An exact algorithm for Subset Feedback Vertex Set on chordal graphs[⋆]

Petr A. Golovach[1], Pinar Heggernes[1], Dieter Kratsch[2], and Reza Saei[1]

[1] Department of Informatics, University of Bergen, Norway,
{petr.golovach,pinar.heggernes,reza.saeidinvar}@ii.uib.no
[2] LITA, Université de Lorraine - Metz, France, kratsch@univ-metz.fr

Abstract. Given a graph $G = (V, E)$ and a set $S \subseteq V$, a set $U \subseteq V$ is a subset feedback vertex set of (G, S) if no cycle in $G[V \setminus U]$ contains a vertex of S. The Subset Feedback Vertex Set problem takes as input G, S, and an integer k, and the question is whether (G, S) has a subset feedback vertex set of cardinality or weight at most k. Both the weighted and the unweighted versions of this problem are NP-complete on chordal graphs, even on their subclass split graphs. We give an algorithm with running time $O(1.6708^n)$ that enumerates all minimal subset feedback vertex sets on chordal graphs with n vertices. As a consequence, Subset Feedback Vertex Set can be solved in time $O(1.6708^n)$ on chordal graphs, both in the weighted and in the unweighted case. On arbitrary graphs, the fastest known algorithm for the problems has $O(1.8638^n)$ running time.

1 Introduction

Given a graph $G = (V, E)$ and a set $S \subseteq V$, a set $U \subseteq V$ is a subset feedback vertex set of (G, S) if no cycle in $G[V \setminus U]$ contains a vertex of S. A subset feedback vertex set U is minimal if no subset feedback vertex set of (G, S) is a proper subset of U. The Subset Feedback Vertex Set problem takes as input G, S, and an integer k, and the question is whether (G, S) has a subset feedback vertex set of cardinality at most k. In the weighted version of the problem, every vertex of G has a weight, and the question is whether there is a subset feedback vertex set of total weight at most k.

Subset Feedback Vertex Set was introduced by Even et al. [4], and it generalizes several well-studied problems. When $S = V$, it is equivalent to the classical Feedback Vertex Set problem [11], and when $|S| = 1$, it generalizes the Multiway Cut problem [7]. Weighted Subset Feedback Vertex Set admits a polynomial-time constant-factor approximation algorithm [4]. The unweighted version of the problem is fixed parameter tractable [3]. The only exact algorithm known for its weighted version is by Fomin et al. [7] and it runs in $O(1.8638^n)$ time and solves the problem by enumerating all minimal subset feedback vertex sets.

[⋆] This work has been supported by the European Research Council, the Research Council of Norway, and the French National Research Agency.

As a comparison, the unweighted version of Feeback Vertex Set can be solved in time $O(1.7347^n)$ [9], whereas the best algorithm for its weighted version runs in time $O(1.8638^n)$ and enumerates all minimal feedback vertex sets [5]. Feeback Vertex Set has also been studied on many graph classes, like chordal graphs and AT-free graphs [1, 15], and several positive results exist. This is not yet the case for Subset Feedback Vertex Set, and no algorithm with a running time of $O(c^n)$ such that $c < 1.8637$ is known for any significant graph class. Interestingly, whereas both the weighted and the unweighted versions of Feeback Vertex Set are solvable in polynomial time on chordal graphs [1, 19], even the unweighted version of Subset Feedback Vertex Set is NP-complete on chordal graphs; in fact on their restricted subclass split graphs, by a standard reduction from Vertex Cover [7].

In this paper we give an algorithm with running time $O(1.6708^n)$ that enumerates all minimal subset feedback vertex sets when the input graph is chordal. As a consequence, Subset Feedback Vertex Set can be solved in time $O(1.6708^n)$ on chordal graphs, both in the weighted and in the unweighted case. Our algorithm differs completely from the $O(1.8638^n)$ time algorithm of [7] for the general case, and it heavily uses the structure of chordal graphs. Chordal graphs form one of the most studied graph classes; they have extensive practical applications in several fields [10, 12, 18], and they are crucial in characterizing and understanding fundamental algorithmic tools, like treewidth.

Enumeration algorithms are central in the field of Exact Exponential Algorithms, as the running times of many exact exponential time algorithms rely on the maximum number of various objects in graphs [8]. A classical example is the widely used result of Moon and Moser [16], showing that the maximum number of maximal cliques or maximal independent sets in an n-vertex graph is $3^{n/3}$. More recently, the maximum numbers and enumeration of objects like minimal dominating sets, minimal feedback vertex sets, minimal subset feedback vertex sets, minimal separators, and potential maximal cliques, have been studied; see e.g., [5–7, 9, 13, 14, 17]. The maximum number of such objects in graphs have traditionally found independent interest also in graph theory and combinatorics.

The results we present in this paper give an upper bound of $O(1.6708^n)$ on the maximum number of minimal subset feedback vertex sets a chordal graph can have. A tight bound on the maximum number of minimal feedback vertex sets on chordal graphs is known to be 1.5848^n [2], and this thus gives a lower bound on the maximum number of minimal subset feedback vertex sets on chordal graphs. Consequently, our results tighten the gap between the upper and lower bounds on the maximum number of subset feedback vertex sets on chordal graphs. The corresponding gap is much larger on general graphs. There, the maximum numbers of minimal feedback and subset feedback vertex sets are both $O(1.8638^n)$ [5, 7], but no examples of graphs having 1.5927^n or more minimal feedback or subset feedback vertex sets are known [5]. Note that the maximum number of minimal subset feedback vertex sets can be dramatically different from the maximum number of minimal feedback vertex sets. Split graphs, which form a subclass of chordal graphs, have at most n^2 minimal feedback vertex sets, whereas they can have $3^{n/3}$ minimal subset feedback vertex sets [7].

2 Preliminaries

We work with simple undirected graphs. We denote such a graph by $G = (V, E)$, where V is the set of vertices and E is the set of edges of G. We adhere to the convention that $n = |V|$. The set of neighbors of a vertex $v \in V$ is denoted by $N_G(v)$. The degree of v, $|N_G(v)|$, is denoted by $d_G(v)$. The *closed neighborhood* of v is $N_G[v] = N(v) \cup \{v\}$. For a vertex subset $X \subseteq V$, the subgraph of G induced by X is denoted by $G[X]$. For ease of notation, we use $G - v$ to denote the graph $G[V \setminus \{v\}]$, and $G - X$ to denote the graph $G[V \setminus X]$.

A *path* in G is a sequence of distinct vertices such that the next vertex in the sequence is adjacent to the previous vertex. A *cycle* is a path with at least three vertices such that the last vertex is in addition adjacent to the first. Given a subset $S \subseteq V$, we call a cycle an S*-cycle* if it contains a vertex of S. For a cycle or an S-cycle C, we use $V(C)$ to denote the set of vertices in C. A subset $F \subseteq V$ will be called a *forest* if $G[F]$ contains no cycle. Similarly, F is an S-forest if no cycle in $G[F]$ contains a vertex of S. A graph is *connected* if there is a path between every pair of its vertices. A maximal connected subgraph of G is called a *connected component* of G. A set $X \subseteq V$ is a *clique* if $uv \in E$ for every pair of vertices $u, v \in X$; and X is an *independent set* if $uv \notin E$ for every pair of vertices $u, v \in X$.

A *chord* of a cycle is an edge between two non consecutive vertices of the cycle. A graph is *chordal* if every cycle of length at least 4 contains a chord. Induced subgraphs of chordal graphs are also chordal [12]. A vertex v is called *simplicial* if $N(v)$ is a clique. Every chordal has a simplicial vertex [12]. A graph is a *split graph* if its vertex set can be partitioned into a clique and an independent set. Split graphs are chordal.

Given a set $S \subseteq V$, a set $U \subseteq V$ is a *subset feedback vertex set (sfvs)* of (G, S) if no cycle in $G - U$ contains a vertex of S. Observe that U is a sfvs of (G, S) if and only if $V \setminus U$ is an S-forest. If $S = V$ then U is a *feedback vertex set (fvs)* of G, and $V \setminus U$ is a forest. A sfvs U is *minimal* if no proper subset of U is a sfvs of (G, S), and an S-forest is *maximal* if it cannot be extended to a larger S-forest by including more vertices of G. Clearly, U is a minimal sfvs of (G, S) if and only if $V \setminus U$ is a maximal S-forest of G. Consequently, the number of minimal sfvs of (G, S) is equal to the number of maximal S-forests of G.

Let $\mu(G, S)$ denote the number of minimal svfs of (G, S), equivalently the number of maximal S-forests of G. Observe that $\mu(G, S) = \prod_{i=1}^{t} \mu(G_i, S)$, where $G_1, G_2, \ldots, G_t$ are the connected components of G. This is because every maximal S-forest of G is the union of maximal S-forests of the connected components of G.

Let $\mu(G) = \max\{\mu(G, S) \mid S \subseteq V\}$. Note that $\mu(G)$ is lower bounded by the number of minimal fvs of G. Let H be the complete graph on 5 vertices. This graph has 10 minimal fvs [2]. Let H_ℓ be the graph obtained by taking ℓ

disjoint copies of H, for $\ell \geq 1$. The number of minimal feedback vertex sets of H_ℓ is thus $10^\ell = 10^{n/5} \approx 1.5848^n$. Any graph H_ℓ is chordal and hence $10^{n/5}$ is a lower bound on the number of minimal sfvs of chordal graphs, i.e., there is a chordal graph $G = (V, E)$ and a set $S \subseteq V$ such that (G, S) has $10^{n/5}$ minimal sfvs. When it comes to the maximum number of minimal fvs in chordal graphs, Couturier et al. showed that the above lower bound is also the upper bound [2]. An upper bound on the number of minimal sfvs of chordal graphs better than the one for general graphs has not been known until the result we present below.

3 Enumerating minimal subset feedback vertex sets in chordal graphs

This section is devoted to proving the following theorem.

Theorem 1. *All minimal subset feedback vertex sets of a chordal graph on n vertices can be listed in $O(1.6708^n)$ time.*

Two corollaries follow from the above result. Corollary 1 follows immediately, whereas Corollary 2 follows by noting that any sfvs of minimum cardinality or minimum weight is a minimal sfvs. Hence we can check the cardinality or weight of each generated minimal sfvs, and compare the smallest one with the given bound k of the input.

Corollary 1. *A chordal graph on n vertices has at most $O(1.6708^n)$ minimal subset feedback vertex sets.*

Corollary 2. *Both weighted and unweighted versions of* Subset Feedback Vertex Set *can be solved in $O(1.6708^n)$ time on chordal graphs.*

To prove Theorem 1, we will describe an algorithm that takes as input a chordal graph $G = (V, E)$ and a vertex subset $S \subseteq V$, and lists all maximal S-forests of G. Our algorithm is a recursive branching algorithm; every maximal S-forest of G will be present at some leaf of the corresponding branching tree, whereas some of the leaves might not correspond to maximal S-forests. Every recursive call has input (G', F, U, R), where F is the set of vertices of G placed so far in an S-forest of G, U is the set of vertices so far deleted from G and hence placed in the corresponding sfvs, $R \subseteq F$ is the set of vertices that are placed in F and that are no longer relevant for making further decisions, and $G' = G - (U \cup R)$. We call the vertices in R *hidden*. The vertices in $V \setminus (U \cup F)$ are called *undecided* vertices. As G and S do not change throughout the algorithm, they are not parts of the input to the recursive calls. Given G and S, the main program runs the recursive branching algorithm on $(G, \emptyset, \emptyset, \emptyset)$.

If at some call (G', F, U, R), the graph G' has no undecided vertices, then we are at a leaf of the branching tree, and the algorithm stops after checking whether F is a maximal S-forest of G. If F is a maximal S-forest, it is added to the list of S-forests that will be output. If G' has undecided vertices, the algorithm continues, but first it checks whether F is an S-forest. If not, then the algorithm stops, discards F since it can never lead to a maximal S-forest, and no new subproblems are generated from this instance. If the algorithm continues, then since G' is chordal, we know that it has a simplicial vertex. The algorithm chooses an arbitrary simplicial vertex v of G' and makes choices depending on v. Vertex v might already be placed in F or not; these two cases will be handled separately in the first two subsections below. The following operations will be used in our algorithm:

– *Deleting* a vertex x: deletes x from G' and adds it to U. Vertex x will be permanently deleted from G' and it will be a part of the suggested sfvs U in all subsequent subproblems.
– *Adding* a vertex x to F: adds x to F. Vertex x will be a part of F in all subsequent subproblems, and will never be considered for deletion.
– *Hiding* a vertex x of F: this operation is only applicable on some simplicial vertices of G' that are already placed in F. We apply it when x is no longer relevant for making further decisions on the remaining vertices of $G' - F$. When x is hidden, it is added to R and removed from G' but it remains a part of F in all subsequent subproblems, and in particular it remains in $G - U$.

Throughout the algorithm we will keep the following invariant.

Invariant 1. *Let (G', F, U, R) be an instance. For any S-cycle C in $G - U$ that contains a vertex of R, there is an S-cycle C' in G' such $V(C') = V(C) \setminus R$.*

Invariant 1 is clearly true when R is empty. Whenever we hide a vertex v, we will argue that the invariant is still true after v is hidden. The next lemma shows that we can safely ignore the vertices in R when we make further decisions on $G - U$, and hence it is safe to work on $G' = G - (U \cup R)$ instead of $G - U$.

Lemma 1. *Let (G', F, U, R) be an instance. Under Invariant 1, F' is a maximal S-forest of $G - U$ such that $F \subseteq F'$ if and only if $F' \setminus R$ is a maximal S-forest of G'.*

Proof. Let F' be a maximal S-forest of $G - U$ such that $F \subseteq F'$. Then clearly $F' \setminus R$ is an S-forest in G'. Let us argue for maximality. Since F' is maximal, for any vertex x of $G - (U \cup F')$, x is involved in an S-cycle C in $G - U$ such that $V(C) \subseteq F' \cup \{x\}$. Observe that since $R \subseteq F \subseteq F'$, any such vertex x is also a vertex in G'. By Invariant 1, x is involved in an S-cycle C' in G' such that $V(C') = V(C) \setminus R$. Since $G' = G - (U \cup R)$, it follows that $V(C') \subseteq (F' \setminus R) \cup \{x\}$. Hence x cannot be added to $F' \setminus R$, which is thus a maximal S-forest of G'.

For the other direction, assume that $F' \setminus R$ is a maximal S-forest of G'. Hence every vertex x in G' outside of $F' \setminus R$ is involved in an S-cycle C in G' such that $V(C) \subseteq (F' \setminus R) \cup \{x\}$. Since $G - U$ is a supergraph of G', C is also an S-cycle in $G - U$. Hence no more vertices can be added to F', which is thus maximal. Let us argue that F' is an S-forest. Assume for contradiction that it is not. Then a vertex y of R is involved in an S-cycle C in $G - U$ such that $V(C) \subseteq F'$. Then by Invariant 1, there is an S-cycle C' in G' such that $V(C') \subseteq F' \setminus R$, which contradicts the assumption that $F' \setminus R$ is an S-forest of G'. $\square$

The *measure* of an instance (G', F, U, R) is the number of undecided vertices, i.e., the vertices in $G' - F$. In the beginning of the algorithm all vertices are undecided and hence the measure of $(G, \emptyset, \emptyset, \emptyset)$ is n. The measure drops by the number of vertices deleted from G' plus the number of vertices added to F. Hiding a vertex does not affect the measure of an instance. In the call with input (G', F, U, R), the algorithm will further branch into subproblems in which some vertices will be deleted from G' and some vertices will be placed in F, and the measure will drop accordingly. If at a step, we branch into t new subproblems, where the measure decreases by $c_1, c_2, \ldots, c_t$ in each subproblem, respectively, we get the *branching vector* $(c_1, c_2, \ldots, c_t)$. At each branching point, we will give the corresponding branching vector to prepare for the running time analysis, which will be given in the last subsection of this section.

We now describe the reduction and the branching rules of the algorithm when G' has undecided vertices and F is an S-forest. Let (G', F, U, R) be a call of the algorithm satisfying this. In the below, we let $N(v) = N_{G'}(v)$, $N[v] = N_{G'}[v]$, and $d(v) = d_{G'}(v)$. First, we state three reduction rules. These rules are applied recursively on the considered instance as long as it is possible to apply at least one of them. It is easy to see that the first reduction rule is safe:

Rule A. *If in G' an undecided vertex v is adjacent to vertices $u, w \in F$ such that $uw \in E$ and $\{u, v, w\} \cap S \neq \emptyset$, then delete v, i.e., reduce to the subproblem $(G' - v, F, U \cup \{v\}, R)$.*

The following observation immediately results in the next reduction rule: Rule B.

Observation 1 *Let v be a vertex of G' such that no S-cycle of G' contains v. Then v must be added to F if it is not in F, and it is then safe to hide v.*

Rule B. *If G' has a vertex v with $d(v) \leq 1$, then add v to F if v is undecided, and when $v \in F$ then hide v, i.e., reduce to the subproblem $(G' - v, F \cup \{v\}, U, R \cup \{v\})$.*

Since G' is not empty and it is chordal, it has a simplicial vertex. With the following observation we obtain the next reduction rule: Rule C.

Observation 2 *Let v be a simplicial vertex of G'. If $N[v] \cap S = \emptyset$, then v must be added to F if it is not already in F, and it is then safe to hide v.*

Rule C. *If there is a simplicial vertex v such that $N[v] \cap S = \emptyset$, then add v to F if v is undecided, and when $v \in F$ then hide v, i.e., reduce to the subproblem $(G' - v, F \cup \{v\}, U, R \cup \{v\})$.*

If we cannot apply Rules A–C, then we start branching. To do it, we pick a simplicial vertex v, hence $N(v)$ is a clique. If vertex v is undecided then we proceed as described in the first subsection below. If $v \in F$ then we proceed as described in the second subsection below. Notice that by Rule B, $d(v) \geq 2$.

3.1 The chosen simplicial vertex v is undecided

Case 3.1.1: $v \notin F$, $v \in S$, and $N(v) \cap F = \emptyset$.

If $d(v) = 2$ then let u_1 and u_2 be the two neighbors of v. Since $v \in S$, at most two vertices from $\{v, u_1, u_2\}$ can be added to F. Note however that, if exactly one of u_1, u_2 is added to F and the other one is deleted, then v must also be added to F by Observation 1. This implies that if v is deleted then both u_1 and u_2 must be added to F. Consequently, we branch into the following subproblems, which cover all possibilities, and we obtain $(3, 3, 3, 3)$ as the branching vector:

- Vertex v is deleted from G' and added to U; vertices u_1 and u_2 are added to F: the decrease in the measure is 3.
- Vertex u_1 is deleted from G' and added to U; vertices v and u_2 are added to F: the decrease is 3.
- Vertex u_2 is deleted from G' and added to U; vertices v and u_1 are added to F: the decrease is 3.
- Vertices u_1 and u_2 are deleted from G' and added to U; vertex v is added to F: the decrease is 3.

If $d(v) = 3$ then let u_1, u_2, u_3 be the three neighbors of v. Again, at most two vertices from $\{v, u_1, u_2, u_3\}$ can be added to F. As above, we will branch on the possibilities of adding v and at most one of its neighbors into F and deleting the other neighbors, or deleting v. For the choice of deleting v, we observe the following: either u_1 is added to F or u_1 is also deleted. If both v and u_1 are deleted, then both u_2 and u_3 must be added to F, by Observation 1. Consequently, we branch into the following subproblems, which cover all possibilities, and we obtain $(4, 4, 4, 4, 2, 4)$ as the branching vector:

- Vertices u_2 and u_3 are deleted from G' and added to U; vertices v and u_1 are added to F: the decrease is 4.
- Vertices u_1 and u_3 are deleted from G' and added to U; vertices v and u_2 are added to F: the decrease is 4.
- Vertices u_1 and u_2 are deleted from G' and added to U; vertices v and u_3 are added to F: the decrease is 4.
- Vertices u_1, u_2, and u_3 are deleted from G' and added to U; vertex v is added to F: the decrease is 4.
- Vertex v is deleted from G' and added to U; vertex u_1 is added to F: the decrease is 2.
- Vertices v and u_1 are deleted from G' and added to U; vertices u_2 and u_3 are added to F: the decrease is 4.

In the rest we assume that $t = d(v) \geq 4$. By the same arguments as above, either v is deleted or it is added to F with at most one of its neighbors. Consequently, we branch into the following subproblems, where $u_1, u_2, \ldots, u_t$ are the neighbors of v in G':

- Vertex v is deleted from G' and added to U; nothing else changes: the decrease in the measure is 1.

- Vertex v is added to F; all of its neighbors are deleted from G' and added to U: the decrease in the measure is $t+1$.
- Vertices v and u_1 are added to F; all other neighbors of v are deleted from G' and added to U: the decrease is $t+1$.
- The last step above is repeated with each of the other neighbors of v instead of u_1: the decrease is $t+1$ in each of these $t-1$ additional cases.

The branching vector is $(1, t+1, t+1, \ldots, t+1)$, where the term $t+1$ appears $t+1$ times, and $t \geq 4$.

Case 3.1.2: $v \notin F$, $v \in S$, and $N(v) \cap F \neq \emptyset$.

As we cannot apply Rule A for the considered instance, $|N(v) \cap F| = 1$. Since $t = d(v) \geq 2$, we know that v has exactly one neighbor in F, say $u_1 \in F$, whereas the rest of its neighbors $u_2, \ldots, u_t$ are undecided. We branch into the two possibilities of adding v to F or deleting v. If we add v to F, since one neighbor is already in F then none of the $t-1$ undecided neighbors can be added, and therefore we delete them from G' and add them to U. We get the following two subproblems: $(G' - v, U \cup \{v\}, F, R)$ and $(G' - \{u_2, \ldots, u_t\}, U \cup \{u_2, \ldots, u_t\}, F \cup \{v\}, R)$. In the first subproblem the measure decreases by 1, and in the second it decreases by t. We get the branching vector $(1, t)$ with $t \geq 2$.

Case 3.1.3: $v \notin F$, $v \notin S$, and $N(v) \cap F = \emptyset$.

Since we cannot apply Rule C, v has at least one neighbor belonging to S.

If $d(v) = 2$, let u_1 and u_2 be the neighbors of v. Since u_1 or u_2 belongs to S, we know that at most two vertices from $\{v, u_1, u_2\}$ can be added to F. Consequently, this case is identical to the subcase of Case 3.1.1 handling $d(v) = 2$. We branch into the same subproblems and we obtain $(3, 3, 3, 3)$ as the branching vector.

If $d(v) = 3$, let u_1, u_2, u_3 be the neighbors of v. Assume without loss of generality that $u_1 \in S$. This case is very similar to the subcase of Case 1 handling $d(v) = 3$, but now we branch on u_1 instead of v. If u_1 is added to F then at most one of v, u_2, u_3 can be added to F. If u_1 is deleted then either v is added to F or v is also deleted. If v is also deleted then both u_2 and u_3 must be added to F, by Observation 1. Consequently, we branch into the following subproblems, which cover all possibilities, and we obtain $(4, 4, 4, 4, 2, 4)$ as the branching vector:

- Vertices u_2 and u_3 are deleted from G' and added to U; vertices u_1 and v are added to F: the decrease is 4.
- Vertices v and u_3 are deleted from G' and added to U; vertices u_1 and u_2 are added to F: the decrease is 4.
- Vertices v and u_2 are deleted from G' and added to U; vertices u_1 and u_3 are added to F: the decrease is 4.
- Vertices v, u_2, and u_3 are deleted from G' and added to U; vertex u_1 is added to F: the decrease is 4.
- Vertex u_1 is deleted from G' and added to U; vertex v is added to F: the decrease is 2.
- Vertices u_1 and v are deleted from G' and added to U; vertices u_2 and u_3 are added to F: the decrease is 4.

If $t = d(v) \geq 4$, then let $u_1, u_2, \ldots, u_t$ be the neighbors of v in G', and assume without loss of generality that $u_1 \in S$. We will branch on the two possibilities of adding u_1 to F and deleting u_1. If we add u_1 to F then we can add at most one other vertex of $N[v]$ to F and all others must be deleted. Consequently, we branch into the following subproblems:

– Vertex u_1 is deleted from G' and added to U; nothing else changes: the decrease in the measure is 1.
– Vertex u_1 is added to F; vertices $v, u_2, \ldots, u_t$ are deleted from G' and added to U: the decrease in the measure is $t + 1$.
– Vertices u_1 and v are added to F; all other neighbors of v are deleted from G' and added to U: the decrease is $t + 1$.
– Vertices u_1 and u_2 are added to F; v and all other neighbors of v are deleted from G' and added to U: the decrease is $t + 1$.
– The last step above is repeated with each of the neighbors $u_3, \ldots, u_t$ of v instead of u_2: the decrease is $t + 1$ in each of these $t - 2$ additional cases.

The branching vector is $(1, t+1, t+1, \ldots, t+1)$, where the term $t + 1$ appears $t + 1$ times, with $t \geq 4$.

Case 3.1.4: $v \notin F$, $v \notin S$, and $N(v) \cap F \neq \emptyset$.

As we cannot apply Rule C, $N(v) \cap S \neq \emptyset$. Suppose that $|N(v) \cap F| \geq 2$. If there is a vertex $u \in (N(v) \setminus F) \cap S$, then Rule A can be applied for u. Consequently, there is a vertex $u \in N(v) \cap F \cap S$, but then Rule A can be applied for v. It means that v has exactly one neighbor u in F. We take action depending on whether or not u belongs to S:

If $u \in S$, then at most one more vertex from $N[v]$ can be added to F, and all others must be deleted from G' and added to U. We get $t = d(v)$ subproblems in each of which a vertex of $N[v] \setminus \{u\}$ is added to F and all others are deleted from G' and added to U. Observe that we do not get a subproblem where all vertices of $N[v] \setminus \{u\}$ are deleted from G', due to Observation 1. Thus we get $(t, \ldots, t)$ as the branching vector, where the term t is repeated t times, and $t \geq 2$.

If $u \notin S$, then we know that v has another neighbor $w \in S$. We branch into two subproblems resulting from adding w to F or deleting w from G'. If we add w to F, then since u is also in F, no other vertex from $N[v]$ can be added to F and hence they must all be deleted from G' and added to U. We get a subproblem in which the measure decreases by $t = d(v)$. In the other subproblem we simply delete w from G' and add it to U; the decrease is 1. Hence we get $(1, t)$ as the branching vector for this case, where $t \geq 2$.

3.2 The chosen simplicial vertex v belongs to F

Case 3.2.1: $v \in F$ and $v \in S$.

Because $G[F]$ has no S-cycles, $|N(v) \cap F| \leq 1$. If $N(v) \cap F \neq \emptyset$, then Rule A can be applied for the vertices $N(v) \setminus F$. It follows that $N(v) \cap F = \emptyset$. Since $v \in S$ and $v \in F$, at most one vertex of $N(v)$ can be added to F, regardless of how many of these are in S.

If $d(v) = 2$ then let u and w be the two neighbors of v. We branch on the two possibilities of either adding u to the S-forest F or adding u to the subset feedback vertex set U. In the latter subproblem we delete u from G' and add it to U; the decrease is 1. In the first subproblem, we add u to F, and consequently we must delete w from G' and add it to U; the decrease is 2. We get $(1, 2)$ as the branching vector.

If $t = d(v) \geq 3$ then we branch into the possibilities of adding exactly one vertex of $N(v)$ to F and deleting all others from G', or deleting all vertices of $N(v)$ from G'. We get t subproblems in which one vertex is added to F and all other vertices of $N(v)$ are deleted from G' and added to U, and one subproblem in which all vertices of $N(v)$ are deleted from G' and added to U. In each of these $t+1$ subproblems the decrease is t. Hence we get $(t, t, t, \ldots, t)$ as the branching vector, where the term t is repeated $t + 1$ times, and $t \geq 3$.

Case 3.2.2: $v \in F$ and $v \notin S$.

Suppose that $N(v) \cap F \neq \emptyset$. If a neighbor u of v is both in F and in S, then all other neighbors of v are undecided, since $G[F]$ has no S-cycles. Then we can apply Rule A for these neighbors of v. If there is $u \in (N(v) \cap F) \setminus S$, then Rule A can be applied for all $w \in N(v) \cap S$. It means that $N(v) \cap S = \emptyset$, but in this case we can apply Rule C. Therefore, $N(v) \cap F = \emptyset$. Because we cannot apply Rule C, v has at least one neighbor that is undecided and belongs to S.

Recall that $t = d(v) \geq 2$, and let $u_1, u_2, \ldots, u_t$ be the neighbors of v, and assume without loss of generality that $u_1 \in S$. We branch into the two possibilities of either deleting u_1 from G' and adding it to U, or adding u_1 to F. In the latter case, no other neighbor of $N(v)$ can be added to F, since they all form S-cycles with v and u_1, and hence they must all be deleted from G' and added to U. We get one subproblem where the decrease is 1, and one subproblem where the decrease is t. This gives us the branching vector $(1, t)$ with $t \geq 2$.

The description of the algorithm is now complete. The correctness of the algorithm follows from Invariant 1, Lemma 1, Observations 1, 2, and the arguments given for each case, observing that we have taken care of all possible cases. In the next section, we analyze the running time.

3.3 Running time analysis

In each of the branching rules, the measure decreases as described, and in each of the reduction rules, either the measure decreases or at least one vertex of F is deleted from G'. When all vertices of G' are either in U or in F, then the recurrence stops. At this point we need to check whether F is a maximal S-forest of G. This can easily be done in polynomial time; F is an S-forest if and only if every vertex of $S \cap F$ is incident in G to edges that are bridges. Maximality is also easy to check since if a subset X of $V \setminus F$ can be added to F to obtain a larger S-forest, then also a single vertex of X can be added, so we can repeatedly check possible extensions by single vertices. Consequently, the running time will be upper bounded by the number of leaves in the search tree.

For the analysis of the number of leaves $T(n)$ in the search tree, we use standard terminology [8]. In particular, a branching vector $(c_1, c_2, \ldots, c_t)$ results in the recurrence $T(n) \leq T(n-c_1)+T(n-c_2)+\ldots+T(n-c_t)$. In this case $T(n) = O^*(\alpha^n)$, where α is the unique positive real root of $x^n - x^{n-c_1} - \ldots - x^{n-c_t} = 0$ [8], and the O^*-notation suppresses polynomial factors. The number α is called the *branching number* of this branching vector. It is common to round α to the fourth digit after the decimal point. By rounding the last digit up, we can use O-notation instead of O^*-notation [8]. As different branching vectors are involved at different steps of our algorithm, the branching vector with the highest branching number gives an upper bound on $T(n)$.

We now list the branching vectors that have appeared during the description of the algorithm, in the order of first appearence. We give the branching number for each of them; however we do not include here the explicit calculations.

- $(3, 3, 3, 3)$: the branching number is ≈ 1.5875.
- $(4, 4, 4, 4, 2, 4)$: the branching number is ≈ 1.6708.
- $(1, t, t, t, t, \ldots, t)$, where the term t appears t times, and $t \geq 5$: $(1, 5, 5, 5, 5, 5)$ gives the maximum branching number for this vector, which is ≈ 1.6595.
- $(1, t)$, $t \geq 2$: $(1, 2)$ gives the maximum branching number for this branching vector, which is ≈ 1.6181.
- $(t, \ldots, t)$, where the term t is repeated t times, and $t \geq 2$: $(3, 3, 3)$ gives the maximum branching number for this vector, which is ≈ 1.4423.
- $(t, t, \ldots, t)$, where the term t is repeated $t+1$ times, and $t \geq 3$: $(3, 3, 3, 3)$ gives the maximum branching number for this vector, which is ≈ 1.5875.
- $(1, 2)$: the branching number is ≈ 1.6181.

The largest branching number is 1.6708, and it is obtained for $(4, 4, 4, 4, 2, 4)$. Thus the running time of our algorithm is $O(1.6708^n)$.

4 Concluding remarks

As mentioned earlier, there are chordal graphs with $10^{n/5} \approx 1.5848$ minimal sfvs. We have shown that the maximum number of minimal sfvs in chordal graphs is $O(1.6708^n)$. Could it be that the lower bound is also an upper bound or are there chordal graphs with more than $10^{n/5}$ minimal sfvs? Is there an algorithm for Subset Feeback Vertex Set on chordal graphs with running time $O(c^n)$ such that $c < 1.6707^n$?

The lower bound on the maximum number of minimal sfvs of a split graph is $3^{n/3}$ [7], and it is obtained when S is equal to the independent set. Is there a better upper bound for split graphs than for chordal graphs? Does Subset Feeback Vertex Set admit a faster solution on split graphs than on chordal graphs?

We conclude by asking whether all minimal sfvs can be enumerated in time that is polynomial in the number of minimal sfvs. Such an algorithm is known for enumerating minimal fvs in general graphs [17]. It would be very interesting to have such an algorithm for sfvs, even on chordal graphs or split graphs.

References

1. D. G. Corneil and J. Fonlupt. The complexity of generalized clique covering. Disc. Appl. Math., 22:109–118 (1988/1989).
2. J.-F. Couturier, P. Heggernes, P. van 't Hof, Y. Villanger. Maximum number of minimal feedback vertex sets in chordal graphs and cographs. In Proceedings. of COCOON 2012, LNCS, to appear.
3. M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. Subset feedback vertex set is fixed parameter tractable. In Proceedings of ICALP 2011, LNCS 6755:449–461 (2011).
4. G. Even, J. Naor, and L. Zosin. An 8-approximation algorithm for the subset feedback vertex set problem. SIAM J. Comput., 30(4):1231–1252, 2000.
5. F. V. Fomin, S. Gaspers, A. V. Pyatkin, and I. Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. Algorithmica 52(2): 293–307 (2008).
6. F. V. Fomin, F. Grandoni, A. V. Pyatkin, and A. A. Stepanov. Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. ACM Trans. Algorithms 5(1): (2008).
7. F. V. Fomin, P. Heggernes, D. Kratsch, C. Papadopoulos, and Y. Villanger. Enumerating minimal subset feedback vertex sets. In Proceedings of WADS, LNCS 6844: 399–410 (2011).
8. F. V. Fomin and D. Kratsch. Exact Exponential Algorithms. Springer, Texts in Theoretical Computer Science (2010).
9. F. V. Fomin and Y. Villanger. Finding induced subgraphs via minimal triangulations. In Proceedings of STACS 2010: 383–394 (2010).
10. J. A. George and J. W. H. Liu. Computer Solution of Large Sparse Positive Definite Systems. Prentice-Hall Inc. (1981)
11. M. R. Garey and D. S. Johnson. Computers and Intractability. Freeman and Co. (1978).
12. M. C. Golumbic. Algorithmic Graph Theory and Perfect Graphs. Annals of Disc. Math. 57, Elsevier (2004).
13. S. Gaspers and M. Mnich. On feedback vertex sets in tournaments. In Proceedings ESA 2010, LNCS 6346:267–277 (2010).
14. M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. Enumeration of Minimal Dominating Sets and Variants. In Proceedings of FCT 2011, LNCS 6914:298–309 (2011).
15. D. Kratsch, H. Müller, and I. Todinca. Feedback vertex set on AT-free graphs. Disc. Appl. Math., 156: 1936–1947 (2008).
16. J. W. Moon and L. Moser. On cliques in graphs. Israel J. Math. 3: 23–28 (1965).
17. B. Schwikowski and E. Speckenmeyer. On enumerating all minimal solutions of feedback problems. Disc. Appl. Math., 117:253–265 (2002).
18. C. Semple and M. Steel. Phylogenetics. Oxford lecture series in mathematics and its applications (2003).
19. J. P. Spinrad. Efficient graph representations. AMS, Fields Institute Monograph Series 19 (2003).