rw is given.

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 rankwidth. *to appear in Discrete Applied Mathematics: special issue of GROW*.

[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] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] W. Espelage, F. Gurski, E. Wanke. How to Solve NP-hard Graph Problems on Clique-Width Bounded Graphs in Polynomial Time. Proceedings WG 2001: 117-128

[11] 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.

[12] 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.

[13] R. Ganian and P. Hliněný. On Parse Trees and Myhill-Nerode-type Tools for handling Graphs of Bounded Rank-width.
`http://www.fi.muni.cz/~hlineny/Research/papers/MNtools-dam3.pdf`

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

[15] 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.

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

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

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

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

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

[21] 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.

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

[23] 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.

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

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

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

[27] 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.

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

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