Internship report

Tight complexity bounds for transversal problems parameterized by the width of a given decomposition

Hugo JACOB
M1 student in Computer Science at ENS Paris-Saclay

Supervised by Marcin Pilipczuk
Co-supervised by Thomas Bellitto and Oscar Defrain
Hosted by the University of Warsaw

February-June 2021

Tight complexity bounds for transversal problems parameterized by the width of a given decomposition

Hugo Jacob, supervised by Marcin Pilipczuk, MIMUW

General context

A meta-theorem of Courcelle, shows the tractability of problems that can be expressed by second order formulae on graphs, parameterized by treewidth or clique-width. However, the obtained complexity is often far from being tight. The optimal complexity of several classical problems under standard assumptions has been studied. One of them is FEEDBACK VERTEX SET which has raised particular interest with new techniques providing a better complexity than what was thought to be possible at first.

Studied problem

A recent publication which studied problems derived from FEEDBACK VERTEX SET parameterized by treewidth showed that they could not be solved with the same complexity under standard assumptions. Its authors left some open questions : finding an algorithm to match their lower bound for one of the problems and generalizing their results with algorithms for the parameter clique-width. The goal of this internship was to tackle these questions. Answering them positively strengthens the lower bound by providing a tight complexity analysis. Translating results on treewidth to clique-width also participates to a better understanding of this less studied parameter.

Contribution

The questions are answered by first making some structural observations, and then refining and adapting previous techniques to the problems at hand. The contribution is mainly of a technical nature. One of the questions is answered in a more general context, providing an improvement on previously known algorithms.

Arguments in favor of correctness

All algorithms are provided with complete proofs. This contribution was also submitted and accepted to IPEC 2021.

Assessment and perspectives

Although the approach for our algorithms parameterized by treewidth is rather general, we only provide a first step towards generalizing the results to clique-width. This is easily explained by the fact that designing an algorithm for clique-width is harder. Completing the generalization of the results to clique-width is an open problem. It should be possible to extend our approach to more technical variants but will require more work.

Contents

1. Introduction

Treewidth is one of the most successful graph width notions, clique-width is another graph width notion that can capture simple dense graphs unlike treewidth, both correspond to the width of a graph decomposition (we denote them by **tw** and **cw**, refer to Subsection 2.3 for definitions). Their algorithmic applications are illustrated by Courcelle's theorems [Cou90, CMR00] stating that any problem expressible by a formula in monadic second order logic with quantification on vertex sets and edge sets (resp. quantification on vertex sets), can be solved in linear time on graphs of bounded treewidth (resp. clique-width). These theorems are very general but the algorithms they construct are often not optimal in complexity: the constants depending on the bound on the width are usually towers of exponentials. This motivated work on finding algorithms with optimal complexity (e.g. [Pil11, BCKN15, CMPP17, BKN+18, CKN18, BST19, SdSS20, BST20a, BST20b, BST20c]). After many simple algorithms were shown to be optimal, it came as a surprise when a randomized algorithm for problem Feedback Vertex Set using the "Cut&Count" method [CNP+11], achieved a running time of $3^{\mathbf{tw}} \cdot k^{\mathcal{O}(1)} \cdot n$ beating the natural algorithm running in time $2^{\mathcal{O}(\mathbf{tw} \log \mathbf{tw})} \cdot n^{\mathcal{O}(1)}$. A deterministic algorithm using the "rank-based" method [BCKN15], was later found, also running in time $2^{\mathcal{O}(\mathbf{tw})} \cdot n^{\mathcal{O}(1)}$. Feedback Vertex Set (FVS) is a problem where given a graph, we are required to find a minimum set of vertices that hit all cycles.

In a recent publication [BBBK20], lower bounds are given for problems that are closely related to Feedback Vertex Set, excluding algorithms running in time $2^{o(\mathbf{tw} \log \mathbf{tw})} \cdot n^{\mathcal{O}(1)}$ under a common assumption. The variant of FVS where we want to hit even cycles or odd cycles are called Even Cycle Transversal (ECT) and Odd Cycle Transversal (OCT) respectively. If we further restrict the cycles to hit by fixing a set of vertices S and asking to hit cycles that contain a vertex in S, we obtain variants Subset Feedback Vertex Set (SFVS), Subset Odd Cycle Transversal (SOCT), and Subset Even Cycle Transversal (SECT). If instead of cycles we hit paths between a specific set of vertices T, we obtain Node Multiway Cut (NMwC) which can be reduced to weighted SFVS by connecting a vertex of S to vertices of T. Given lower bounds include SFVS, SOCT, ECT and NMwC. The following open questions were expressed in a live talk at IPEC 2020 and are the topic of this internship:

- Is there an algorithm for Even Cycle Transversal parameterized by treewidth matching the given lower bound?
- Is there an algorithm for Node Multiway Cut or Subset Feedback Vertex Set parameterized by clique-width matching the given lower bound?
- What running time can we achieve for Odd Cycle Transversal parameterized by clique-width?

We provide a positive answer to all of these questions. For the first question, we provide algorithms for ECT and SECT matching the lower bounds, and also apply our method to SFVS and SOCT, improving on the complexity of the algorithms given in [BBBK20]. For the second question, we give a simple algorithm for NMwC and a more

convoluted algorithm for SFVS, both matching lower bounds. For the third question, we provide an algorithm for OCT and a reduction to argue that our algorithm is likely to be optimal. In Section 2, we introduce some notions of parameterized complexity, the graph decompositions used by the algorithms and the studied problems. In Section 3, we give an overview of the contribution, leaving detailed proofs to the appendix.

2. Preliminaries

2.1. Parameterized Complexity

Expressing complexity of problems only in terms of the size of the input can prove to be quite inadequate in some cases, there are big instances that are easy and small instances that are hard. For this reason, we add parameters that provide further information on the instance to be solved. Consider the classical NP-complete problem Vertex Cover, where given a graph, one looks for a set of vertices called *vertex cover* of size at most k such that each edge is incident to at least one of these vertices. Suppose that you want to solve instances with $k = 3$, while you might think that solving your instances cannot be done in a reasonable amount of time because of the NP-hardness of Vertex Cover, there is a simple algorithm running in time $2^k \cdot n^{\mathcal{O}(1)}$. Hence, your instances are solvable in polynomial time and, even better, solving them will take a reasonable amount of time. This example shows that some additional information on instances of NP-hard problems have a serious impact on their hardness, motivating a more precise classification of these problems via parameters.

Definition 1 *A* parameterized problem *is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed finite alphabet. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the* parameter. *The* size *of the instance is $|x| + k$, as if the parameter was written in unary.*

Definition 2 *A parameterized problem L is called* fixed-parameter tractable *(FPT) if there exists an algorithm $\mathcal{A}$ (called fixed-parameter algorithm), a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant c such that, "$(x, k) \in L$" can be decided by $\mathcal{A}$ in time bounded by $f(k) \cdot |(x, k)|^c$. The class of FPT problems is called* FPT.

It should be clear that P $\subseteq$ FPT. We saw the example of Vertex Cover which is in NP and in FPT parameterized by solution size. Other classical NP-complete problems Independent Set and Dominating Set, are believed not to be in FPT (more details later).

Intuitively, if a NP-hard problem is FPT for some parameterization, we can consider that the parameter is a good way to quantify the hardness of some instance. For Vertex Cover, it is easy to find a small vertex cover and hard to find a big vertex cover. A very natural parameter is the solution size and when the parameter is omitted it should be this parameter.

For FPT algorithms, depending on the type of instance we are trying to solve, it can be interesting to look for the best possible f or the best possible c.

Parameterized problems can be classified using reductions, similarly to classical problems. However, the crucial difference is that the parameter growth must be controlled.

Definition 3 *Given two parameterized problems A, B over the same alphabet, a* parameterized reduction *from A to B is an algorithm $\mathcal{A}$ that, given an instance (x, k) of A, outputs an instance (x', k') of B such that:*

- *(x, k) is a yes-instance of A if and only if (x', k') is a yes-instance of B*
- *$k' \leq g(k)$ for some computable function g*
- *$\mathcal{A}$ is FPT*

If B is in FPT and a *parameterized reduction* from A to B is given, we can deduce that A is also FPT.

Note that the simple reduction from Independent Set to Vertex Cover by taking the complement of the solution is not a parameterized reduction because a solution of size k is mapped to a solution of size $n-k$. On the contrary, Independent Set and Clique can be reduced to one another because the simple reduction consists of considering the complemented graph and preserves the solution size.

We now informally explain why Independent Set and Dominating Set are thought not to be in FPT. The W hierarchy is based on *parameterized reduction*, if a problem is in W[t] it means that it can be reduced to solving a formula with nested "big" quantifications of depth at most t with the number of satisfied variables as the parameter. Independent Set is W[1]-complete and Dominating Set is W[2]-complete, while FPT = W[0]. It is commonly assumed that FPT $\neq$ W[1], hence to prove that a problem is believed not to be in FPT, W[1]-hardness results are given. Note that this assumption is stronger than P $\neq$ NP because W[1] $\subseteq$ NP.

A more precise study of problems that are in FPT requires stronger assumptions. We use some conjectures on the optimal complexity for this problem:

k-SAT **Input:** A CNF formula φ with at most k variables per clause **Parameter:** k **Question:** Is φ satisfiable ?

Let s_k be the infimum over positive reals ε such that there exists an algorithm running in time $\mathcal{O}(2^{\varepsilon n})$ to solve k-SAT, where n is the number of variables.

The **Exponential Time Hypothesis** (ETH) states that $s_3 > 0$. This implies that there is no $2^{o(n)}$ time algorithm for 3-SAT. By reducing k-SAT to some problem, we can conclude that there is no $2^{o(f(n))}$ time algorithm for it under the ETH. The ETH implies FPT $\neq$ W[1], hence P $\neq$ NP.

The **Strong Exponential Time Hypothesis** (SETH) states that $\lim_{k\to+\infty} s_k = 1$. This means that for some arbitrary formula, there is no algorithm significantly better than testing every possible assignment. By reducing k-SAT to some problem with the new parameter bounded independently of the number of clauses, we can prove that a running time of the form $\mathcal{O}\left(c^k|x|^{\mathcal{O}(1)}\right)$ has optimal c under the SETH. While this assumption is considered less likely by part of the computational complexity community, note that this mainly allows to express neatly the hardness result obtained from a reduction but is not used in the reduction. Furthermore, it indicates that trying to beat the lower bound is at least as hard as disproving the SETH.

2.2. Graph theory terminology

We call *walk* a sequence of vertices such that consecutive vertices are adjacent, a walk without repetition of vertices is called (simple) *path*. A walk with the first vertex equal to the last vertex is called a *closed* walk, if it is the only vertex repetition, it is called a *cycle*. Given a graph $G = (V, E)$ and a subset of vertices $X \subseteq V$, $G[X]$ denotes the *induced subgraph* of X. For A a set of edges, $G + A$ denotes the graph $G' = (V, E \cup A)$. For G, G' two graphs, their union $G \cup G'$ is the graph $(V(G) \cup V(G'), E(G) \cup E(G'))$.

We will sometimes need multigraphs, which are tuples (V, E) where V is a set of vertices and E is a multiset of pairs of vertices, i.e. it can have repeated edges.

When a subset of vertices S is fixed, we call *S-path* a path that contains a vertex of S, *S-cycle* a cycle that contains a vertex of S.

A 2-connected component of a (multi)graph is a maximal induced subgraph such that more than one vertex must be removed for it to be disconnected. We call *nontrivial* a 2-connected component that contains a cycle, excluding the degenerate cases of the isolated vertex and the bridge (2 vertices connected by a single edge). We call a (multi)graph nontrivial 2-connected if it consists of a single 2-connected component.

2.3. Graph decompositions

One approach to solving hard problems on graphs that have a nice structure, is to first compute a decomposition of the graph and then apply an efficient algorithm on the decomposition. The decompositions that we will consider are well suited to the design of dynamic programming.

Definition 4 *A* tree decomposition *of a graph $G = (V, E)$ is a pair $(T, \mathcal{X})$ where T is a tree and $\mathcal{X} = \{X_t\}_{t\in T}$ is a set of* bags *such that :*

- *bags contain vertices (elements of V)*
- *for $v \in V$, the bags that contain v form an induced subtree of T (they are connected)*
- *every vertex $v \in V$ is contained in at least one bag*
- *for every edge $uv \in E$, there is a bag X_t that contains u and v.*

The width of a tree decomposition is defined as $\max_{t \in T}\{|X_t| - 1\}$.

The treewidth of G *is the minimal width over all possible tree decompositions of* G*, we will denote it by* $\boldsymbol{tw}(G)$.

The pathwidth of G *is the minimal width over all possible tree decompositions of* G *where the tree is a path, we will denote it by* $\boldsymbol{pw}(G)$.

Definition 5 (Nice tree decomposition) *A nice tree decomposition of a graph* G *is a rooted tree decomposition* $(T, \{X_t\}_{t \in V(T)})$ *such that:*

- *the root and leaves of* T *have empty bags; and*
- *other nodes are of one of the following types:*
  - ***Introduce vertex node****: a node* t *with exactly one child* t' *such that* $X_t = X_{t'} \cup \{v\}$ *with* $v \notin X_{t'}$*. We say that* v *is introduced at* t*;*
  - ***Forget vertex node****: a node* t *with only one child* t' *such that* $X_t = X_{t'} \setminus \{v\}$ *with* $v \in X_{t'}$*. We say that* v *is forgotten at* t*; and*
  - ***Join node****: a node* t *with two children* t_1*,* t_2 *such that* $X_t = X_{t_1} = X_{t_2}$.

For each node t *of the decomposition, we define a* partial *graph* $G_t = G\left[\bigcup_{s \leq t} X_s\right] - E(G[X_t])$.

Note that edges of partial graphs appear at forget vertex nodes and that they correspond to adding edges between the forgotten vertex and its neighbours.

From a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G of width k, a nice tree decomposition of width k with $\mathcal{O}(k|V(G)|)$ nodes can be computed in time $\mathcal{O}(k^2 \cdot \max(|V(T)|, |V(G)|))$ [Klo94].

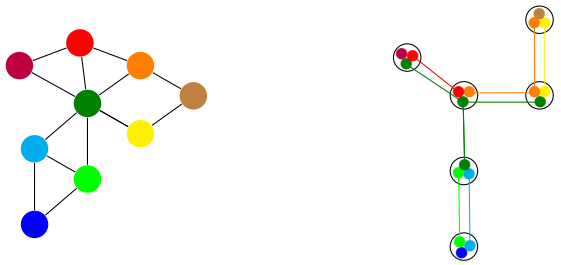


Figure 1.: A graph and one of its tree decompositions

The state of the art for FPT computation of a tree decomposition of size k (or a certificate that the input has greater treewidth than k) is an algorithm running in time $k^{\mathcal{O}(k^3)} \cdot n$ [Bod96]. There is also a very recent 2-approximation algorithm running in time $2^{\mathcal{O}(k)} \cdot n$ [Kor21], it is based on previous approximation algorithms and computes an improved tree decomposition by running dynamic programming on an initial tree decomposition.

Since the number of edges of a graph is at most $\mathbf{tw} \cdot n$, this decomposition is only interesting for sparse graphs, this contrasts with clique-width which captures simple dense graphs.

To define clique-width and k-expressions, we first define labeled graphs and operations on them.

Definition 6 *A* k-labeled graph *is a triple* $G = (V, E, \gamma)$ *where* (V, E) *is a graph and* $\gamma : V \to \{1, \ldots, k\}$ *is a function called* labeling function.

$V(G), E(G)$ and $\gamma(G)$ denote the set of vertices, set of edges and labeling function of labeled graph G.

Given a k-labeled graph $G = (V, E, \gamma)$, we refer to $\gamma^{-1}(\{i\})$ by vertices of label i or even simply label i.

We will only consider undirected graphs in the following (but this notion can be extended to directed graphs).

Definition 7 *For two k-labeled graphs G_1, G_2, $G_1 \oplus G_2$ is their disjoint union.*

For a given k-labeled graph G, $\rho_{i \to j}(G)$ is the labeled graph obtained from G when changing the label of vertices labeled i to j.

For a given k-labeled graph G, $\eta_{i \times j}(G)$ is the labeled graph obtained from G by adding edges between vertices labeled i and vertices labeled j.

The graph consisting of a single vertex v labeled i is simply denoted $i(v)$.

k-expressions are terms that define a labeled graph using these operations.

Definition 8 *The* clique-width *of a graph is the minimal k such that it can be represented by a k-expression.*

Definition 9 *A linearized version of clique-width called* linear clique-width *corresponds to the minimum width of k-expressions such that disjoint union operations must contain a single vertex on one side.*

Note that it may happen that an edge having its endpoints in two different labels i and j is already present in the graph G before performing the join $\eta_{i \times j}(G)$. Despite that the edge is not produced twice, the existence of such a situation may be problematic in our algorithms when we only consider a compact representation of G. To circumvent this problem, we can assume that every edge of a graph appears at most once in the join of our given k-expressions. More precisely, when performing a join operation $\eta_{i \times j}(G)$, we can assume that none of the edges in G has its endpoints in i and j, respectively. An expression with such property can be computed in time $\mathcal{O}(k^2 \cdot n)$ from a given arbitrary expression of size n [CO00].

The following inequality stands between treewidth and clique-width :

$$\mathbf{cw}(G) \leq 3 \cdot 2^{\mathbf{tw}(G)-1}$$

Hence, the class of graphs of bounded treewidth is included in the class of graphs of bounded clique-width. Graphs of bounded clique-width can be dense unlike graphs of bounded treewidth, a simple example is the complete graph on n vertices, it has treewidth $n-1$ (a single bag containing all vertices) but linear clique-width 2 (add vertices one by one by joining a label containing the new vertex to a label containing other vertices and then relabeling the new vertex).

Pathwidth can be compared to the other considered parameters with $\mathbf{tw}(G) \leq \mathbf{pw}(G)$ (obvious from definition) and $\mathbf{cw}(G) \leq \mathbf{lcw}(G) \leq \mathbf{pw}(G) + 2$ (use one label per vertex in the bag and an additional label for forgotten vertices). This means that a complexity lower bound involving pathwidth also stands for treewidth and clique-width.

There is no known FPT algorithm to obtain an optimal k-expression. The best approximation algorithms running in FPT time provide only expressions with exponentially too many labels [Oum05]. Graphs of clique-width at most 3 (called distance hereditary graphs) can be recognized in polynomial time [CHL+12]. Recognition of graphs of clique-width at most k for $k > 3$ is an open problem, hence for algorithms parameterized by clique-width, we assume that the graph is given with a k-expression describing it.

The monadic second order logic of graphs consists of formulas on vertices, edges, sets of edges and sets of vertices with predicates for equality, membership, adjacency and incidence testing. MSO_2 stands for the Monadic Second Order logic of graphs in which quantification over sets of vertices and sets of edges is possible, in MSO_1 only quantification over sets of vertices is allowed. A common extension called counting monadic second order logic consists of adding modular counting predicates.

A theorem by Bruno Courcelle shows that a graph property defined by a formula φ in MSO_2 can be checked in time $f(|\varphi|, \mathbf{tw}) \cdot n$ on graphs of treewidth $\mathbf{tw}$ [Cou90]. Similarly, a graph property defined by a formula φ in MSO_1 can be checked in time $g(|\varphi|, \mathbf{cw}) \cdot n$ on graphs of clique-width $\mathbf{cw}$ [CMR00]. The original hypothesis that a decomposition is given as an input is not necessary due to approximation algorithms for both decompositions. Courcelle's theorems give a very general result, however the running time of algorithms constructed by the proof is often irrelevant. Their result is only interesting from a theoretic point of view.

2.4. Studied problems and relatives

Feedback Vertex Set
Input: A graph G and an integer k
Question: Is there a set of at most k vertices hitting every cycle in G ?

As discussed earlier, this problem is known to have an algorithm running in time $2^{\mathcal{O}(\mathbf{tw})} n^{\mathcal{O}(1)}$ [BCKN15]. It is also known to have an algorithm running in time

$2^{\mathcal{O}(\mathbf{cw})}n^{\mathcal{O}(1)}$ when a clique-width expression is provided [BK19]. Note that the variant of this problem in the context of directed graphs (or digraphs) was shown to be solvable in time $2^{\mathcal{O}(\mathbf{tw}\log\mathbf{tw})}n^{\mathcal{O}(1)}$ and not faster (the parameter is the treewidth of the underlying undirected graph of the input digraph) [BKN+18].

Odd Cycle Transversal
Input: A graph G and an integer k
Question: Is there a set of at most k vertices hitting every odd cycle in G ?

This is the easiest variant of FVS, it can be solved in time $3^{\mathbf{tw}}n^{\mathcal{O}(1)}$ with a given tree decomposition and not faster [LMS11]. We show a similar result for clique-width in 3.4.

The following variants of FVS were all shown to be significantly harder, they admit no algorithm running in time $2^{o(\mathbf{pw}\log\mathbf{pw})}n^{\mathcal{O}(1)}$, assuming the ETH.

Even Cycle Transversal
Input: A graph G and an integer k
Question: Is there a set of at most k vertices hitting every even cycle in G ?

Node Multiway Cut
Input: A graph G, a set of terminals $T(G) \subseteq V(G)$, and an integer k
Question: Is there a set of at most k non-terminal vertices hitting every path between a pair of terminals ?

Subset Feedback Vertex Set
Input: A graph G, a subset of vertices $S \subseteq V(G)$ and an integer k
Question: Is there a set of at most k vertices hitting every cycle containing a vertex in S ?

Subset Odd Cycle Transversal
Input: A graph G, a subset of vertices $S \subseteq V(G)$ and an integer k
Question: Is there a set of at most k vertices hitting every odd cycle containing a vertex in S ?

Subset Even Cycle Transversal
Input: A graph G, a subset of vertices $S \subseteq V(G)$ and an integer k
Question: Is there a set of at most k vertices hitting every even cycle containing a vertex in S ?

All these problems can be equivalently expressed as finding a set of vertices to delete for the resulting graph to have some properties. Such problems are often called *deletion problems*, and the set of vertices to find is called *deletion set*. We adopt this point of view to develop our algorithms by constructing maximal graphs that preserve the desired properties.

We study these problems in the weighted setting meaning vertex v is given weight

$c(v)$, we extend this notation to subset of vertices with $c(U)$ being the sum of weights of vertices in U.

3. Contribution

In subsection 3.1, we present an algorithm for SFVS, ECT, SOCT, and SECT, parameterized by treewidth, using a common framework to solve the open question in a more general setting. In subsection 3.2, we present an algorithm for NMwC parameterized by clique-width as a first step towards solving the harder SFVS. In subsection 3.3, we present a more involved algorithm for SFVS parameterized by clique-width. In subsection 3.4, we present an algorithm for OCT parameterized by clique-width and a parameterized reduction showing optimality of our OCT algorithm under the SETH.

3.1. Feedback Vertex Set relatives parameterized by treewidth

In [BBBK20], Bergougnoux, Bonnet, Brettell, and Kwon gave a $2^{\Omega(\mathbf{tw}\log\mathbf{tw})}n^{\mathcal{O}(1)}$ lower bound for Subset Feedback Vertex Set, Subset Odd Cycle Transversal, and Even Cycle Transversal under the ETH. We present an algorithm of complexity $2^{\mathcal{O}(\mathbf{tw}\log\mathbf{tw})}n$ for them and Subset Even Cycle Transversal, closing the gap for Even Cycle Transversal and Subset Even Cycle Transversal, and improving on the previous $2^{\mathcal{O}(\mathbf{tw}\log\mathbf{tw})}n^3$ algorithm for Subset Feedback Vertex Set and Subset Odd Cycle Transversal of [BBBK20].

Rather than simply giving an algorithm for just SOCT and SECT to which we can reduce ECT and SFVS, we also show how our method gives less involved algorithms for SFVS and ECT. In order to have a common notation, we will call $\square$-cycles the cycles that have to be hit in the problem and $\square$-cycle-free the graphs that do not contain $\square$-cycles.

Before diving in the explanation of the algorithm, we will discuss how to design a bottom-up dynamic programming algorithm for these problems on a nice tree decomposition. A natural way to proceed is to find a criterion such that the union of two $\square$-cycle-free graphs with k common vertices is also $\square$-cycle-free. To check that no $\square$-cycle is created by the union, for each pair of the common vertices, we need to know what type of paths exist between them (e.g. for ECT we want to know if they are connected by an even path and if they are connected by an odd path) in each of the two graphs. A naive way to do this is to simply have a matrix with the answer to each of these $\mathcal{O}(k^2)$ questions. This would lead to an algorithm running in time $2^{\mathcal{O}(\mathbf{tw}^2)}n$. By representing the graph with an auxiliary forest, we can encode the same information using $\mathcal{O}(k\log k)$ bits, leading to a $2^{\mathcal{O}(\mathbf{tw}\log\mathbf{tw})}n$ algorithm.

We begin by giving a characterisation of nontrivial 2-connected $\square$-cycle-free graphs for each problem. This implies characterisations of $\square$-cycle-free graphs, because it can be decomposed in 2-connected components and then the characterisation can be applied to each component.

Lemma 3.1 *Let G be a nontrivial 2-connected multigraph.*

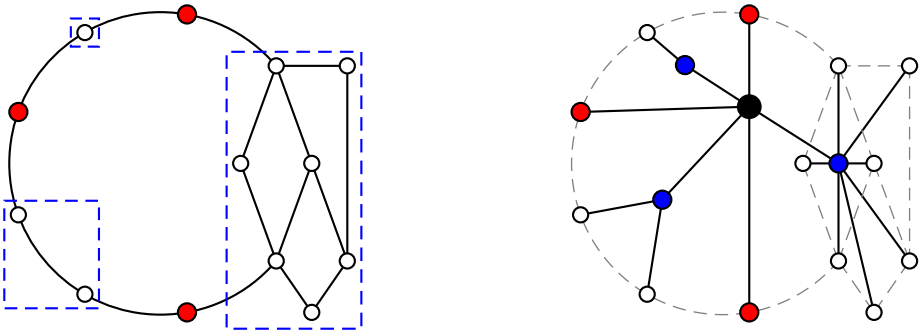
1. *G contains no S-cycle if and only if it contains no S-vertex.*
2. *G contains no even cycle if and only if it is an odd cycle.*
3. *G contains no odd S-cycle if and only if it has one of the following forms:*
   - *G contains no S-vertex and is not bipartite*
   - *G contains no S-vertex and is bipartite*
   - *G contains at least one S-vertex and is bipartite.*
4. *G contains no even S-cycle if and only if it has of one of the following forms:*
   - *G contains no S-vertex and is not bipartite*
   - *G contains no S-vertex and is bipartite*
   - *G contains at least one S-vertex, the connected components of $G - S$ are bipartite, together with S-vertices they form a cycle: each S-vertex has degree 2 and each connected component of $G-S$ has outdegree 2. One S-cycle is odd. We later call* bipartite subcomponents *the connected components of $G - S$. This is illustrated in Figure 2a.*

The first point is immediate, the second was observed in [MRRS12] and the third in [BBBK20]. The last point was not known to us and we provide a proof in appendix (2).

Definition 10 *Given a $\square$-cycle-free graph G, we define its* underlying forest *$F(G)$ as the graph obtained from G by modifying independently each nontrivial 2-connected component C as follows:*

1. *For SFVS, remove edges inside C and add an unlabeled vertex adjacent to all vertices of C.*
2. *For ECT, remove edges inside C and add a vertex adjacent to all vertices of C and label it "odd cycle".*
3. *For SOCT, remove edges inside C and add a vertex adjacent to all vertices of C, label it "bipartite" or "not bipartite" based on the property of C and make it an S-vertex if C contains an S-vertex.*
4. *For SECT, in the two first forms we remove edges inside C and add a vertex adjacent to all vertices of C and label it "bipartite" or "not bipartite" based on the property of C. For the last form, for each bipartite subcomponent in the cycle, we remove its edges, add a vertex labeled "internal bipartite" adjacent to its vertices. Then remove edges of C incident to S, add an S-vertex labeled "odd cycle" adjacent to S-vertices and vertices labeled "internal bipartite". This is illustrated in Figure 2b.*

Observe that, because labeled vertices are only introduced by this underlying forest, to each labeled vertex v, each of them can be associated with a nontrivial 2-connected



(a) An example of graph with no even S-cycle. The vertices of S are depicted in red. The blue boxes denote the bipartite subcomponents.

(b) The underlying forest we build from the graph on Figure 2a. The "internal bipartite" vertices are depicted in blue and the "odd cycle" vertex is black.

Figure 2.: The last form of SECT: "internal bipartite" vertices

component C: the one that resulted in the creation of v. Observe also that for a path P between two unlabeled vertices, if it contains a labeled vertex, then it contains a vertex of its associated component before it on P and another vertex of its associated component after it on P.

Using some reduction rules inspired by [BBBK20], the forest can be reduced to $\mathcal{O}(\mathbf{tw})$ vertices, we denote the set of reduced underlying forests with active vertices (vertices in common when performing unions) X by $F(G, X)$. The idea of the reduced forest is to preserve only the paths between active vertices and to contract these paths as much as possible. The length of paths still has to be maintained, there are two ways of doing this. The simplest to describe is to have edges with weights in $\mathbb{F}_2$, but this can also be done by coloring active vertices, which can give better constants for the size of the reduced forests.

The main technical engine of our algorithms is the following join operation (see 2 for proof).

Lemma 3.2 *There exists a polynomial-time algorithm that, for every pair of $\square$-cycle-free graphs G_1 and G_2 with $V(G_1) \cap V(G_2) = X$, given on input two reduced forests with valid $\mathbb{F}_2$-labelings (F_1, α_1) and (F_2, α_2), with $F_1 \in F(G_1, X)$ and $F_2 \in F(G_2, X)$, decides whether $G_1 \cup G_2$ is $\square$-cycle-free and, in case of a positive answer, computes a reduced forest $F \in F(G_1 \cup G_2, X)$ and, except for the* SFVS *problem, a valid $\mathbb{F}_2$-labeling α.*

We now describe our dynamic programming algorithm on a nice tree decomposition $(T, \{X_t\}_{t \in V(T)})$ of graph G. It consists of a bottom-up computation with states (t, Y, F, α) where $t \in V(T)$, $Y \subseteq X_t$ is the set of undeleted vertices of X_t, F is a labeled forest description with active vertices Y, and α is a $\mathbb{F}_2$-labeling of edges of F. We denote by $d[t, Y, F, \alpha]$ the cell of the table corresponding to this state. We call a state *reachable* if its cell is updated at least once by a transition.

We call a state (t, Y, F, α) *admissible* if there exists $U \subseteq V(G_t)$ such that $Y = X_t \setminus U$, $G_t - U$ is $\square$-cycle-free, F is a forest description of a member of $F(G_t - U, Y)$, α is a valid 2-labeling of F and $d[t, Y, F, \alpha] = c(U)$.

To prove the correctness of the algorithm we will prove that reachable states are admissible and that for each $t \in V(T)$ and $U \subseteq V(G_t)$, if $G_t - U$ is $\square$-cycle-free there exists a state with value at most $c(U)$. The optimal transversal weight will be in $d[r, \varnothing, \varnothing, \varnothing]$ where r is the root of the decomposition.

We now describe the computations for each node t of the nice tree decomposition based on its type.

1. **Leaf node.** We set $d[t, \varnothing, \varnothing, \varnothing] = 0$
2. **Introduce vertex node.** Let t' denote the child node of t and v be the introduced vertex. For each reachable state (t', Y', F', α'), we have two transitions representing the choice of deleting the vertex or not:

$$d[t, Y', F', \alpha'] \leftarrow d[t', Y', F', \alpha'] + c(v)$$

$$d[t', Y' \cup \{v\}, F, \alpha'] \leftarrow d[t', Y', F', \alpha']$$

   where F is obtained from F' by adding an isolated active vertex v.
3. **Forget vertex node.** Let t' denote the child node of t and v be the forgotten vertex. For each reachable state (t', Y', F', α'), if $v \notin Y'$ then the transition is simply:

$$d[t, Y', F', \alpha'] \leftarrow d[t', Y', F', \alpha']$$

   If $v \in Y'$, we perform a join processing of (F', α') and (H, β) where H is the union of a star graph with internal vertex v and leaves $N_G(v) \cap Y'$ and isolated vertices for $Y' \setminus N_G[v]$, and β its edges to 1. If the join processing does not reject and returns $(\widetilde{F}, \widetilde{\alpha})$, we obtain (F, α) from $(\widetilde{F}, \widetilde{\alpha})$ by making v inactive and applying reduction rules, then have transition:

$$d[t, Y' \setminus \{v\}, F, \alpha] \leftarrow d[t', Y', F', \alpha']$$

4. **Join node.** Let t_1 and t_2 denote the two children of t. For each pair of reachable states (t_1, Y, F_1, α_1) and (t_2, Y, F_2, α_2), we perform a join process of (F_1, α_1) and (F_2, α_2). If the join isnt rejected, we obtain (F, α) and have transition $d[t, Y, F, \alpha] \leftarrow d[t_1, Y, F_1, \alpha_1] + d[t_2, Y, F_2, \alpha_2]$.

Lemma 3.3 *All reachable states are admissible.*

Lemma 3.4 *For every node $t \in V(T)$, every $U \subseteq V(G_t)$, if $G_t - U$ is $\square$-cycle-free then there exists F, α such that F is a forest description of a member of $F(G_t - U, X_t \setminus U)$, α is a valid $\mathbb{F}_2$-labeling of F, $(t, X_t \setminus U, F, \alpha)$ is reachable, and $d[t, X_t \setminus U, F, \alpha] \leq c(U)$.*

Lemma 3.5 *The final value of $d[r, \varnothing, \varnothing, \varnothing]$ is the weight of an optimal transversal.*

Proof. By applying Lemma 3.4 with U a $\square$-cycle transversal, because $V(G)$ is always a transversal, we conclude that state $(r, \varnothing, \varnothing, \varnothing)$ is reachable, $d[r, \varnothing, \varnothing, \varnothing] \leq c(U)$. By applying Lemma 3.3 to reachable state $(r, \varnothing, \varnothing, \varnothing)$, there exists U^* a $\square$-cycle transversal such that $d[r, \varnothing, \varnothing, \varnothing] = c(U^*)$. It has optimal weight by Lemma 3.4. $\square$

Theorem 3.1 Subset Feedback Vertex Set, Subset Odd Cycle Transversal, *and* Subset Even Cycle Transversal, *even in the weighted setting, can be solved in time* $2^{\mathcal{O}(k \log k)} \cdot n$ *on* n*-vertex graphs of treewidth* k*.*

Proof. We use an approximation algorithm to compute a tree decomposition of width $\mathcal{O}(k)$ in time $2^{\mathcal{O}(k)} \cdot n$. We have $2^{\mathcal{O}(k \log k)} \cdot n$ states and transitions. Since transitions are computed in time $k^{\mathcal{O}(1)}$, the values of all states are computed in time $2^{\mathcal{O}(k \log k)} \cdot n$. The solution to the problem instance is correctly computed, by Lemma 3.5. $\square$

3.2. Node Multiway Cut parameterized by clique-width

In this problem the first obstacle to deal with is the fact that we can't hope to maintain the internal structure of a each label with $\mathcal{O}(k \log k)$ bits. However, a crucial property is that once a join is performed, the vertices of a label become connected. To solve this problem, we apply a technique called "expectation from the outside" in [BSTV13] that consists in guessing in advance what operation will happen next to a label (this is not obvious from the expression because some joins will not add edges due to vertex deletions) to design a bottom-up dynamic programming algorithm.

A state in our dynamic programming will consist of a near-partition $\mathcal{P}$ of labels according to the expected evolution of their connectivity, where two subsets have a special role : $P_\varnothing$ will denote labels that contain no vertex, P_f will denote labels that will only be joined to empty labels in the future, other subsets $P_1, \ldots, P_k$ represent labels that may form a connected component in the future, together they form $\mathcal{P}^*$; and a function $\phi : [k] \to \{0, 1\}$ indicating if the component contains a terminal.

A state is designed to efficiently represent a graph $\widetilde{G}$ resulting from some vertex deletions on our input graph with just the information that is necessary for the computation. If a label is in P_f, it means that we expect that its vertices will not receive additional incident edges in the rest of the construction. Thus, a label in this situation will only be joined to a label in $P_\varnothing$, which guarantees that no edge is added. For vertices in labels that are in the same P_i, we expect that they will be in the same connected component and *already* consider them as connected. Two nonempty labels are only joined when they are in the same P_i, using this we make sure that the actual connected components of $\widetilde{G}$ are subdivisions of the $\bigcup_{j \in P_i} V_j(\widetilde{G})$ with the addition of some vertices of labels in P_f.

The difficult question of knowing what is accessible from a vertex is circumvented by our guessing strategy, which takes the form of the partitioning of non empty labels into

P_f and $P_1, \ldots, P_k$. When a vertex is in a label of P_f, we know from our guess that the question is irrelevant because we are done adding edges incident to this vertex. When a vertex is in a label of P_i, we know from our guesses that it can only be connected to vertices in labels of P_f and to vertices in labels of P_i.

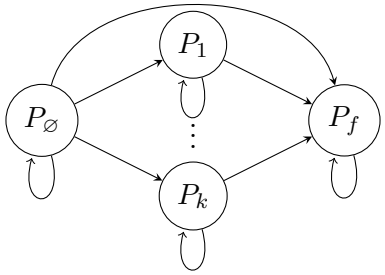


Figure 3.: The automaton shows how labels are allowed to move after a join.

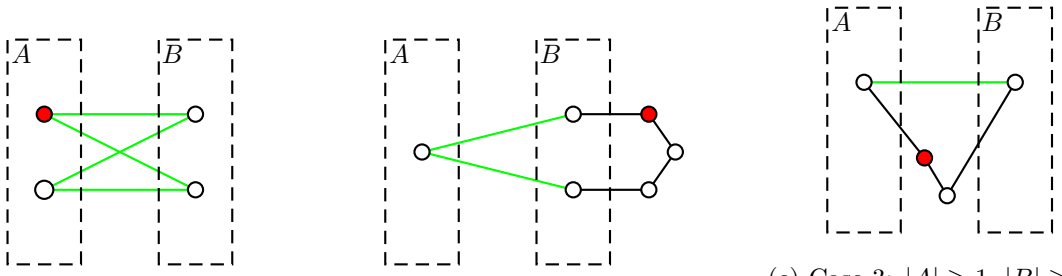
The details of the algorithm are in appendix (2), the correctness proofs are structured similarly to the proofs of the algorithm of Subsection 3.1, to represent the guesses on labels we use an auxiliary graph that contains the graph under construction with additional edges to connect vertices when their labels are in the same set of $\mathcal{P}^*$.

3.3. Subset Feedback Vertex Set parameterized by clique-width

Our algorithm for SFVS on k-expressions will reuse ideas from the two previous subsections. We will make guesses to simplify the information to be kept on our labels as for NMwC. The information about how vertices are connected is more complicated as we have to consider the fact that paths may be S-paths. This is done by maintaining a partially labelled forest as we did in the algorithm on tree decompositions.

The first step to design this algorithm has been to consider the different cases of joins that can create an S-cycle, see Figure 4. There are few cases where the join does not create an S-cycle which motivates a distinction between a finite number of what we call *label states*, that allow a classification of the possible joins based on internal structure of the label.

State $Q_\varnothing$ is assigned to labels that are completely contained in the current deletion set, which means that the label is empty for the graph resulting from the deletion. States Q_1 and Q_1^* are assigned to labels consisting of a single non-S-vertex, or a single S-vertex, respectively. States Q_w and Q_w^* are called *waiting states*: they are assigned to labels for which we have guessed that they will be joined (only once) to a non-S-vertex from a label in state Q_1, or to an S-vertex from a label in state Q_1^*, respectively. State Q_2 is assigned to labels having at least two vertices but no S-vertex: it is assigned to labels for which we have guessed that they will be joined (potentially several times) to either a vertex from a label in state Q_1, or to vertices from a label in state Q_2. These guessing tricks



(a) Case 1: $|A| \geq 2$, $|B| \geq 2$, and $|A \cap S| + |B \cap S| \geq 1$.

(b) Case 2: $|A| \geq 1$, $|B| \geq 2$, and there is an S-path with endpoints in B.

(c) Case 3: $|A| \geq 1$, $|B| \geq 1$, and there is an S-path connecting a vertex of A to one of B.

Figure 4.: The three cases when a join (depicted in green) creates an S-cycle. The figures illustrate the smallest number of vertices of S required. Thus, up to symmetry, the vertices depicted in red have to be in S while the vertices in white may or may not be in S.

can be seen as a form of what is called "expectation from the outside" in [BSTV13]. We point that guessing these joins implies that labels in states Q_w, Q_w^*, Q_2 will eventually be connected. At last, state Q_f is called *final state*: it will contain vertices that will not be joined anymore. To summarize, states in $\mathcal{Q}$ express the following constraints on joins:

- joins with a label in state $Q_\varnothing$ will be ignored;
- no join with a label in state Q_f will be performed;
- labels in state Q_w (resp. Q_w^*) will only be joined with those in state Q_1 (resp. Q_1^*); and
- labels in state Q_2 will never be joined with those in state Q_1^*.

As illustrated by Subfigure 4c, we need to check the properties of a potential path between two labels. To cope with these constraints, we also give a reduced forest with a vertex per label and contracted paths between them. We have to adapt the notion of active vertices to the context of k-expressions, now they are vertices representing labels that are not in state Q_f. This makes a lot of sense: in the context of the nice tree decomposition, no additional incident edges may be added to a vertex once it has been forgotten, and here, thanks to the guessing technique, we are exactly in the same case where we know that there will not be any additional edge.

Theorem 3.2 *Given a k-expression describing graph G,* Subset Feedback Vertex Set *can be solved in time $2^{\mathcal{O}(k \log k)} \cdot n$, where n is the size of the given k-expression.*

The details of the computation and the proof of the algorithm are deferred to the appendix (2).

3.4. Odd Cycle Transversal parameterized by clique-width

To introduce our algorithm for OCT on k-expressions, we first explain the idea behind the known algorithm for tree decompositions, which is optimal under the SETH [LMS11].

It is a dynamic programming algorithm with the following states: for each node t of the nice tree decomposition, each tripartition (U, A, B) of X_t, compute the minimum $|\widetilde{U}|$ for all tripartitions $(\widetilde{U}, \widetilde{A}, \widetilde{B})$ of $V(G_t)$ such that $U = X_t \cap \widetilde{U}$, $A = X_t \cap \widetilde{A}$, $B = X_t \cap \widetilde{B}$, and $\widetilde{A}$ and $\widetilde{B}$ are independent sets of $G_t - \widetilde{U}$. The intuition is that we construct a bipartition for the graph resulting from vertex deletion, ensuring that it is bipartite, meaning it has no odd cycle. Furthermore, the part of the tripartition of the vertices of the partial graph that is not in the current bag has no effect on the constraints on new vertices.

We will simply adapt the same idea to k-expressions. As for the previous algorithms on k-expressions, we have to find a way to manage the fact that labels may contain many vertices. Luckily, in this case it will be quite simple, due to the following fact:

Fact 3.1 *For (A, B) a bipartition of graph labeled graph H, (A, B) is a not a bipartition of $\eta_{i\times j}(H)$ if and only if there is a side of the bipartition with a vertex in each of i and j.*

Proof. Suppose that (A, B) is not a bipartition of $H' = \eta_{i\times j}(H)$ then A or B is not an independent set of H'. Without loss of generality, we assume that A is not an independent set of H', hence there exists $u, v \in A$ adjacent in H'. Since A is an independent set of H, edge uv must be added by the join operation between i and j, so one of u, v is in i and the other is in j.

Conversely, if there is a side of the bipartition with a vertex in each of the joined labels then in H' there is an edge between them and this side is not an independent set anymore, so (A, B) is not a bipartition of H'. □

This means that to check if a join $\eta_{i\times j}$ is compatible with a bipartition (A, B), we just have to know the values of $|A \cap V_i| > 0$, $|A \cap V_j| > 0$, $|B \cap V_i| > 0$, and $|B \cap V_j| > 0$. This requires 2 bits of information per label, hence there are 4^k possible configurations. For each node of the k-expression, we have a state for each of the 4^k possible configurations. Note that manipulating these informations is easy when performing a renaming label operation or disjoint union operation.

To obtain our optimal complexity, we have to use a smart way of computing disjoint union operations. We do it in time $\mathcal{O}(4^k k)$ (see Lemma 2.12), while the naive enumeration of pairs of states takes time $\mathcal{O}(16^k)$.

For the lower bound, we encode SAT with an OCT instance that can be constructed by a linear k-expression. The main ideas are to represent SAT variables by paths, to impose that consecutive paths are on distinct sides of the optimal bipartition, and to encode clauses by odd cycles. The reason behind the optimal base of the exponent

is that we can manage to store two variables in one label (more precisely the current endpoints of the two consecutive paths), this will result in additional edges between the two paths but they can be ignored because they are between vertices on distinct sides of the optimal bipartition.

The paths representing variables come from the reduction for Independent Set and the gadgets to transmit information called "arrows" come from the reduction for Odd Cycle Transversal both in [LMS11].

Theorem 3.3 *Given a k-expression describing graph G,* Odd Cycle Transversal *can be solved in time $\mathcal{O}(4^k \cdot k \cdot n)$. Furthermore, the base of the exponent is optimal under the SETH.*

Complete description and proofs are provided in the appendix (2).

3.5. Conclusion

This work contributes to a precise understanding of the hardness of variants of Feedback Vertex Set, by solving open problems left by a previous publication: finding an algorithm for Even Cycle Transversal matching the lower bound for treewidth, finding an algorithm for Node Multiway Cut or Subset Feedback Vertex Set matching the lowerbound for clique-width, finding an optimal algorithm for Odd Cycle Transversal parameterized by clique-width. Our algorithms have a complexity matching lower bounds. For the first open question, we extend the result to a slightly more general setting. For Subset Even Cycle Transversal, we provided some structural properties that may be useful to study this problem in another context. We can ask the following open question: Is there an algorithm for Even Cycle Transversal, Subset Odd Cycle Transversal or Subset Even Cycle Transversal parameterized by clique-width matching the lower bound?

After the end of the internship, a publication [HK21] generalized our result for Odd Cycle Transversal to a more general problem. Their proposed algorithm can be improved by applying our tailored convolution computation.

This contribution has been submitted and accepted to a specialized conference (IPEC 2021). The results are mainly of technical nature, they are not particularly surprising but required work.

A. Appendix

1. Meta-information

First two weeks. Mainly bibliography. First ideas for an ECT algorithm parameterized by treewidth.

Weeks 3-4. Writing a proof of algorithm for ECT. Bibliography and search on clique-width algorithms for the second and third questions.

Weeks 5-6. Found algorithm for OCT and for NMwC parameterized by clique-width, looking for lowerbound for OCT.

Weeks 7-8. Attending talks of "Journées CALAMAR" on graph width parameters. Looking for a lower bound for OCT. Description and proof of NMwC algorithm.

Weeks 9-10. Lower bound proof for OCT. Looking for algorithm for SFVS parameterized by clique-width.

Weeks 11-14. Looking for formalisation of SFVS algorithm. Improving previous proofs. Thinking about generalisation of ECT algorithm.

Weeks 15-16. Writing first version of SFVS algorithm, SECT graph characterization. Attending (remotely) CanaDAM conference.

Weeks 17-18. Group meetings on Directed Feedback Vertex Set. Planning write-up for submission to IPEC 2021. Generalization of ECT algorithm to SECT and related problems.

Last weeks. Finalizing write-up for submission. A few meetings on a project on twin-width with Marcin after submission.

2. Proofs

Feedback Vertex Set relatives

Proof of Lemma 3.1. Suppose that G is a nontrivial 2-connected multigraph containing no even S-cycle.

If G contains no S-vertex, it is either bipartite or not, leading to the first possible forms.

If G contains an S-vertex, it must contain an S-cycle C due to being a nontrivial 2-connected multigraph and C is odd because G contains no even S-cycle.

Claim 2.1 *If two vertices are connected by three disjoints paths at least two of which are S-paths then two of the paths form an even S-cycle.*

Proof. The three cycles formed by combining the paths are S-cycles and they cannot all be odd: if we denote them C_1, C_2, C_3, $|C_3| = |C_1| + |C_2| - 2|C_1 \cap C_2|$. □

Consider a connected component A of $G - V(C)$.

Consider an S-vertex v of A, because G is a nontrivial 2-connected multigraph, there exist two disjoint paths that connect v to distinct vertices a, b of C, a and b satisfy the conditions of claim 2.1 leading to a contradiction. Hence, A cannot contain an S-vertex.

Since G is a nontrivial 2-connected multigraph, there are at least 2 edges between A and distinct vertices of C. Consider 2 arbitrary distinct such edges, they cut C into two paths P_1 and P_2 with extremities u and v. Since A is connected, there is a third u–v path P_3 through A. Only one of P_1 and P_2 may contain an S-vertex, by claim 2.1 applied to P_1, P_2, P_3. In particular, u and v cannot be S-vertices, this implies that the only edges incident to S-vertices in G are edges of cycle C, so S-vertices have degree 2.

Let $\widetilde{A}$ be the connected component of $G-S$ containing A. $\widetilde{A}$ contains a maximal S-free path of C because A is connected to C and cannot be adjacent to S-vertices. $\widetilde{A}$ contains only one maximal S-free path of C because otherwise either we get two edges from A to C that separate C in two S-paths and this was excluded in the previous paragraph, or we have a chord ab in C that connects two distinct maximal S-free-paths and this is excluded by Claim 2.1. In particular, note that this shows that $\widetilde{A}$ has outdegree 2 in G.

Consider a cycle C' of $\widetilde{A}$, then there are 2 disjoint paths from it to the S-vertices adjacent to $\widetilde{A}$. If they are distinct we can connect them with a disjoint path via C. This constructions contains two S-cycles C and $C \Delta C'$ which must both be odd so C' can only be even. Hence $\widetilde{A}$ contains no odd cycle so it is bipartite.

We can conclude that all connected components of $G - S$ are bipartite and that together with S-vertices they form a cycle.

Conversely, if G contains no S-vertex it does not contain any even S-cycle. If it is a cycle of bipartite components and S-vertices with one S-cycle C being odd, then each S-cycle C' goes through all bipartite components and S-vertices. Replacing the path of C by the path of C' in each bipartite component preserves parity because endpoints are unique. We conclude that all S-cycles are odd in G. □

Proof of Lemma 3.2 We now introduce reduction rules that allow us to maintain a simplified description of underlying forests relatively to a set of *active* vertices. Vertices that are not active are called *inactive*. Theses rules and this terminology are derived from [BBBK20].

Definition 11 *Given a $\square$-cycle-free graph G, its underlying forest $F(G)$ and subset of active vertices X, a reduced underlying forest F_r is obtained by applying exhaustively the*

following rules on $F(G)$:

- *Delete inactive vertices of degree at most one.*
- *For each maximal path P with internal inactive vertices of degree 2, we replace it with a path P' with same endpoints, such that P' contains exactly one occurrence of each label present in P and a single S-vertex if P contained one, where endpoints are considered to be contained in P and P'.*

For SECT we add another rule: if a maximal path with internal inactive vertices of degree 2 contains at least 2 vertices labeled "internal bipartite" but no vertex labeled "odd cycle", we keep 2 occurences of the label "internal bipartite".

The set of reduced underlying forests obtained from $F(G)$ with active vertices X is denoted $F(G, X)$.

Observe that a reduced forest is not unique, however properties that we will show on them will not depend on the choice of representative. Furthermore, in our dynamic programming states, we use rooted representations that are not necessarily unique either for each reduced underlying forest.

In the next lemma we show that a reduced forest has bounded size. On a maximal path with internal inactive vertices of degree 2 and 2 vertices labeled "internal bipartite", there is no vertex labeled "odd cycle". Hence the maximum number of vertices on such a path after the reduction is also bounded by the number of labels.

Lemma 2.1 *In a problem using K label symbols (including S-membership), $F \in F(G, X)$ has at most $(K+1)(2|X|-2)+1$ vertices.*

Proof. By the first reduction rule, all leaves of F are active vertices. We use the following result that can be proved by induction.

Claim 2.2 *A non-empty tree with p leaves and internal degree at least 3 has at most $2p-1$ vertices.*

Consider the forest F' obtained from F by replacing maximal paths with inactive internal vertices of degree 2 by edges. Let p denote the number of leaves of F' and q denote the number of active vertices of degree 2. We deduce from the claim that F' has at most $2p+q-1$ vertices. $p+q \leq |X|$ because p and q are cardinals of disjoint parts of X, so F' has at most $2|X|-1$ vertices. Then F' has at most $2|X|-2$ edges which correspond to paths in F with at most K internal vertices. Summing up, F has at most $K(2|X|-2)+2|X|-1 = (K+1)(2|X|-2)+1$ vertices. □

The crucial property preserved by a reduced forest is the following.

Claim 2.3 *For $F \in F(G, X)$, for each pair of active vertices u and v, there is a path between them in $F(G)$ if and only if there is a path between them in F. For each label*

symbol, the path in $F(G)$ contains a vertex with this symbol if and only if the path in F contains a vertex with this symbol.

We immediately deduce the following lemma.

Lemma 2.2 *For $F \in F(G, X)$, for each pair of active vertices u and v, there is a path between them in G if and only if there is a path between them in F. For each type of nontrivial 2-connected component, a u–v path in G goes through at least one such component if and only if the u–v path in F contains a vertex with the corresponding label symbol ("internal bipartite" counts for the bipartite subcomponent but also the S-cycle containing it). There exists a u–v path in G containing an S-vertex if and only if there exists a u–v path in F containing an S-vertex or a vertex labeled "internal bipartite". If there is a u–v path in F, every unlabeled vertex that is on the u–v path in F is also on all u–v paths in G.*

A property that is not preserved by a reduced forest is the length of paths. Since we are only interested in parity, we maintain a $\mathbb{F}_2$-labeling α of edges. We say that α is a *valid* $\mathbb{F}_2$-labeling of $F \in F(G, X)$ if, there exists β a $\mathbb{F}_2$-labeling of the edges of $F(G)$ such that edges incident to vertices labeled "bipartite" or "internal bipartite" are labeled 0 for one side of the bipartition and 1 for the other side, edges incident to other labeled vertices are labeled 0, and edges between unlabeled vertices are labeled 1, and for each edge uv of F, its label is the sum of labels on the edges of the u–v path in $F(G)$. During the application of reduction rules, each edge is given as its label the sum of labels of the path that was connecting its endpoints.

Lemma 2.3 *For $F \in F(G, X)$, for each pair of active vertices u and v connected in G, all u–v paths in G have same parity if and only if the path between u and v in F contains no vertex with label symbol "odd cycle", "not bipartite" or "internal bipartite". Furthermore, when this condition is satisfied, the parity of the paths in G is given by the sum of labels on the edges of F.*

The main technical engine of our algorithms is the following join operation.

Lemma 2.4 *There exists a polynomial-time algorithm that, for every pair of $\square$-cycle-free graphs G_1 and G_2 with $V(G_1) \cap V(G_2) = X$, given on input two reduced forests with valid $\mathbb{F}_2$-labelings (F_1, α_1) and (F_2, α_2), with $F_1 \in F(G_1, X)$ and $F_2 \in F(G_2, X)$, decides whether $G_1 \cup G_2$ is $\square$-cycle-free and, in case of a positive answer, computes a reduced forest $F \in F(G_1 \cup G_2, X)$ and, except for the* SFVS *problem, a valid $\mathbb{F}_2$-labeling α.*

This section is devoted to the proof of Lemma 3.2. Denote $H = F_1 \cup F_2$. Note that the common vertices of F_1 and F_2 are exactly the vertices of X and they are unlabeled vertices.

We start with some claims about the correspondence between H and $G_1 \cup G_2$. We partition 2-connected components of H into *blobs*: a blob is a maximal union of nontrivial

2-connected components of H such that every two 2-connected components of a blob can be connected in H with a path whose all cutvertices are labeled vertices of H. The reason behind this is that although a labeled vertex may be a cutvertex in H, since it is labeled, it corresponds to a nontrivial 2-connected component of G_1 or G_2 by definition of the underlying forest. Observe that if a labeled vertex is in a cycle of H then in the union of underlying forests (not reduced) the cycle obtained by expanding contracted paths contains two vertices of the labeled vertex' corresponding component, meaning that vertices along this cycle are in the same 2-connected component in $G_1 \cup G_2$.

Given a path P between active vertices in F_i, we define a *lift* of this path in G_i as follows. Because there is a path between the endpoints of P in F_i, by Lemma 2.2, there is a path P' between the two vertices in G_i. Note that the nontrivial 2-connected components and cut vertices on the path are already determined. Inside the nontrivial 2-connected components, the path is completed arbitrarily except for the case of odd cycles in SECT where a path without S-vertices is chosen if it exists. The labeled vertices of P are not useful to define P', however by construction of F_i, they give information about the lift.

Claim 2.4 *Let W be a closed walk in a graph G, such that for every vertex v on W, at most one visit of W in v has the property that the in- and out-edge belong to distinct 2-connected components. Then, all edges of W lie in one 2-connected component of G.*

Proof. Assume towards a contradiction that W contains edges of at least two 2-connected components, then it contains a cut vertex v of G. v is a vertex of W, so it is visited once by W going from 2-connected component C_1 to C_2. Since it is a cut vertex of G and W is a closed walk, there must be a second visit going back to C_1. We conclude that v violates the property that at most one visit of W in v has in- and out-edges belonging to distinct 2-connected components. □

Claim 2.5 *If C is a blob of H, then there exists C' a nontrivial 2-connected component of $G_1 \cup G_2$ that contains all unlabeled vertices of C and vertices in components represented by labeled vertices of C.*

Proof. First we consider a cycle C in H, it must consist of a succession of paths in F_1 and paths in F_2 because F_1 and F_2 are forests. From each such maximal path of F_i we can deduce a lift, now the succession of lifts defines a closed walk W. Let v be a repeated vertex in W. Because C is a simple cycle in H, v is visited at most once in C so other visits can only be caused by C visiting a labeled vertex u associated with the nontrivial 2-connected component containing v. In such other visit, the vertices from which W enter the 2-connected component represented by u are distinct from v because it cannot be visited more than once in C. This means that the in- and out-edges of such visit are in the nontrivial 2-connected component represented by u. We conclude that conditions of Claim 2.4 are satisfied and deduce that W is contained in one 2-connected component of $G_1 \cup G_2$.

Now if two cycles C_1 and C_2 of H are not edge disjoint, then there exist two distinct vertices a, b contained by both walks W_1 and W_2 obtained by lifts from C_1 and C_2. This means that the 2-connected components of $G_1 \cup G_2$ containing W_1 and W_2 have two distinct vertices in common, so it must be the same 2-connected component.

If a labeled vertex u is in a cycle C of H, then all of the vertices that are contained in the nontrivial 2-connected component B represented by u are inside the same maximal 2-connected component of $G_1 \cup G_2$ as W the walk obtained by lifts from C. Indeed, if u is in C, then W must contain two distinct vertices of B, and since B is 2-connected it must be contained in the same 2-connected component of $G_1 \cup G_2$ as W.

Finally, if two cycles C_1 and C_2 of H are edge disjoint but share a labeled vertex u, then consider the walks W_1 and W_2 in $G_1 \cup G_2$ obtained by lifts from C_1 and C_2. Each walk must contain two distinct vertices of the 2-connected subgraph B represented by u. Hence, the 2-connected component of $G_1 \cup G_2$ containing B also contains W_1 and W_2.

We conclude from the previous points that for B a blob of H, all of its vertices and the vertices contained in its labeled vertices' associated nontrivial 2-connected component are in the same maximal 2-connected component of $G_1 \cup G_2$. □

For a cycle C in $G_1 \cup G_2$ that contains edges of both G_1 and G_2, we construct a walk W in H as follows. Since C contains edges of both G_1 and G_2, it can be decomposed into paths $P_1, P_2, \ldots, P_\ell$, with ℓ even, and paths $P_1, P_3, P_5, \ldots$ lying in G_1 and P_2, P_4, P_6 lying in G_2; the path P_i has both endpoints being active vertices, one being also the endpoint of P_{i-1} and one being also the endpoint of P_{i+1}. By Lemma 2.2, for each P_i there is a corresponding path P_i' between the same endpoints in F_1 or F_2. The concatenation of the paths P_i' is a closed walk in H which we call *the contracted walk of* C.

Claim 2.6 *If C is a cycle of $G_1 \cup G_2$ that contains at least one edge of G_1 and at least one edge of G_2, then its contracted walk in H visits more than once only labeled vertices. Furthermore, it is contained in a single blob of H.*

Proof. Because vertices of C appear at most once and vertices of the contracted walk W that are not labeled are vertices of C appearing as many times, we conclude that only labeled vertices can be visited more than once. Now only labeled vertices can be cut vertices in W and by definition of blobs they do not separate distinct blobs. □

We now move to the description of the algorithm of Lemma 3.2. For all problems, start by computing the 2-connected components of H and blobs. Then, proceed as follows, depending on the problem at hand.

SFVS. Check that no blob of H contains an S-vertex. Then, return any element of $F(H, X)$ as the desired forest. (There is no need to compute α for SFVS.)

ECT. For each blob B of H, proceed as follows. Reject if B is not a cycle or contains a vertex with label "odd cycle". Also reject if the sum of labels of edges of the cycle

is 0. Otherwise, the edges of B are removed and a vertex adjacent to all vertices of B is added, with a label "odd cycle". The new edges are given label 0.

SOCT. For each blob B of H, proceed as follows. Check if B contains a cycle that does not sum to 0, if not, compute the bipartition of vertices based on their potential. Reject if B contains an S-vertex and either a cycle that does not sum to 0 or a vertex labeled "not bipartite". Otherwise, the internal edges of B are removed, a vertex adjacent to unlabeled vertices of B is added, labeled vertices of B are identified to it. The new vertex is an S-vertex if B contains an S-vertex and is labeled "bipartite" if B contained no cycle that does not sum to 0 nor vertex labeled "not bipartite", otherwise, it is labeled "not bipartite". If we add a vertex labeled "bipartite", two cases arise for edges incident to it. First, if the other endpoint is in B, it is labeled with the computed potential of this endpoint. Second, if the other endpoints is not in B, it corresponds to an edge that was incident to a former labeled vertex v in B, we add the potential of v to the previous label of the edge. If we add a vertex labeled "not bipartite", we label all edges with 0.

SECT. For each blob B of H, proceed as follows. First, we consider the case when B contains no S-vertex. Check with depth-first search if B contains a cycle that does not sum to 0, if not, compute the bipartition of vertices based on their potential. Reject if there is more than one vertex labeled "internal bipartite" in B, or there is a vertex labeled "internal bipartite" and a cycle that does not sum to 0, or there is a vertex labeled "internal bipartite" and a vertex labeled "not bipartite". Otherwise, add a vertex, make it adjacent to unlabeled vertices of B and identify labeled vertices of B to it. This additional vertex is labeled "bipartite" if B contained no cycle that does not sum to 0 nor vertex labeled "not bipartite" or "internal bipartite", it is labeled "internal bipartite" if it contained a vertex labeled "internal bipartite", and it is labeled "not bipartite" otherwise. If we add a vertex labeled "bipartite" or "internal bipartite", two cases arise for edges incident to it. First, if the other endpoint is in B, it is labeled with the computed potential of this endpoint. Second, if the other endpoints is not in B, it corresponds to an edge that was incident to a former labeled vertex v in B, we add the potential of v to the previous label of the edge. If the additional vertex is labeled differently, we label all edges with 0.

Consider now the case when B contains an S-vertex. Reject if one of the following cases occur: B contains a vertex labeled "not bipartite", "odd cycle", or "internal bipartite", there is an S-vertex in B that is of degree more than 2, or there is a connected component of $B - S$ where the number of edges with exactly one endpoint in the said component is more than 2. Otherwise, pick some S-vertex v in B. We define $\widetilde{B}$ as the graph obtained from B by subdividing v in two vertices separated by a new edge labeled 1 and each adjacent to one of the two edges that were adjacent to v. Check with depth-first search if $\widetilde{B}$ contains a cycle that does not sum to 0, if not, compute the bipartition of vertices based on their potential. If we did not reject, for each connected component B' of of $B - S$, a vertex adjacent to unlabeled vertices of B' and labeled "internal bipartite" is added and labeled

vertices of B' are identified with it. Then edges of B incident to its S-vertices are also removed and a vertex labeled "odd cycle" adjacent to S-vertices of C and to the new vertices labeled "internal bipartite" is added. For an edge e incident to a vertex labeled "internal bipartite" but not to a vertex labeled "odd cycle", either e is labeled with the computed potential of its other endpoint in B, or the other endpoint is not in B, so e used to be incident to labeled vertex v in B, then we add the potential of v to the edge's previous label. Other edges are labeled 0.

We conclude by reducing the forest obtained from H and denote by F the resulting forest.

Claim 2.7 *The join processing rejects if and only if $G_1 \cup G_2$ is not $\square$-cycle-free.*

Proof. For SFVS, if the join processing rejects then there is a blob of H containing an S-vertex, by Claim 2.5, there is a nontrivial 2-connected component containing an S-vertex in $G_1 \cup G_2$ so there is an S-cycle.

Conversely, if $G_1 \cup G_2$ is not S-cycle-free, since G_1 and G_2 are S-cycle-free, it contains an S-cycle C. We consider the contracted walk of C in H. By Claim 2.6, this walk is contains an S-vertex in H and is contained in a blob of H, causing the join to reject.

For ECT, if the join processing rejects one the following happens.

- There is a blob B that is not a cycle in H, then by Claim 2.5, there is a nontrivial 2-connected component C of $G_1 \cup G_2$ containing its vertices and because it each path in B can be lifted in a walk in C, there is more than one cycle in C.
- There is a blob B that contains a vertex labeled "odd cycle", then by Claim 2.5, there is a 2-connected component of $G_1 \cup G_2$ containing an odd cycle and a vertex that is not in this cycle.
- The sum of labels on edges of the cycle is 0 in a blob of H that is a cycle, then there must be a non trivial 2-connected component containing its vertices in $G_1 \cup G_2$ by Claim 2.5 and it must be bipartite, so there is an even cycle in $G_1 \cup G_2$.

In all cases, we conclude that $G_1 \cup G_2$ is not even-cycle-free with Lemma 3.1.

Conversely, if there exists an even cycle in $G_1 \cup G_2$, since it is not contained in G_1 or G_2, it contains edges of both, by Claim 2.6, its contracted walk W visits more than once only labeled vertices. First, if W contains a labeled vertex it causes a reject. Otherwise, W must be a cycle of H. Consider the blob containing W, either it is not a cycle, causing a reject, or it is a cycle, hence it is exactly W which cannot sum to 0 because it represents an even cycle, this also causes a reject.

For SOCT, if the join processing rejects then there must exist a blob B of H that contains an S-vertex v and either a cycle that does not sum to 0 or a vertex labeled "not bipartite", by Claim 2.5 there is a 2-connected component C of $G_1 \cup G_2$ containing v but also an odd cycle: if B contains a vertex labeled "not bipartite", then C contains an odd cycle contained in G_1 or G_2, otherwise B contains a cycle that does not sum to 0

which means that C is not bipartite by Lemma 2.3. From Lemma 3.1 we conclude that $G_1 \cup G_2$ is not odd-S-cycle-free.

Conversely, if $G_1 \cup G_2$ contains an odd S-cycle, since it is not contained in G_1 or G_2, by Claim 2.5, there must be a corresponding contracted walk W in H, which is contained by a blob B of H. If the blob B contains a vertex labeled "not bipartite", it causes a reject, otherwise the walk W in H encodes parity correctly by Lemma 2.3, so a cycle that does not sum to 0 will be found, causing a reject.

For SECT, if the join processing rejects then there must exist a blob B of H that does not respect a condition, let C denote the nontrivial 2-connected component of $G_1 \cup G_2$ containing its vertices (Claim 2.5). We first suppose that B contains no S-vertex.

- If B contains two vertices labeled "internal bipartite", then either C contains only one odd-S-cycle component but then we added an S-free path between two of its bipartite subcomponents, or C contains two odd-S-cycle components. In both cases, C is not of a form that is possible for $G_1 \cup G_2$ even-S-cycle-free.
- If B contains a vertex labeled "internal bipartite" and a vertex labeled "not bipartite" or a cycle that does not sum to 0, then C contains an odd-S-cycle component and an odd cycle (either from the contained component that is not bipartite, or the cycle that does not sum to 0). In both cases, C is not of a form that is possible for $G_1 \cup G_2$ even-S-cycle-free.

We now suppose that B contains an S-vertex.

- If B contains a vertex labeled "not bipartite" or "odd cycle", then C cannot be of the form that is possible for $G_1 \cup G_2$ even-S-cycle-free.
- If B contains a vertex labeled internal bipartite then either we have added a path containing an S-vertex between two vertices of the same bipartite subcomponent of an odd-S-cycle component contained in G_1 or G_2, or we have added a path between two existing bipartite subcomponents. In both cases C cannot be of the form that is possible for $G_1 \cup G_2$ even-S-cycle-free.
- If B contains an S-vertex that is of degree more than two, or a connected component of $B - S$ has more than 2 edges with exactly one endpoint in the said component, then the same holds in C. This contradicts the form of a nontrivial 2-connected component containing an S-vertex for $G_1 \cup G_2$ even-S-cycle free
- If $\widetilde{B}$ contains a cycle that does not sum to 0, then there must exist an even S-cycle in G by construction of $\widetilde{B}$ and Lemma 2.3. This immediately implies that $G_1 \cup G_2$ is not even-S-cycle-free.

Conversely, if $G_1 \cup G_2$ contains an even S-cycle C, then because G_1 and G_2 don't contain it, by Claim 2.6, the corresponding contracted walk W in H is in a single blob B of H. B contains an S-vertex from containing W. Either, B contains labeled vertices that are not labeled bipartite or does not respect the degree conditions, causing a reject, or it contains only labeled vertices with label "bipartite" which means that the parity of paths in B is correctly encoded (Lemma 2.3), and all S-cycles

contain all S-vertices from degree properties, in particular, we can deduce that any S-vertex chosen to be subdivided in $\widetilde{B}$ would be in C, and then the cycle has even length which will cause a the existence of a cycle that does not sum to 0 in $\widetilde{B}$, causing a reject. □

Claim 2.8 *The computed forest is in $F(G_1 \cup G_2, X)$ and the computed $\mathbb{F}_2$-labeling α of its edges is valid for it, if $G_1 \cup G_2$ is $\square$-cycle-free.*

Proof. Consider a nontrivial 2-connected component C of $G_1 \cup G_2$, it may be contained in G_1 or G_2, in this case its active vertices have already been connected to a labeled vertex representing C so there is nothing more to do. If C contains edges from both G_1 and G_2 then there must be a blob B in H that contains part of the active vertice of C. By construction, active vertices of C will be adjacent to the labeled vertex representing their component: they can only be in B or adjacent to a labeled vertex in B and labeled vertices were identified to the new vertex. Furthermore, one can check that the choice of labels correspond to the forms of the components of $G_1 \cup G_2$ based on the information stored in the labels of F_1 and F_2, and no information can be missing (Lemmata 2.2 and 2.3).

The vertices that were removed in $F(G_1)$ and in $F(G_2)$ to obtain F_1 and F_2 must still be removed in $F(G_1 \cup G_2)$. In particular when a path of inactive vertices of degree 2 leads to the creation of a cycle, its inactive vertices become leaves and can then all be simplified. Because after producing a forest we reduce it, there cannot be additional reductions to be performed, hence the computed forest is in $F(G_1 \cup G_2, X)$.

It is straightforward to check that α is a valid $\mathbb{F}_2$-labeling of F. □

We conclude the proof of Lemma 3.2 with an observation that it is straightforward to implement the discussed algorithm to run in time polynomial in the input size. Finally, note that the input size is of size $\mathcal{O}(|X|)$. □

Proof of Lemma 3.3. By induction on T.

1. If t is a leaf node, then choosing $U = \varnothing$ we have that $(t, \varnothing, \varnothing, \varnothing)$ is admissible.
2. If t is an introduce vertex node with child t', introducing vertex v, then for (t, Y, F, α) a reachable state, there exists the state that gave it optimal value (t', Y', F', α'). By induction hypothesis applied to (t', Y', F', α'), there exists U' such that $Y' = X_{t'} \setminus U'$, $G_{t'} - U'$ is $\square$-cycle-free, F' is a forest description of a member $F(G_{t'} - U', Y')$, α' is a valid $\mathbb{F}_2$-labeling of F' and $d[t', Y', F', \alpha'] = c(U')$. If $v \notin Y$, then we can deduce the transition that was used, we set $U = U' \cup \{v\}$, since $Y' = X_{t'} \setminus U'$ and $X_t = X_{t'} \cup \{v\}$, we have $Y = X_t \setminus U$. We also have $d[t, Y, F, \alpha'] = d[t', Y', F', \alpha'] + c(v) = c(U') + c(v) = c(U)$. $G_t - U = G_{t'} - U'$, $F' = F$, $Y' = Y$, so $G_t - U$ is $\square$-cycle-free, F is a forest description of a member of $F(G_t - U, Y)$ and α is a valid $\mathbb{F}_2$-labeling of F. If $v \in Y$, then we deduce the transition and set $U = U'$. then $Y = Y' \cup \{v\}$ and since $Y' = X_{t'} \setminus U'$ and $X_t = X_{t'} \cup \{v\}$,

we have $Y = X_t \setminus U$. We also have $d[t, Y, F, \alpha] = d[t', Y', F', \alpha'] = c(U') = c(U)$. $G_t - U$ is $\square$-cycle-free because $G_{t'} - U'$ is and v is isolated. F is a forest description of $F(G_t - U, Y)$ by construction, and α is a valid $\mathbb{F}_2$-labeling of F because v is an isolated vertex.

3. If t is a forget vertex node with child t', forgetting vertex v, then for (t, Y, F, α) a reachable state, there exists the state that gave it optimal value (t', Y', F', α'). By induction hypothesis applied to (t', Y', F', α'), there exists U such that $Y' = X_{t'} \setminus U$, $G_{t'} - U$ is $\square$-cycle-free, F' is a forest description of a member of $F(G_{t'} - U, Y')$, α' is a valid $\mathbb{F}_2$-labeling of F' and $d[t', Y', F', \alpha'] = c(U)$. In all cases, $d[t, Y, F, \alpha] = c(U)$. If $v \notin Y'$, then we can deduce the transition that was used, and there is nothing to show since $Y = Y'$, $G_t - U = G_{t'} - U$, $F = F'$ and $\alpha = \alpha'$. If $v \in Y'$, then we can deduce the transition that was used. We have $Y = Y' \setminus \{v\} = X_t \setminus U$. $G_t - U = G_{t'} - U \cup H$ so, by Claim 2.7, $G_t - U$ is $\square$-cycle-free. By Claim 2.8, F is a forest description of a member of $F(G_t - U, Y)$ and α is a valid $\mathbb{F}_2$-labeling of F.
4. If t is a join node with children t_1 and t_2, then for (t, Y, F, α) a reachable state, there exist states that gave it the optimal value (t_1, Y, F_1, α) and (t_2, Y, F_2, α_2). By induction hypothesis applied to these states, for $i \in \{1, 2\}$, there exists U_i such that $Y = X_{t_i} \setminus U_i$, $G_{t_i} - U_i$ is $\square$-cycle-free, F_i is a forest description of a member of $F(G_{t_i} - U_i, Y)$, α_i is a valid $\mathbb{F}_2$-labeling of F_i and $d[t_i, Y, F_i, \alpha_i] = c(U_i)$. We set $U = U_1 \cup U_2$, then $G_t - U = G_{t_1} - U_1 \cup G_{t_2} - U_2$ and since $X_t = X_{t_1} = X_{t_2}$, we have $Y = X_t \setminus U$. By Claim 2.7, $G_t - U$ is $\square$-cycle-free. By Claim 2.8, F is a forest description of a member of $F(G_t - U, Y)$ and α is a valid $\mathbb{F}_2$-labeling of F.

$\square$

Proof of Lemma 3.4 By induction on T.

1. If t is a leaf node, then for $U \subseteq V(G_t)$, we have $U = \varnothing$ and $G_t - U$ is the empty graph which is $\square$-cycle-free. we have a reachable state $(t, \varnothing, \varnothing, \varnothing)$ with $d[t, \varnothing, \varnothing, \varnothing] = 0 \leq c(U)$.
2. If t is an introduce vertex node with child t' introducing vertex v, then for $U \subseteq V(G_t)$ such that $G_t - U$ is $\square$-cycle-free, we set $U' = U \setminus \{v\} = U \cap V(G_{t'})$. $G_{t'} - U'$ is an induced subgraph of $G_t - U$, hence it is $\square$-cycle-free. By induction hypothesis, there exist F', α' such that $(t', X_{t'} \setminus U', F', \alpha')$ is reachable and $d[t', X_{t'} \setminus U', F', \alpha'] \leq c(U')$. There are two cases, either $v \in U$ or $v \notin U$ and a transition for each case so there exist F, α such that $(t, X_t \setminus U, F, \alpha)$ is reachable and $d[t, X_t \setminus U, F, \alpha] \leq c(U)$.
3. If t is a forget vertex node with child t' forgetting vertex v, then for $U \subseteq V(G_t)$ such that $G_t - U$ is $\square$-cycle-free, $G_{t'} - U$ is a subgraph of $G_t - U$ so it is $\square$-cycle-free. By induction hypothesis, there exist F', α' such that $(t', X_{t'} \setminus U, F', \alpha')$ is reachable and $d[t', X_{t'} \setminus U, F', \alpha'] \leq c(U)$. By Claim 2.7, there is a transition producing F, α such that $(t, X_t \setminus U, F, \alpha)$ is reachable and $d[t, X_t \setminus U, F, \alpha] \leq c(U)$.
4. If t is a join node with children t_1 and t_2, then for $U \subseteq V(G_t)$ such that $G_t - U$ is $\square$-cycle-free, we set $U_1 = U \cap V(G_{t_1})$ and $U_2 = U \cap V(G_{t_2})$. Hence, $G_{t_1} - U_1$ and

$G_{t_2} - U_2$ are induced subgraphs of $G_t - U$ so they are $\square$-cycle-free. By induction hypothesis, for $i \in \{1, 2\}$, there exist F_i, α_i such that $(t_i, X_{t_i} \setminus U_i, F_i, \alpha_i)$ is reachable and $d[t_i, X_{t_i} \setminus U_i, F_i, \alpha_i] \leq c(U_i)$. By Claim 2.7, there is a transition producing F, α such that $(t, X_t \setminus U, F, \alpha)$ is reachable and $d[t, X_t \setminus U, F, \alpha] \leq d[t_1, X_{t_1} \setminus U_1, F_1, \alpha_1] + d[t_2, X_{t_2} \setminus U_2, F_2, \alpha_2] \leq c(U_1) + c(U_2) = c(U)$.

$\square$

Node Multiway Cut

Algorithm and auxiliary graph A state in our dynamic programming will consist of a near-partition $\mathcal{P}$ of labels according to the expected evolution of their connectivity, where two subsets have a special role : $P_\varnothing$ will denote labels that contain no vertex, P_f will denote labels that will only be joined to empty labels in the future, other subsets $P_1, \ldots, P_k$ represent labels that may form a connected component in the future, together they form $\mathcal{P}^*$; and a function $\phi : [k] \to \{0, 1\}$ indicating if the component contains a terminal.

Given a labeled graph $G = (V, E, \gamma)$ and a near-partition $\mathcal{P}$ of labels, we define the auxiliary graph

$$H_{\mathcal{P}}(G) := \left(V, E \cup \{uv \mid u, v \in \bigcup_{j \in P_i} V_j(G), u \neq v, P_i \in \mathcal{P}^*\}, \gamma \right)$$

In $H_{\mathcal{P}}(G)$, for each $P_i \in \mathcal{P}^*$, $\bigcup_{j \in P_i} V_j(G)$ is completed to form a clique.

We say that $X \subseteq V(G_t)$ is *admissible* for state $(t, \mathcal{P}, \phi)$ if :

- $\forall j \in P_\varnothing, V_j(G_t) \subseteq X$ (or equivalently $V_j(G_t - X) = \varnothing$)
- for $u \in V_\alpha(G_t - X), v \in V_\beta(G_t - X)$ with $\alpha \in P_i, \beta \in P_j, i \neq j$, there is no path from u to v in $G_t - X$
- for each $P_i \in \mathcal{P}^*$, if the connected component of vertices $\bigcup_{j \in P_i} V_j(G_t - X)$ in $H_{\mathcal{P}}(G_t - X)$ contains a terminal, $\phi(i) = 1$
- each connected component of $H_{\mathcal{P}}(G_t - X)$ contains at most one terminal

The minimum solution to Node Multiway Cut is stored in $d[r, \mathcal{P}_f, \widetilde{0}]$ where r is the root of the expression, $\mathcal{P}_f$ contains all labels in P_f, $\widetilde{0}$ is the constant function with value 0.

Computations for each operation of the k-expression :

1. $G_t = i(v)$: For non-terminals, enumerate all possible $(\mathcal{P}, \widetilde{0})$ and put value $c(v)$ if i is in $P_\varnothing$ and 0 otherwise. For other all other states, put value $+\infty$.

   For terminals, enumerate all possible $(\mathcal{P}, \phi)$ and put value 0 if i is not in $P_\varnothing$ and $\phi(j) = 1$ with $P_j \in \mathcal{P}^*$ containing i, and otherwise value $+\infty$.

2. $G_t = G_{t_1} \oplus G_{t_2}$: The value for each state of t is computed by $d[t, \mathcal{P}, \phi] :=$

   $\min\{d[t_1, \mathcal{P}, \phi_1] + d[t_2, \mathcal{P}, \phi_2] \mid \forall P_i \in \mathcal{P}^*, \phi_1(i) = 0 \text{ or } \phi_2(i) = 0, \phi(i) = \phi_1(i) + \phi_2(i)\}$

3. $G_t = \eta_{i \times j}(G_{t'})$: For each state $(t', \mathcal{P}, \phi)$ such that $i \in P_\varnothing$ or $j \in P_\varnothing$ or $\exists \alpha, i, j \in P_\alpha$, we apply the following transitions:

   $$d[t, \mathcal{P}', \phi'] \leftarrow d[t', \mathcal{P}, \phi]$$

   with $\mathcal{P}'$ describing all possible near-partitions obtained from $\mathcal{P}$ by moving i, j according to the automaton described in figure 3 and ϕ' obtained from ϕ by changing to 0 the values for α such that P_α becomes empty in $\mathcal{P}'$.

4. $G_t = \rho_{i \to j}(G_{t'})$: For each state $(t', \mathcal{P}, \phi)$ such that $i \in P_\varnothing$ or $j \in P_f$ or $\exists \alpha, i, j \in P_\alpha$, we apply the transitions:

   $$d[t, \mathcal{P}', \phi'] \leftarrow d[t', \mathcal{P}, \phi]$$

   with $\mathcal{P}'$ describing all possible near-partitions obtained from $\mathcal{P}$ by moving i to any of $P_\varnothing, P_1, \ldots, P_k, P_f$ and ϕ' obtained from ϕ by changing to 0 the values for α such that P_α becomes empty or contains only i in $\mathcal{P}'$.

Transitions preserve admissibility We will call empty a label that is in $P_\varnothing$.

1. $G_t = i(v)$: Only admissible states have finite value.
2. $G_t = G_{t_1} \oplus G_{t_2}$: If there exist X_1 and X_2 admissible for $(t_1, \mathcal{P}, \phi_1)$ and $(t_2, \mathcal{P}, \phi_2)$ s.t. $\forall P_i \in \mathcal{P}^*, \phi_1(P_i) = 0$ or $\phi_2(P_i) = 0$, then $X := X_1 \cup X_2$ is admissible for $(t, \mathcal{P}, \phi)$ for ϕ s.t. $\phi = \phi_1 + \phi_2$:
   - $\bigcup_{j \in P_\varnothing} V_j(G_t) = \left(\bigcup_{j \in P_\varnothing} V_j(G_{t_1})\right) \cup \left(\bigcup_{j \in P_\varnothing} V_j(G_{t_2})\right) \subseteq X_1 \cup X_2$
   - for $u \in \bigcup_{j \in P_\alpha} V_j(G_t - X)$, $v \in \bigcup_{j \in P_\beta} V_j(G_t - X)$ with $\alpha \neq \beta$, if they are not from the same subtree, there cannot be a path between them in $G_t - X$. If they are from the same subtree, there is no path between them because X_1 and X_2 are admissible.
   - the conditions on ϕ ensure that if the connected component of $P_i(V_t \setminus X)$ contains a terminal, $\phi(A) = 1$.
   - since only one of ϕ_1 and ϕ_2 is positive on each i, the corresponding connected component of $H(G_t - X)$ contains at most one terminal.
3. $G_t = \eta_{i \times j}(G_{t'})$: If X is admissible for $(t', \mathcal{P}', \phi')$ and i and j that are in the same $P_\alpha \in \mathcal{P}'^*$, then X is admissible for $(t, \mathcal{P}', \phi)$ because $H_{\mathcal{P}'}(G_t - X) = H_{\mathcal{P}'}(G_{t'} - X)$. If $\mathcal{P}$ is obtained from $\mathcal{P}'$ by putting i and/or j in P_f, X is admissible for $(t, \mathcal{P}, \phi)$:
   - $\bigcup_{j \in P_\varnothing} V_j(G_t) = \bigcup_{j \in P_\varnothing} V_j(G_{t'}) \subseteq X$
   - for $u \in \bigcup_{j \in P_\alpha} V_j(G_t - X), v \in \bigcup_{j \in P_\beta} V_j(G_t - X)$ with $\alpha \neq \beta$, suppose towards a contradiction that there is a u-v path in $G_t - X$, then there is a u-v path in $H_{\mathcal{P}}(G_t - X)$ which contains a subset of edges of $H_{\mathcal{P}'}(G_t - X) = H_{\mathcal{P}'}(G_{t'} - X)$,

but then we get a u-v path in $G_{t'} - X$ with $u \in \bigcup_{j \in P'_\alpha} V_j(G_{t'} - X), v \in \bigcup_{j \in P'_\beta} V_j(G_{t'} - X)$ so X is not admissible for $(t', \mathcal{P}', \phi')$.

- If $\phi(i) = 0$ then the corresponding component in $H_{\mathcal{P}'}(G_t - X)$ contains no terminal so its vertices form components in $H_{\mathcal{P}}(G_t - X)$ that do not contain terminals.
- Connected components in $H_{\mathcal{P}}(G_t - X)$ are subdivisions of those in $H_{\mathcal{P}'}(G_t - X)$ so they cannot contain more than 1 terminal.

Moving a label from $P_\varnothing$ to some $P_\alpha \in \mathcal{P}^*$ does not affect $H_{\mathcal{P}}(G_t - X)$ so it does not affect admissibility.

4. $G_t = \rho_{i \to j}(G_{t'})$: If X is admissible for $(t', \mathcal{P}', \phi')$, if i and j are in the same $P_\alpha \in \mathcal{P}'^*$ or $i \in P_\varnothing$, then $H_{\mathcal{P}}(G_t - X) = H_{\mathcal{P}'}(G_{t'} - X)$. If $j \in P_f$ then we apply the same arguments as in the previous case, to show that X is admissible.

All possible cuts are considered Consider $X \subseteq V(G)$ a node multiway cut, we will show states such that $X \cap V(G_t)$ is admissible, that are linked by our transitions. We construct such states by descending from the final state $(r, \mathcal{P}_f, \widetilde{0})$, where r is the root of the k-expression tree.

We will call empty a label i such that $V_i(G_t - X) = \varnothing$.

Suppose we are in t with state $(t, \mathcal{P}, \phi)$.

We describe the partition for children nodes by giving the operations to perform on $\mathcal{P}$ to obtain $\mathcal{P}'$. Connected components C of $G - X$ are attributed indices $\delta(C)$ greedily, for connected component C, P_C denotes $P_\delta(C)$. Note that there cannot be more than k indices used because there are only k labels and, when P_α becomes empty, α may be attributed again.

For $P_C \in \mathcal{P}'^*$, we put $\phi(\delta(C)) = 1$ if $C \cap V(G_t)$ contains a terminal and $\phi(\delta(C)) = 0$ otherwise.

If $G_t = \eta_{i \times j}(G_{t'})$, for each $\alpha \in \{i, j\}$, if $V_\alpha(G_t) \subseteq X$ then α is put in $P'_\varnothing$, if $V_\alpha(G_t - X)$ is contained in only one connected component C of $G - X$, put α in $P_C \in \mathcal{P}'^*$.

If $G_t = \rho_{i \to j}(G_{t'})$, if $V_i(G_{t'} - X)$ is empty, put it in $P'_\varnothing$ else put i where j is.

If $G_t = G_{t_1} \oplus G_{t_2}$, the partition is left unchanged for each branch and ϕ_1, ϕ_2 are defined as mentionned earlier for the children of the node.

This constructions uses only transitions of our algorithm :

- Join: Moving an affected label from $P_\varnothing$ to anywhere else is considered so putting such label in $P'_\varnothing$ in the construction is fine. The vertices contained in labels do not change in this node, so some non empty label in $P_C \in \mathcal{P}$ will necessarily be in $P'_C \in \mathcal{P}'^*$. Affected labels can be moved to P_f. If none of the labels is empty, then the joined labels can't be in different components of $G - X$ because G is defined by the expression so $G - X$ contains edges $\{uv : u \in V_i(G_t - X), v \in V_j(G_t - X)\}$.

Moving labels to $\mathcal{P}^*$ is only done when the labels are empty so ϕ will satisfy the transition conditions.

- Relabel: Label i can be put anywhere so, in the construction, it does not matter where it is in $\mathcal{P}$ (at this point it contains nothing). The labels always satisfy the transition conditions. Vertices remain in the same element of $\mathcal{P}$ so ϕ is unaffected.
- Disjoint union: Since connected components of $G - X$ contain at most one terminal, and the vertices in the children are disjoint, the new functions satisfy the conditions.

X is admissible for the final state $(r, \mathcal{P}_f, \phi_\varnothing)$:

- $\bigcup_{j \in P_\varnothing} V_j(G_r) = \varnothing \subseteq X$
- all vertices are in labels of P_f.
- $\mathcal{P}_f^* = \varnothing$
- $H_{\mathcal{P}_f}(G_t - X) = G_t - X$ is such that each component contains at most one terminal since X is a node multiway cut.

If $X \cap V_t$ is admissible for constructed state $(t, \mathcal{P}, \phi)$, we show that $X' := X \cap V(G_{t'})$ is admissible for constructed state $(t', \mathcal{P}', \phi')$ of some children t'.

It is clear that $\bigcup_{j \in P'_\varnothing} V_j(G_{t'}) \subseteq X$ since only empty labels are added to $P'_\varnothing$.

For relabel, $H_{\mathcal{P}}(G_t - X) = H_{\mathcal{P}'}(G_t - X)$ because only an empty label moves to form $\mathcal{P}'$ and $\phi' = \phi$, so X is admissible for $(t', \mathcal{P}', \phi')$.

For disjoint union :

- for $u \in \bigcup_{j \in P_\alpha} V_j(G_{t'} - X), v \in \bigcup_{j \in P_\beta} V_j(G_{t'} - X)$ with $\alpha \neq \beta$, suppose towards a contradiction that there is no u-v path in $G_{t'} - X'$, then there is a u-v path in $G_t - X$ but then X can't be admissible for $(t, \mathcal{P}, \phi)$.
- for $P_C \in \mathcal{P}^*$, if the connected component of $\bigcup_{j \in P_C} V_j(G_{t'} - X)$ in $H_{\mathcal{P}}(G_{t'} - X)$ contains a terminal, then $C \cap V(G_{t'})$ contains it and, since its a terminal, $\phi(\delta(C)) = 1$.
- connected components of $H_{\mathcal{P}}(G_{t'} - X)$ contain less terminals because we only remove edges and vertices from $H_{\mathcal{P}}(G_t - X)$, thus they also contain at most one terminal.

For join node, $H_{\mathcal{P}^*}(G_t - X) = H_{\mathcal{P}^*}(G_{t'} - X)$ because we use transitions of the algorithm. Moving empty labels has no effect on $H(G_t - X)$ and moving labels to $\mathcal{P}^*$ that are contained in the same connected component changes graph H but not its connected components which is enough for the desired properties.

Therefore, the construction yields admissible states for the leaves where all admissible states are enumerated so the algorithm cannot miss them. We can conclude that all node multiway cuts will be found.

We can conclude that the root contains the size of a minimal node multiway cut (the last property of admissibility for the final state is exactly what we want).

Complexity There are less than $(k+2)^k 2^k$ states per node. For merge nodes, there are less than 2^k transitions per state, and for other nodes there are less than k^2 transitions per state. Thus we achieve $\mathcal{O}((k+2)^{k+2}4^k n)$ time complexity, with n the size of the k-expression.

This is $2^{\mathcal{O}(k \log k)}n)$ which meets the lower bound.

Subset Feedback Vertex Set

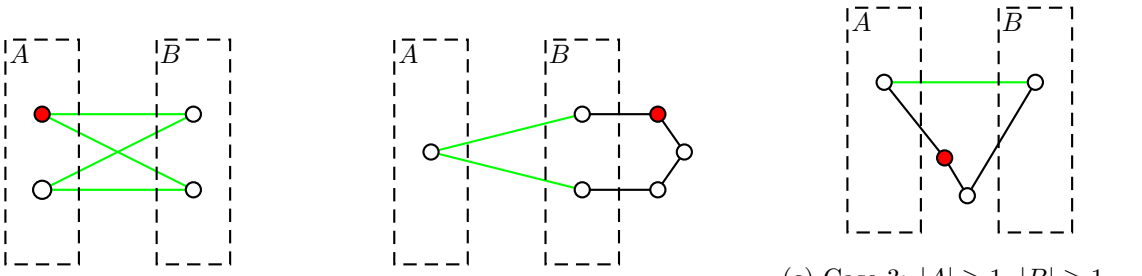
We describe a dynamic programming algorithm to solve Subset Feedback Vertex Set on clique-width expressions. With a bottom-up computation, it builds small labeled forests that describe the graphs that can be obtained by vertex deletion but also incorporate some guesses on the evolution of the graph.

A state of our dynamic programming will consist of a node of the k-expression, a partially labeled forest, and a label state assignment $\mathcal{P} : [k] \to \mathcal{Q}$, with $\mathcal{Q} = \{Q_\varnothing, Q_1, Q_1^*, Q_2, Q_w, Q_w^*, Q_f\}$ the set of label states.

Now, considering an S-cycle-free graph $\widetilde{G}$ obtained by vertex deletion, we will say that a label i is *compatible* with label state:

- $Q_\varnothing$ if no vertex of $\widetilde{G}$ is labeled i;
- Q_1 if exactly one vertex of $\widetilde{G}$ is labeled i, and it is not in S;
- Q_1^* if exactly one vertex of $\widetilde{G}$ is labeled i, and it is in S;
- Q_2 if at least two vertices of $\widetilde{G}$ are labeled i, they are not in S, and no S-path in $\widetilde{G}$ has both its endpoints labeled i;
- Q_w if at least two vertices of $\widetilde{G}$ are labeled i, at least one S-vertex is labeled i, and no S-path in $\widetilde{G}$ has both its endpoints labeled i;
- Q_w^* if at least two vertices of $\widetilde{G}$ are labeled i, no path in $\widetilde{G}$ has both its endpoints labeled i; and
- Q_f if at least two vertices of $\widetilde{G}$ are labeled i.

These conditions, together with the constraints on joins that are expressed above, aim to capture cases for which a join between labels of pairs of label states will not create S-cycles—this will be explicited in proofs and illustrated in Figure 5. In the following, we say that a label state assignment $\mathcal{P}$ is *compatible* with $\widetilde{G}$ if each label is compatible with its state in this graph. Note that looking at the properties of vertices in a label in part gives the label state assignment that it should have: the conflicts are for choosing between Q_f, Q_w^* and, based on the presence or not of an S-vertex, either Q_w or Q_2. This is expected because these states contain the information on a guess on what will later be added to the graph.



(a) Case 1: $|A| \geq 2$, $|B| \geq 2$, and $|A \cap S| + |B \cap S| \geq 1$.

(b) Case 2: $|A| \geq 1$, $|B| \geq 2$, and there is an S-path with endpoints in B.

(c) Case 3: $|A| \geq 1$, $|B| \geq 1$, and there is an S-path connecting a vertex of A to one of B.

Figure 5.: The three cases when a join (depicted in green) creates an S-cycle. The figures illustrate the smallest number of vertices of S required. Thus, up to symmetry, the vertices depicted in red have to be in S while the vertices in white may or may not be in S.

Let us now introduce an auxiliary partially labeled graph which will conveniently represent the connectedness implied by guesses we made so far when assigning labels to label states, while simplifying the manipulation of labels. We point that this auxiliary graph will *not* be computed by the algorithm: it shall only be used in the proofs. Given a labeled graph $\widetilde{G}$ and a label state assignment $\mathcal{P}$, we denote by $H(\widetilde{G}, \mathcal{P})$ the partially labeled graph obtained from $\widetilde{G}$, by conducting the following modifications for each label i:

- if i is in state Q_2 or Q_w, we add a vertex labeled i, connect it to other vertices labeled i, and unlabel these vertices, making the added vertex the only vertex labeled i;
- if i is in state Q_w^*, we add an S-vertex labeled i, connect it to other vertices labeled i, and unlabel these vertices, making the added vertex the only vertex labeled i; and
- if i is in state Q_f, we unlabel vertices labeled i.

Note that in the auxiliary graph, we add vertices that are not part of the original graph. The role of these vertices—for states Q_2, Q_w, and Q_w^*—is to represent the label i as if it was connected (which will eventually be the case as we guessed a later join), as well as manipulating nonempty labels as single vertices: for $\widetilde{G}$ compatible with $\mathcal{P}$, each nonempty label i contains exactly one vertex in $H(\widetilde{G}, \mathcal{P})$, which we call *representative* of i in $H(\widetilde{G}, \mathcal{P})$, and that we denote by $h(i)$.

Recall that, when $\mathcal{P}$ is compatible with $\widetilde{G}$, some connectedness conditions are satisfied by label states. We say that a partially labeled multigraph $\widehat{F}$ *expresses the connectedness* in $H(\widetilde{G}, \mathcal{P})$, for $\widetilde{G}$ and $\mathcal{P}$ compatible, if:

- for each label i, there is at most one vertex labeled i in $\widehat{F}$;

- to every vertex $h(i)$ in $H(\widetilde{G},\mathcal{P})$ corresponds a vertex $r(i)$ labeled i in $\widehat{F}$: we call it the *representative* of label i in $\widehat{F}$, and $r(i)$ is an S-vertex if and only if $h(i)$ is an S-vertex; and
- for any two vertices $h(i), h(j)$ in $H(\widetilde{G},\mathcal{P})$, there exists a $h(i)$–$h(j)$ path in $H(\widetilde{G},\mathcal{P})$ if and only if there exists a $r(i)$–$r(j)$ path in $\widehat{F}$, and there exists a $h(i)$–$h(j)$ S-path in $H(\widetilde{G},\mathcal{P})$ if and only if there exists a $r(i)$–$r(j)$ S-path in $\widehat{F}$.

We are now ready to introduce reduction rules which, when applied on the multigraph $\widehat{F}$ expressing the connectedness in $H(\widetilde{G},\mathcal{P})$, will produce the aforementioned partially labeled forest. The idea behind this forest is that, to check the existence of (S-)paths linking representatives of labels i and j, unlabeled vertices of degree at most two in such (S-)paths may be "contracted" as long as we do not remove all (S-)vertices on these paths. In the following for a partially labeled multigraph $\widehat{F}$, we denote by $\mathrm{Red}(\widehat{F})$ the forest obtained from $\widehat{F}$ by applying the following reduction rules:

- for each nontrivial 2-connected component C, we introduce an unlabeled vertex, call it *central vertex* of C, connect it to vertices of C, and remove all other edges inside the component;
- we iteratively remove unlabeled vertices of degree at most one;
- for each maximal S-path with internal unlabeled vertices of degree two, we replace it by connecting the endpoints to a single new unlabeled S-vertex; and
- for each maximal path with internal unlabeled vertices of degree two that is not an S-path, we replace it by a single edge between its endpoints.

It is easily seen that the produced graph is indeed a forest as the graph of nontrivial 2-connected components of any graph is a tree, and each nontrivial 2-connected component is replaced by a star.

Claim 2.9 $\mathrm{Red}(\widehat{F})$ *has* $\mathcal{O}(k)$ *vertices.*

Lemma 2.5 *Let* $\widetilde{G}$ *be a labeled graph, and* $\mathcal{P}$ *be a label state assignment compatible with* $\widetilde{G}$*. If* $H(\widetilde{G},\mathcal{P})$ *is* S*-cycle-free and* $\widehat{F}$ *expresses the connectedness in* $H(\widetilde{G},\mathcal{P})$*, then* $F = \mathrm{Red}(\widehat{F})$ *expresses the connectedness in* $H(\widetilde{G},\mathcal{P})$*.*

Proof. Note that according to the reduction rules, no labeled vertex is removed nor added to $\widehat{F}$ when computing $F = \mathrm{Red}(\widehat{F})$. Hence the first two conditions on expressing the connectedness in $H(\widetilde{G},\mathcal{P})$ are fulfilled by F, whenever they are by $\widehat{F}$. It remains to show that the existence of (S-)paths between labeled vertices of $\widehat{F}$ is preserved in F. First, note that any path P of $\widehat{F}$ going through a nontrivial 2-connected component C can be turned into a 3-vertex path of F going through the central vertex of the component, and having the first and last vertices of P restricted to C as endpoints. Hence the first reduction rule preserves the desired property. Second, no unlabeled vertex of degree at most one lies on a path linking two labeled vertices. Hence the second reduction rule preserves the desired property, and the same conclusion is

straightforward for the last two reduction rules. □

A *state* of the dynamic programming algorithm is a tuple $(t, F, \mathcal{P})$, where $t \in V(T)$, F is a partially labeled forest, and $\mathcal{P} : [k] \to \mathcal{Q}$ is a label state assignment. We say that $(t, F, \mathcal{P})$ is *admissible* if there exists $X \subseteq V(G)$ such that $\mathcal{P}$ is compatible with $G_t - X$, $H(G_t - X, \mathcal{P})$ is S-cycle-free, and F expresses the connectedness in $H(G_t - X, \mathcal{P})$. Our dynamic programming algorithm will not consider all possible states, but compute a value $d[t, F, \mathcal{P}]$ for some states $(t, F, \mathcal{P})$. We call *reachable* a state that is considered by the algorithm. We will show that reachable states are admissible, that for every $t \in V(T)$, for each $X \subseteq V(G_t)$, if $G_t - X$ is S-cycle-free, then there exists a reachable state $(t, F, \mathcal{P})$ such that $d[t, F, \mathcal{P}] \leq |X|$, and that the optimal value for SFVS on the given instance is the minimum of values $d[r, F, \mathcal{P}]$ where r is the root of the k-expression.

Computations and Correctness First, let us slightly modify our clique-width expression in order to simplify the description of our computations. We double the set of labels, denoting them by $\{1, ..., k, 1', ..., k'\}$, and replace each disjoint union node t with children t_1, t_2 by the following subexpression: $\rho_{1' \to 1}(\dots \rho_{k' \to k}(G_{t_1} \oplus (\rho_{1 \to 1'}(\dots \rho_{k \to k'}(G_{t_2})))))$. This gives the property that in disjoint union nodes, each label is used by at most one of the children nodes.

We now describe the bottom-up computation of reachable states for each possible type of node in the clique-width expression.

Leaf node. If t is a leaf node with $G_t = i(v)$, two cases arise. Either v is deleted which is described by state $(t, F_\varnothing, \mathcal{P}_\varnothing)$ initialized with value $c(v)$, where $F_\varnothing$ is the empty graph, and $\mathcal{P}_\varnothing$ is the function that maps every $i \in [k]$ to $Q_\varnothing$. Otherwise we keep v, which is described by state $(t, F, \mathcal{P})$ where F consists of the isolated vertex v, $\mathcal{P}(i) = Q_1^*$ if $v \in S$, $\mathcal{P}(i) = Q_1$ otherwise, and, for all $j \neq i$, $\mathcal{P}(j) = Q_\varnothing$.

Join node. Let t be a join node with $G_t = \eta_{i \times j}(G_{t'})$. For each reachable state $(t', F', \mathcal{P}')$, we proceed as follows. If the representatives of i and j are connected by an S-path in F', we do nothing. Otherwise, we will construct states $(t, F, \mathcal{P})$ defined in the following cases, depending on $\mathcal{P}'(i)$ and $\mathcal{P}'(j)$, starting with $F := F'$ and $\mathcal{P} := \mathcal{P}'$:

- if one of i and j is in state $Q_\varnothing$, we do not modify F nor $\mathcal{P}$;
- if i and j are in states Q_1 or Q_1^*, we add an edge between the representatives of i and j in F;
- if i and j are in states Q_1 or Q_2, we add an edge between the representatives of i and j in F, and if i or j are in state Q_2 they are allowed to change to Q_f in $\mathcal{P}$, if they do we also unlabel their representative: we enumerate all possibilities here;
- if i and j are in states Q_1^* and Q_w^*, we identify their representative in F: the resulting vertex has its label in state Q_1^*, and the label in state Q_w^* is assigned state Q_f in $\mathcal{P}$; and

- if i and j are in states Q_1 and Q_w, we identify their representative in F: the resulting vertex has its label in state Q_1, and the label in state Q_w is assigned state Q_f in $\mathcal{P}$.

For each such cases, we reduce F and propagate the value $d[t', F', \mathcal{P}']$ to the states $(t, F, \mathcal{P})$, where $\mathcal{P}$ is the modified label state assignment.

Renaming label node. Let t be a renaming label node with $G_t = \rho_{i\to j}(G_{t'})$. For each reachable state $(t', F', \mathcal{P}')$, we construct states $(t, F, \mathcal{P})$ starting with $\mathcal{P} := \mathcal{P}$ and $F := F'$ by first setting $\mathcal{P}(i) = Q_\varnothing$, and proceeding as follows depending on $\mathcal{P}'(i)$ and $\mathcal{P}'(j)$:

- if i and j are in a state among $\{Q_f, Q_1, Q_1^*\}$, we unlabel the representatives of i and j in F', and set $\mathcal{P}(j) = Q_f$;
- if one of i and j is in state $Q_\varnothing$, then either i is in state $Q_\varnothing$ and we do nothing, or j is in state $Q_\varnothing$, we assign it to the other label state, and the vertex of F labeled i is relabeled j;
- if i and j are in state Q_1, and the representatives of i and j are not connected by a path in F', in F, we add an S-vertex labeled j, connect it to these vertices, and unlabel them. Label j is then assigned state Q_w^* in $\mathcal{P}$;
- if one of i and j is in state Q_1^*, the other is in state Q_1 or Q_1^*, and the representatives of i and j are not connected by a path in F', we consider two possibilities depending on whether they will be joined to a vertex of S, or to a vertex of $V(G) \setminus S$. First, in F, we add a new vertex labeled j, connect it to the representatives of i and j, and unlabel the representatives of i and j. Then, if the new vertex is chosen to be in S, j is assigned state Q_w^* in $\mathcal{P}$. Otherwise, j is assigned state Q_w in $\mathcal{P}$;
- if i and j are in states Q_α and Q_β, for $\alpha, \beta \in \{1, 2, w\}$, and the representatives of i and j are not connected by an S-path in F', in F, we identify the representatives of i and j: the resulting vertex is of label j, and j is assigned state Q_δ in $\mathcal{P}$ with $\delta := \max_{1<2<w} \{2, \alpha, \beta\}$;
- if one of i and j is in state Q_1^*, the other is in state Q_w or Q_2, and the representatives of i and j are not connected by a path in F', in F, we add an edge between the representatives of i and j, and the representative of the label in state Q_1^* becomes unlabeled, while the other vertex is given label j. Label j is assigned state Q_w in $\mathcal{P}$; and
- if one of i and j is in state Q_w^*, the other is in state Q_1 or Q_1^*, and the representatives of i and j are not connected by a path in F', in F, we add an edge between the representatives of i and j, and the representative of the label in state Q_1 or Q_1^* becomes unlabeled, while the other vertex is given label j. Label j is assigned state Q_w^* in $\mathcal{P}$.

For each such cases, we reduce F, and we propagate the value $d[t', F', \mathcal{P}']$ to the state $(t, F, \mathcal{P})$, where $\mathcal{P}$ is the modified label state assignment.

Disjoint union node. If t is a disjoint union node with $G_t = G_{t_1} \oplus G_{t_2}$, for each pair of reachable states $(t_1, F_1, \mathcal{P}_1)$, $(t_2, F_2, \mathcal{P}_2)$, since they use disjoint sets of labels, we can simply define $F = F_1 \oplus F_2$. The label state assignment $\mathcal{P}$ is defined by $\mathcal{P}(i) = \mathcal{P}_1(i)$ for $i \in [k]$ and $\mathcal{P}(i') = \mathcal{P}_2(i')$ for $i' \in \{1', \ldots, k'\}$. The value $d[t_1, F_1, \mathcal{P}_1] + d[t_2, F_2, \mathcal{P}_2]$ is propagated to state $(t, F, \mathcal{P})$.

We now prove the correctness of the algorithm.

Lemma 2.6 *For each reachable state $(t, F, \mathcal{P})$, there exists $X \subseteq V(G_t)$ such that $\mathcal{P}$ is compatible with $G_t - X$, $H(G_t - X, \mathcal{P})$ is S-cycle-free, F expresses the connectedness in $H(G_t - X, \mathcal{P})$, and $d[t, F, \mathcal{P}] = c(X)$.*

Proof. We proceed by induction on T. For convenience in the following, let us denote by H and H' the graphs $H(G_t - X, \mathcal{P})$ and $H(G_{t'} - X, \mathcal{P}')$, by $h(i)$ and $h'(i)$ the representative of label i in H and H', and by $r(i)$ and $r'(i)$ the representative of label i in F and F', respectively. By an abuse of notation, we will denote by F both the graph expressing the connectedness in H and its reduced forest $\mathrm{Red}(\widehat{F})$; we may do so as a reduction of F is computed at each transition of our algorithm, and by Lemma 2.5, the obtained forest expresses the connectedness in H.

1. If t is a leaf node, the property is trivial.
2. Let t be a join node with $G_t = \eta_{i \times j}(G_{t'})$ and $(t, F, \mathcal{P})$ be a reachable state. Then there exists a reachable state $(t', F', \mathcal{P}')$ which gave the optimal value to $(t, F, \mathcal{P})$, i.e., such that $d[t, F, \mathcal{P}] = d[t', F', \mathcal{P}']$. By induction hypothesis, there exists X such that $\mathcal{P}'$ is compatible with $G_{t'} - X$, H' is S-cycle-free, F' expresses the connectedness in H', and $d[t', F', \mathcal{P}'] = c(X)$. Hence $d[t, F, \mathcal{P}] = c(X)$. It remains to show that the properties for $\mathcal{P}$, $H(G_t - X, \mathcal{P})$, and F hold for each of the transitions, depending on the label states assigned to i and j in $\mathcal{P}'$. The following observation will be convenient.

   Observation 2.1 *Let $\widetilde{G}$ be a graph, $\mathcal{P}$ be a label state assignment, i be a label such that $\mathcal{P}(i) \in \{Q_2, Q_w\}$, and j be a label such that $\mathcal{P}(j) = Q_w^*$. Then there is an S-cycle containing $h(i)$ in $H(\widetilde{G}, \mathcal{P})$ whenever there is a S-path having elements of i as its endpoints in $\widetilde{G}$, and then there is an S-cycle containing $h(j)$ in $H(\widetilde{G}, \mathcal{P})$ whenever there is a path having elements of j as its endpoints in $\widetilde{G}$.*

   - If one label is in state $Q_\varnothing$, by compatibility of $\mathcal{P}'$ and $G_{t'} - X$, we know that the join added no edge to the graph, i.e., $G_t - X = G_{t'} - X$. By the described transitions, $\mathcal{P} = \mathcal{P}'$ and $F = F'$. Hence $H = H'$, and the conclusion follows.
   - We now deal with the two next cases: these cases do not involve waiting states. From the transition we know that there is no S-path between i and j in F', and since F' expresses the connectedness in H', there is no S-path between representatives of i and j in H'. Let us show that H is S-cycle-free. Consider the graph $H' + A$, where $A = E(G_t - X) \setminus E(G_{t'} - X)$ is the set of edges added by the join. We note that, according to the transitions, either

$\mathcal{P} = \mathcal{P}'$, or $\mathcal{P}'$ differs from $\mathcal{P}$ by assigning one of i and j to Q_f. In both cases, H is a subgraph of $H' + A$. Recall that H' is S-cycle-free, and no S-path connects the representatives of i and j. Hence if $H' + A$ contains an S-cycle, it must be contained in A, which is excluded by the compatibility of $\mathcal{P}'$ and $G_{t'} - X$, more particularly by the constraints induced by compatibility on the two joined labels: a join between labels of Q_1 and Q_1^*, or labels of Q_1 and Q_2, cannot create a cycle. Hence H is S-cycle-free. The claim that $G_t - X$ and $\mathcal{P}$ are compatible follows from Observation 2.1 for the connectivity constraints, and on the fact that either $\mathcal{P} = \mathcal{P}'$ or $\mathcal{P}$ assigns one of i and j to Q_f, for the vertex constraints. Finally, note that any path having its endpoints in i and j after the join has its "connectedness" expressed by F after adding the edge $r(i)r(j)$. This concludes that case.

- The other transitions involve a label in state Q_1 or Q_1^*, joined with a label in a waiting state Q_w or Q_w^*, respectively. W.l.o.g., let us assume that i is in state Q_1 or Q_1^* and j in state Q_w or Q_w^*. Let $v = h'(i)$ and $w = h'(j)$. Consider the graph $H'_{/vw}$, and denote by u the vertex obtained by identification of v and w, where u has label i. We show that $H = H'_{/vw}$. By the transitions, i stays in his state, and j is assigned state Q_f in $\mathcal{P}$. Hence in H, label i consists of the single element v, and label j has no representative. Thus $V_i(H) = V_i(H'_{/vw})$ and $V_j(H) = V_j(H'_{/vw})$. Now, the performed join on $G'_t - X$ adds every edge between v and $V_j(G_{t'})$. Since $N_{H'}(w) = V_j(G_{t'})$, these added edges are exactly those incident to u in $H'_{/vw}$ and that were not already present in H'. We conclude that $H = H'_{/vw}$ as desired. Since H' is S-cycle-free, an S-cycle of H must contain one of the identified vertices. However these vertices are not connected by an S-path, by the first condition expressed in the decription of the transitions. Consequently H is S-cycle-free, and the compatibility of $G_t - X$ and $\mathcal{P}$ then follows from Observation 2.1 for the connectivity constraints, and from the fact that $\mathcal{P}$ only differs on $\mathcal{P}'$ on the fact that it assigns j to Q_f, for the vertex constraints. Clearly, identifying the the representatives in F' preserves expressing the connectedness in $H'_{/vw}$, and the conclusion follows.

3. Let t be a relabel node with $G_t = \rho_{i\to j}(G_{t'})$, and $(t, F, \mathcal{P})$ be a reachable state. Then there exists a reachable state $(t', F', \mathcal{P}')$ which gave the optimal value to $(t, F, \mathcal{P})$, i.e., such that $d[t, F, \mathcal{P}] = d[t', F', \mathcal{P}']$. By induction hypothesis, there exists X such that $\mathcal{P}'$ is compatible with $G_{t'} - X$, H' is S-cycle-free, F' expresses the connectedness in H', and $d[t', F', \mathcal{P}'] = c(X)$. Hence $d[t, F, \mathcal{P}] = c(X)$. We show that the properties for $\mathcal{P}$, $H(G_t - X, \mathcal{P})$, and F hold for each of the transitions, depending on the label types of i and j. We stress that in the following, $G_t - X$ and $G_{t'} - X$ are isomorphic: they only differ by their label assignments.

   - We consider the first case where i and j are in a state among $\{Q_f, Q_1, Q_1^*\}$. Since $\mathcal{P}'$ is compatible with $G_{t'} - X$, and the transition assigns i and j to $Q_\varnothing$ and Q_f, it is easily seen that $\mathcal{P}$ is compatible with $G_t - X$: after the

relabeling, $V_i(G_t - X)$ is empty, and $|V_j(G_t - X)| \geq 2$. Furthermore, as labels i and j have no representative in H, and the representative of i and j in H', if they exist, are vertices of $G_t - X$, we conclude that H and H' are isomorphic. Consequently H is S-cycle-free. At last, as F only differs with F' on i and j being unlabeled, and H has no representative for labels i and j, F expresses the connectedness in H.

- The transition with one of i and j being in state $Q_\varnothing$ either corresponds to doing nothing, or to swapping labels i and j in $G_t - X$ and F: the conclusion follows.
- We now consider the transition with both i and j in state Q_1. Since their representatives in F' are not connected by a path, their representatives in H' are not connected either. Since $G_{t'} - X$ and $\mathcal{P}'$ are compatible, labels i and j are nonempty in $G_{t'} - X$, and hence $|V_j(G_t - X)| \geq 2$. Consequently, j is compatible with state Q_w^* in $G_t - X$ after relabeling. As $V_i(G_t - X)$ is empty, we conclude that $\mathcal{P}$ and $G_t - X$ are compatible. By construction, H contains one representative for label j, and it is an S-vertex. The same holds for F from the transition. Since F' expresses the connectedness in H', and given that $h(j)$ and $r(j)$ are connected to $V_j(G_t - X)$, and to $r'(i)$ and $r'(j)$, respectively, it is easily checked that for any two representatives $h(\alpha), h(\beta)$ in H, there is a $h(\alpha)$–$h(\beta)$ path going through $h(j)$ in H if and only if there is a $r(\alpha)$–$r(\beta)$ path going through $r(j)$ in F. We deduce that F expresses the connectedness in H. At last, H is S-cycle-free because H' is S-cycle-free, and no path of H' links two vertices labeled j in $G_{t'} - X$.
- We now consider the transition with a label in state Q_1^*, and the other in state Q_1 or Q_1^*. Since the representatives of i and j are not connected by a path in F', and since F' expresses the connectedness in H', they are not connected in $G_t - X$. As $V_i(G_{t'} - X)$ and $V_j(G_{t'} - X)$ are nonempty, we deduce that j is compatible with state Q_w^* or Q_w, depending on whether j is a S-vertex or not. As $V_i(G_t - X)$ is empty, we conclude that $G_t - X$ and $\mathcal{P}$ are compatible. The fact that F expresses the connectedness in H follows from the same arguments as in the previous case: the new representative of j in F corresponds exactly to the new representative of j in H. Similarly, we deduce that H is S-cycle-free because H' is S-cycle-free, and no path of H' links two vertices labeled j in $G_{t'} - X$.
- We now consider the transition with labels in states Q_α and Q_β, for $\alpha, \beta \in \{1, 2, w\}$. From the transition, we also know that the representatives of i, j in F' are not connected by an S-path. By the compatibility of $\mathcal{P}'$ and $G_{t'} - X$, labels i and j have a representative in H', and no pair of vertices in such labels in $G_t - X$ are connected by an S-path. Hence after relabeling, no pair of vertices labeled j in $G_t - X$ is connected by an S-path. If the new label state is Q_2, then the previous states were forcing j to contain no S-vertex. If the new label state is Q_w, then one of the previous states was Q_w, which

also requires i to contain at least one S-vertex. Now as $V_i(G_t - X)$ is empty, $\mathcal{P}$ and $G_t - X$ are compatible. Note that the vertices that were put to label j when relabeling are now connected in H. Identifying $r'(i)$ and $r'(j)$ in F' thus produces the desired result of expressing the connectivity in H. At last, H is S-cycle-free because the new connections induced by the relabeling are between vertices that were not connected by S-paths in H'.

- We now consider the transition with a label in state Q_1^*, and the other in state Q_w or Q_2. Representatives of i, j are not connected by a path in F'. As one label is in state Q_1^*, by compatibility of $\mathcal{P}'$ and $G_{t'} - X$, it contains an S-vertex. From F' expressing the connectedness in H', we conclude that j is compatible with Q_w. Since $V_i(G_t - X)$ is empty, we have that $\mathcal{P}$ and $G_t - X$ are compatible. Then, the fact that F expresses the connectedness in H follows from the fact that new connections created by $h(j)$ in H are expressed by an edge between $r'(i)$ and $r'(j)$ in F. At last, H is S-cycle-free because H' is S-cycle-free, and vertices labeled i, j in $G_{t'} - X$ were not connected by a path in H'.
- We now consider the transition with a label in state Q_w^*, and the other in Q_1 or Q_1^*. We in addition know that the representatives of i, j in F' are not connected by a path, and because F' expresses the connectedness in H', the same holds in H'. As one label is assigned Q_w^* in $\mathcal{P}'$, we deduce from compatibility that there is no path in H' between its vertices of $G_{t'} - X$. Since $V_i(G_t - X)$ is empty, we get that $\mathcal{P}$ and $G_t - X$ are compatible. The same argument as in the previous case show that F expresses the connectedness in H, and that H is S-cycle-free: new connections created by $h(j)$ in H are expressed by an edge between $r'(i)$ and $r'(j)$ in F.

4. Let t be a disjoint union node with $G_t = G_{t_1} \oplus G_{t_2}$, and $(t, F, \mathcal{P})$ be a reachable state. Then $(t, F, \mathcal{P})$ was constructed by a transition from some $(t_1, F_1, \mathcal{P}_1)$ and $(t_2, F_2, \mathcal{P}_2)$. By induction hypothesis applied to $(t_1, F_1, \mathcal{P}_1)$ and $(t_2, F_2, \mathcal{P}_2)$ and the fact that the union is disjoint, there exist X_1 and X_2 such that, for $X := X_1 \cup X_2$, $H(G_t - X, \mathcal{P})$ is S-cycle-free, $\mathcal{P}$ is compatible with $G_t - X$, F expresses the connectedness in $H(G_t - X, \mathcal{P})$, and $d[t, F, \mathcal{P}] = c(X_1) + c(X_2) = c(X)$. Hence the conclusion.

□

In the following, we say that graph G is *less connected* than graph G' if $E(G) \subseteq E(G)$. For auxiliary graphs $H = H(G, \mathcal{P})$ and $H' = H(G', \mathcal{P})$ with $V(G) = V(G')$, denote by R and R' their accessibility relation restricted to $V(G) = V(G')$ (paths through other vertices are allowed), we say that H is less connected than H' if $R \subseteq R'$.

Lemma 2.7 *For each $t \in V(T)$, for each $X \subseteq V(G_t)$, and for each label state assignment $\mathcal{P}$ that is compatible with $G_t - X$ and such that $H(G_t - X, \mathcal{P})$ is S-cycle-free, there exists F such that F expresses the connectedness in $H(G_t - X, \mathcal{P})$, $(t, F, \mathcal{P})$ is reachable*

and $d[t, F, \mathcal{P}] \leq |X|$.

Proof. We proceed by induction on T.

1. If t is a leaf node, then the statement holds by construction.
2. Let t be a join node with $G_t = \eta_{i\times j}(G_{t'})$, $X \subseteq V(G_t)$, and $\mathcal{P}$ be a label state assignment such that $\mathcal{P}$ is compatible with $G_t - X$ and $H(G_t - X, \mathcal{P})$ is S-cycle-free. First note that if label ℓ is compatible with $\mathcal{P}(\ell)$ in $G_t - X$, then it is compatible with $\mathcal{P}(\ell)$ in $G_{t'} - X$. This allows us for labels $\ell \in [k]\setminus\{i, j\}$ to keep label state $\mathcal{P}(\ell)$ in the label state assignments $\mathcal{P}'$ and $\mathcal{P}''$ that we will construct, while preserving compatibility. In the following, we will only justify the compatibility for i and j. For the transitions with a label in state $Q_\varnothing$, we immediately conclude that the forest given by the transition expresses the connectedness in $H(G_t - X, \mathcal{P})$ from the induction hypothesis. For other transitions, by comparing paths that appear when constructing F from F' with paths that are in $H(G_t - X, \mathcal{P})$ but not $H(G_{t'} - X, \mathcal{P}')$, we deduce that F expresses the connectedness in $H(G_t - X, \mathcal{P})$, this is already done in detail in the previous lemma. We consider the following cases based on the states of i and j:

   - We first consider the case where $\mathcal{P}(i), \mathcal{P}(j) \in \{Q_2, Q_w, Q_w^*, Q_f\}$. Since $\mathcal{P}$ is compatible with $G_t - X$, $|V_i(G_t - X)| \geq 2$, $|V_j(G_t - X)| \geq 2$. Then as $H(G_t - X, \mathcal{P})$ is S-cycle-free, it must be that none of the vertices in $V_i(G_{t'} - X)$ and $V_j(G_{t'} - X)$ are S-vertices, that no S-path connects a vertex of i to a vertex of j in $G_{t'} - X$, and that no S-path connects two vertices from i, or two vertices from j, in $G_{t'} - X$. Under these constraints, labels i and j are compatible with state Q_2 in $G_{t'} - X$. We set $\mathcal{P}'(i) = \mathcal{P}'(j) = Q_2$, $\mathcal{P}'$ is compatible with $G_{t'} - X$. Let us show that $H(G_{t'} - X, \mathcal{P}')$ is S-cycle-free. Suppose toward a contradiction that there is an S-cycle C in $H(G_{t'} - X, \mathcal{P}')$. Since C is not a subgraph of $H(G_t - X, \mathcal{P})$, it must contain at least one of the representatives of i and j. Suppose w.l.o.g. that C contains the representative i. Then, if C contains an element of $V_j(G_{t'} - X)$, we deduce the existence of an S-path connecting i and j in $H(G_{t'} - X, \mathcal{P}')$, which can be completed to form an S-cycle in $H(G_t - X, \mathcal{P})$ using one of the edges added by the join operation. This contradicts the fact that $H(G_t - X, \mathcal{P})$ is S-cycle-free. If C does not contain any vertex from $V_j(G_{t'} - X)$, replacing the representative of i by an arbitrary vertex in $V_j(G_{t'} - X)$ also yields an S-cycle in $H(G_t - X, \mathcal{P})$, and leads to the same contradiction. We conclude that $H(G_{t'} - X, \mathcal{P}')$ is S-cycle-free, as desired.

     By induction hypothesis with, there exists F' such that F' expresses the connectedness in $H(G_t - X, \mathcal{P})$, $(t', F', \mathcal{P}')$ is reachable, and $d[t', F', \mathcal{P}'] \leq |X|$. Since the join connected vertices of $V_i(G_{t'} - X)$ to vertices of $V_j(G_{t'} - X)$, $G_t - X$ would not be compatible with $\mathcal{P}$ if $\mathcal{P}(i) = Q_w^*$ or $\mathcal{P}(j) = Q_w^*$. Since $V_i(G_t - X)$ and $V_j(G_t - X)$ are of size at least two, i and j are assigned Q_2 or Q_f in $\mathcal{P}$. Both cases are covered by the transitions. We deduce that there

exists F such that F expresses the connectedness in $H(G_t - X, \mathcal{P})$, $(t, F, \mathcal{P})$ is reachable, and that $d[t, F, \mathcal{P}] \leq d[t', F', \mathcal{P}'] \leq |X|$.

- If $\mathcal{P}(i) = Q_\varnothing$ or $\mathcal{P}(j) = Q_\varnothing$, then $G_t - X = G_{t'} - X$ from compatibility and join definition. So we can apply induction hypothesis with $\mathcal{P}$ and follow the transition corresponding to $Q_\varnothing$ to conclude.
- If $\mathcal{P}(i), \mathcal{P}(j) \in \{Q_1, Q_1^*\}$, $\mathcal{P}$ is compatible with $G_{t'} - X$ and $H(G_{t'} - X, \mathcal{P}) \subseteq H(G_t - X, \mathcal{P})$ so $H(G_t - X, \mathcal{P})$ contains no S-cycle and no S-path between the representatives of i and j. The induction hypothesis applied with $\mathcal{P}$ provides F' such that F' expresses the connectedness in $H(G_{t'} - X, \mathcal{P}')$, $(t', F', \mathcal{P})$ is reachable and $d[t', F', \mathcal{P}] \leq |X|$. Since $H(G_{t'} - X, \mathcal{P})$ contains no S-path between representatives of i and j, and F' expresses the connectedness in $H(G_{t'} - X, \mathcal{P})$, we can follow the transition to get F such that F expresses the connectedness in $H(G_t - X, \mathcal{P})$, $(t, F, \mathcal{P})$ is reachable and $d[t, F, \mathcal{P}] \leq d[t', F', \mathcal{P}'] \leq |X|$.
- If $\{\mathcal{P}(i), \mathcal{P}(j)\} = \{Q_1, Q_f\}$, w.l.o.g. let us assume that $\mathcal{P}(i) = Q_1$ and $\mathcal{P}(j) = Q_f$. We define $\mathcal{P}'$ and $\mathcal{P}''$ obtained from $\mathcal{P}$ with $\mathcal{P}'(j) = Q_w$ and $\mathcal{P}''(j) = Q_2$.

  If label j contains no S-vertex in $G_{t'} - X$ then it is compatible with $\mathcal{P}''$, and $H(G_{t'} - X, \mathcal{P}'')$ is S-cycle-free because $H(G_t - X, \mathcal{P})$ is S-cycle-free and we can transform paths of $H(G_{t'} - X, \mathcal{P}')$ as described in the key observation. By applying the induction hypothesis with $\mathcal{P}''$, we get F'' such that F'' expresses the connectedness in $H(G_{t'} - X, \mathcal{P}'')$, $(t', F'', \mathcal{P}'')$ is reachable and $d[t', F'', \mathcal{P}''] \leq |X|$. We can then follow the transition that accepts types Q_1 and Q_2 and conclude.

  Otherwise, $G_{t'} - X$ is compatible with $\mathcal{P}'$ and $H(G_{t'} - X, \mathcal{P}')$ is S-cycle-free because $H(G_t - X, \mathcal{P})$ is S-cycle-free and we can transform paths of $H(G_{t'} - X, \mathcal{P}')$ as described in the key observation. By applying the induction hypothesis with $\mathcal{P}'$, we get F' such that F' expresses the connectedness in $H(G_{t'} - X, \mathcal{P}')$, $(t', F', \mathcal{P}')$ is reachable and $d[t', F', \mathcal{P}'] \leq |X|$. We can then follow the transition that accepts types Q_1 and Q_w and conclude.
- If $\{\mathcal{P}(i), \mathcal{P}(j)\} = \{Q_1^*, Q_f\}$, w.l.o.g. let us assume that $\mathcal{P}(i) = Q_1^*$ and $\mathcal{P}(j) = Q_f$. Consider $\mathcal{P}'$ obtained from $\mathcal{P}$ by setting $\mathcal{P}'(j) = Q_w^*$. Since $H(G_t - X, \mathcal{P})$ is S-cycle-free, we get that $\mathcal{P}'$ is compatible with $G_t - X$ and that $H(G_{t'} - X, \mathcal{P}')$ is S-cycle-free. By induction hypothesis, there exists F' such that F' expresses the connectedness in $H(G_{t'} - X, \mathcal{P}')$, $(t, F', \mathcal{P}')$ is reachable and $d[t, F', \mathcal{P}'] \leq |X|$. We can follow the transition for types Q_1^* and Q_w^* and conclude.
- If one of i and j is in state Q_1 and the other is in state Q_w^* or Q_w, the newly added path with endpoints in the label in state Q_w^* makes it impossible for $\mathcal{P}$ to be compatible with $G_t - X$.
- If $\{\mathcal{P}(i), \mathcal{P}(j)\} = \{Q_1, Q_2\}$, we apply the induction hypothesis with $\mathcal{P}$ and follow the transition for types Q_1 and Q_2 to conclude.

- If one of i and j is in state Q_1^* and the other is in state Q_2, Q_w or Q_w^*, then the join adds an S-path in $G_t - X$ with endpoints in the label in state Q_2, Q_w or Q_w^* which makes it impossible for $\mathcal{P}$ to be compatible with $G_t - X$.

3. Let t be a renaming label node with $G_t = \rho_{i\to j}(G_{t'})$, let $X \subseteq V(G_t)$ and $\mathcal{P} \in \mathcal{Q}^k$ such that $\mathcal{P}$ is compatible with $G_t - X$ and $H(G_t - X, \mathcal{P})$ is S-cycle-free. Note that $\mathcal{P}(i) = Q_\varnothing$ by compatibility. In the following, by comparing paths that appear when constructing F from F' with paths that are in $H(G_t - X, \mathcal{P})$ but not $H(G_{t'} - X, \mathcal{P}')$, we deduce that F expresses the connectedness in $H(G_t - X, \mathcal{P})$, this is already done in detail in the previous lemma. We consider the following cases based on $\mathcal{P}(j)$:
   - If $\mathcal{P}(j) = Q_f$, then consider $\mathcal{P}'$ where $\mathcal{P}'(i)$ and $\mathcal{P}'(j)$ are determined by the type of the vertex with the corresponding label if it is unique in $G_{t'} - X$ and are equal to Q_f otherwise. Thus, $\mathcal{P}'$ is compatible with $G_{t'} - X$ and $H(G_{t'} - X, \mathcal{P}')$ is S-cycle-free because only labels change with $H(G_t - X, \mathcal{P})$. By induction hypothesis, we have F' such that F' expresses the connectedness in $H(G_{t'} - X, \mathcal{P}')$, $(t', F', \mathcal{P}')$ is reachable and $d[t', F', \mathcal{P}'] \leq |X|$. We can thus use the corresponding transition and conclude.
   - If $\mathcal{P}(j) = Q_1$ or $\mathcal{P}(j) = Q_1^*$, then only one of i and j contains a vertex in $G_{t'} - X$. We obtain $\mathcal{P}'$ from $\mathcal{P}$ by setting $\mathcal{P}'(i)$ and $\mathcal{P}'(j)$ according to which label contains the vertex (note that one of them will be $Q_\varnothing$). Again, $\mathcal{P}'$ is compatible with $G_{t'} - X$ and $H(G_{t'} - X, \mathcal{P}')$ is S-cycle-free because only labels change with $H(G_t - X, \mathcal{P})$. By induction hypothesis, we have F' such that F' expresses the connectedness in $H(G_{t'} - X, \mathcal{P}')$, $(t', F', \mathcal{P}')$ is reachable and $d[t', F', \mathcal{P}'] \leq |X|$. We can follow the transition associated to type $Q_\varnothing$ and conclude.
   - If $\mathcal{P}(j) = Q_2$, then the vertices labeled i and j in $G_{t'} - X$ cannot contain S-vertices by compatibility. We obtain $\mathcal{P}'$ from $\mathcal{P}$ by setting $\mathcal{P}'(i)$ and $\mathcal{P}'(j)$ to $Q_\varnothing$, Q_1 or Q_2 based on the number of vertices with the corresponding label in $G_{t'} - X$. Again, $\mathcal{P}'$ is compatible with $G_{t'} - X$ and $H(G_{t'} - X, \mathcal{P}')$ is S-cycle-free. There cannot be an S-path between representatives of i and j in $H(G_{t'} - X, \mathcal{P}')$ because $H(G_t - X, \mathcal{P})$ contains no S-cycle but has them connected by an additional path. By induction hypothesis, we get F' such that F' expresses the connectedness in $H(G_{t'} - X, \mathcal{P}')$, $(t', F', \mathcal{P}')$ is reachable and $d[t', F', \mathcal{P}'] \leq |X|$. We can then follow a transition and conclude.
   - If $\mathcal{P}(j) = Q_w$, then consider $\mathcal{P}'$ obtained from $\mathcal{P}$ by setting $\mathcal{P}'(i)$ and $\mathcal{P}'(j)$ to $Q_\varnothing$, Q_1, Q_1^*, Q_2 or Q_w based on the vertices with corresponding label in $G_{t'} - X$. $\mathcal{P}'$ is compatible with $G_{t'} - X$, $H(G_{t'} - X, \mathcal{P}')$ is S-cycle-free because it is less connected than $H(G_t - X, \mathcal{P})$ and $H(G_{t'} - X, \mathcal{P}')$ cannot have an S-path between representatives of i and j because $H(G_t - X, \mathcal{P})$ is S-cycle-free. By induction hypothesis, we get F' such that F' expresses the connectedness in $H(G_{t'} - X, \mathcal{P}')$, $(t', F', \mathcal{P}')$ and $d[t', F', \mathcal{P}'] \leq |X|$. We can then follow a transition and conclude.

- Finally, if $\mathcal{P}(j) = Q_w^*$, then consider $\mathcal{P}'$ obtained from $\mathcal{P}$ by setting $\mathcal{P}'(i)$ and $\mathcal{P}'(j)$ to $Q_\varnothing$, Q_1, Q_1^* or Q_w^* based on the vertices with corresponding label in $G_{t'} - X$. $\mathcal{P}'$ is compatible with $G_{t'} - X$ because a path between its vertices labeled i or j would contradict $H(G_t - X, \mathcal{P})$ being S-cycle-free, $H(G_{t'} - X, \mathcal{P}')$ is S-cycle-free because it is less connected than $H(G_t - X, \mathcal{P})$ and there is no path between the representatives of i and j in $H(G_{t'} - X, \mathcal{P}')$ because it would also contradict $H(G_t - X, \mathcal{P})$ being S-cycle-free. By induction hypothesis, we get F' such that F' expresses the connectedness in $H(G_{t'} - X, \mathcal{P}')$, $(t', F', \mathcal{P}')$ is reachable and $d[t', F', \mathcal{P}'] \leq |X|$. We can then follow a transition and conclude.

4. Let t be a disjoint union node with $G_t = G_{t_1} \oplus G_{t_2}$, let $X \subseteq V(G_t)$ and $\mathcal{P} \in \mathcal{Q}^k$ such that $\mathcal{P}$ is compatible with $G_t - X$ and $H(G_t - X, \mathcal{P})$ is S-cycle-free. We define $X_1 = X \cap V(G_{t_1})$ and $X_2 = X \cap V(G_{t_2})$, and construct $\mathcal{P}_1$ and $\mathcal{P}_2$ by taking the value of $\mathcal{P}$ for their respective used labels, and taking value $Q_\varnothing$ otherwise. We immediately get that, for $i \in \{1, 2\}$, $\mathcal{P}_i$ is compatible with $G_{t_i} - X_i$ and $H(G_{t_i} - X_i, \mathcal{P}_i)$ because used labels are disjoint. By induction hypothesis, we get F_1 and F_2 such that F_1 and F_2 express the connectedness in $H(G_{t_1} - X, \mathcal{P}_1)$ and $H(G_{t_2} - X, \mathcal{P}_2)$ respectively, $(t_1, F_1, \mathcal{P}_1)$ and $(t_2, F_2, \mathcal{P}_2)$ are reachable, $d[t_1, F_1, \mathcal{P}_1] \leq |X_1|$ and $d[t_2, F_2, \mathcal{P}_2] \leq |X_2|$. We can follow the transition and conclude.

□

Lemma 2.8 *The minimum value of a reachable state of the root of the expression is equal to the minimum size of a subset feedback vertex set.*

Proof. For X^* a subset feedback vertex set of minimum size, we define $\mathcal{P}^*$ by $\mathcal{P}^*(i) = Q_\varnothing$ is $G_r - X^*$ has no vertex labeled i, $\mathcal{P}^*(i) = Q_f$ if there are at least 2 vertices labeled i in $G_r - X^*$ and $\mathcal{P}^*(i) = Q_1$ or $\mathcal{P}^*(i) = Q_1^*$ based on the type of vertex labeled a if there is exactly one. $\mathcal{P}^*$ is compatible with $G_r - X^*$ and $H(G_r - X, \mathcal{P}) = G_r - X$ contains no S-cycle. By Lemma 2.7, there is a state $(r, F^*, \mathcal{P})$ such that $d[r, F^*, \mathcal{P}] \leq |X^*|$. By Lemma 2.6, every reachable state has a value that corresponds to a subset feedback vertex set so by minimality of $|X^*|$, we have $d[r, F^*, \mathcal{P}] = |X^*|$ and $d[r, F^*, \mathcal{P}^*] \leq d[r, F, \mathcal{P}]$ for any other reachable state $(r, F, \mathcal{P})$. □

Proof of Theorem 3.2. From Lemma 2.8 we know that the algorithm computes the solution to SFVS on G, For each $t \in T$, the number of states $(t, F, \mathcal{P})$ is $2^{\mathcal{O}(k \log k)}$ since the forests have $\mathcal{O}(k)$ vertices by claim 2.9. Note that enumerating pairs of states for disjoint union nodes takes time $\left(2^{\mathcal{O}(k \log k)}\right)^2 = 2^{\mathcal{O}(k \log k)}$. Summing for all nodes of T (at most n), we conclude that the values for all states can be computed in time $2^{\mathcal{O}(k \log k)} \cdot n$. □

Odd Cycle Transversal

We describe a dynamic programming algorithm for Odd Cycle Transversal parameterized by clique-width. In the following, we consider a k-labeled graph G, and assume that it comes with a k-expression of size n and its associated tree T with root $r \in V(T)$.

A *state* in our dynamic programming algorithm is a tuple $(t, \mathbf{x})$ where t is a node of T, and $\mathbf{x} \in \{0,1,2,3\}^k$ is a vector. It is called *admissible* if there exists a triple (X, A, B) with X an OCT of G_t and (A, B) an ordered bipartition of $G_t - X$, called *compatible* with $(t, \mathbf{x})$, such that

- $V_i(G_t) \subseteq X$ for every $i \in [k]$ with $\mathbf{x}[i] = 0$;
- $V_i(G_t - X) \subseteq A$ for every $i \in [k]$ with $\mathbf{x}[i] = 1$; and
- $V_i(G_t - X) \subseteq B$ for every $i \in [k]$ with $\mathbf{x}[i] = 2$.

We call *category* of label i with respect to vector $\mathbf{x}$ the integer $\mathbf{x}[i]$. Labels from category 3 may contain any vertex of G_t. Hence X is not characterized by $\mathbf{x}$. In particular, two OCTs for a same graph G_t may witness the admissibility of a single state $(t, \mathbf{x})$: in $d[t, \mathbf{x}]$ will be stored the size of one such OCT. We will later prove that the minimum size of an OCT for G is found by iterating through the different values of states $(r, \mathbf{x})$.

For convenience in the following, let us define the poset on ground set $\{0,1,2,3\}$ with comparabilities $0 < 1$, $0 < 2$, $1 < 3$, and $2 < 3$, and consider the *supremum* and *infimum* operations $\vee$ and $\wedge$ on such a poset: $a \vee b$ denotes the unique minimal element c such that both $a \leq c$ and $b \leq c$, $a \wedge b$ denotes the unique element c' such that both $a \geq c'$ and $b \geq c'$. We extend the order relation to pairs of vectors $\mathbf{x}, \mathbf{y} \in \{0,1,2,3\}$ by defining $\mathbf{x} \leq \mathbf{y}$ if $\mathbf{x}[i] \leq \mathbf{x}[j]$ for all $i, j \in [k]$. Similarly, the supremum of pairs of vectors $\mathbf{x}, \mathbf{y} \in \{0,1,2,3\}^k$ is obtained by performing the supremum on each of their coordinates.

We describe a bottom-up computation of the values of states, handling each type of node in T. First, we initialize $d[t, \mathbf{x}] = +\infty$ for every node $t \in V(T)$ and vector $\mathbf{x} \in \{0,1,2,3\}^k$. We then proceed to the computation of some states, called *reachable* in the remaining of the section, going trough the following transitions:

- **Leaf node.** If t is a leaf node with $G_t = i(v)$ for some vertex v and label $i \in [k]$, we set $d[t, \{0\}^k] = c(v)$, $d[t, \mathbf{x}] = 0$, and $d[t, \mathbf{y}] = 0$, where $\mathbf{x}[i] = 1$, $\mathbf{y}[i] = 2$, and $\mathbf{x}[j] = \mathbf{y}[j] = 0$ for $i \neq j \in [k]$. These states correspond to either placing v into an OCT we are constructing, or placing it into one of the two sides of a bipartition we are setting.
- **Disjoint union node.** Let t be a disjoint union node with children t_1, t_2 such that $G_t = G_{t_1} \oplus G_{t_2}$. For each $\mathbf{x} \in \{0,1,2,3\}^k$, we put

$$d[t, \mathbf{x}] = \min_{\mathbf{x}_1 \vee \mathbf{x}_2 \leq \mathbf{x}} d[t_1, \mathbf{x}_1] + d[t_2, \mathbf{x}_2].$$

- **Join node.** Let t be a join node with child t' such that $G_t = \eta_{i \times j}(G_{t'})$. We do not modify states but only let through those whose join can only preserve the bipartition induced by categories 1 and 2 in $G_{t'}$. Such states are those $(t', \mathbf{x})$ which

satisfy $\mathbf{x}[i] \wedge \mathbf{x}[j] = 0$, for which we put

$$d[t, \mathbf{x}] = d[t', \mathbf{x}].$$

- **Renaming label node.** Let t be a renaming label node with child t' such that $G_t = \rho_{i \to j}(G_{t'})$. For each reachable state $(t', \mathbf{x}')$ we create a vector $\mathbf{x}$ by setting $\mathbf{x}[i] = 0$, $\mathbf{x}[j] = \mathbf{x}'[i] \vee \mathbf{x}'[j]$, and $\mathbf{x}[\ell] = \mathbf{x}'[\ell]$ for $\ell \notin \{i, j\}$. We then put

$$d[t, \mathbf{x}] \leftarrow d[t', \mathbf{x}'].$$

We show in the following lemma that reachable states are admissible and contain the value of a compatible OCT.

Lemma 2.9 *For every reachable state $(t, \mathbf{x})$ there exists a compatible triple (X, A, B) such that $c(X) = d[t, \mathbf{x}]$.*

Proof. We proceed by induction on T.

Let t be a leaf node with $G_t = i(v)$ for some vertex v and label $i \in [k]$, then three different states are initialized depending on whether v is placed in the OCT, or in one of the two sides of the bipartition. Each of these states satisfies the conclusion.

Let t be a disjoint union node with children t_1, t_2 such that $G_t = G_{t_1} \oplus G_{t_2}$, and $(t, \mathbf{x})$ be a reachable state. By the described transitions, there exist reachable states $(t_1, \mathbf{x}_1)$ and $(t_2, \mathbf{x}_2)$ such that $\mathbf{x} \geq \mathbf{x}_1 \vee \mathbf{x}_2$ and $d[t, \mathbf{x}] = d[t_1, \mathbf{x}_1] + d[t_2, \mathbf{x}_2]$. By induction hypothesis, for $i \in [2]$, there exist triples (X_i, A_i, B_i) compatible with $(t_i, \mathbf{x}_i)$ such that $c(X_i) = d[t_i, \mathbf{x}_i]$. As $G_t = G_{t_1} \oplus G_{t_2}$, $X = X_1 \cup X_2$ is an OCT of G_t and $(A_1 \cup A_2, B_1 \cup B_2)$ defines a bipartition of $G_t - X$. Furthermore as $\mathbf{x} \geq \mathbf{x}_1 \vee \mathbf{x}_2$, it is straightforward to check that $(X, A_1 \cup A_2, B_1 \cup B_2)$ are compatible with $(t, \mathbf{x})$. We conclude noting that $c(X) = c(X_1) + c(X_2) = d[t_1, \mathbf{x}_1] + d[t_2, \mathbf{x}_2] = d[t, \mathbf{x}]$.

Let t be a join node with child t' such that $G_t = \eta_{i \times j}(G_{t'})$, and $(t, \mathbf{x})$ be a reachable state. By the described transitions $(t', \mathbf{x})$ is also a reachable state satisfying one of the conditions $\mathbf{x}[i] = 0$, or $\mathbf{x}[j] = 0$, or ($\mathbf{x}[i] = 1$ and $\mathbf{x}[j] = 2$), or ($\mathbf{x}[i] = 2$ and $\mathbf{x}[j] = 1$). Furthermore, $d[t, \mathbf{x}] = d[t', \mathbf{x}]$. By induction hypothesis, there exists a triple (X, A, B) compatible with $(t', \mathbf{x})$ such that $c(X) = d[t', \mathbf{x}]$, and hence such that $c(X) = d[t, \mathbf{x}]$. According to the transition, as (X, A, B) and $(t', \mathbf{x})$ are compatible, every edge added to $G_{t'}$ when performing the join operation either has an endpoint in X, or one in A and the other in B. We deduce that X is also an OCT of G_t, and that (A, B) also defines a bipartition of G_t. Hence, (X, A, B) is compatible with $(t, \mathbf{x})$.

Let t be a renaming label node with child t' such that $G_t = \rho_{i \to j}(G_{t'})$, and let $(t, \mathbf{x})$ be a reachable state. By the described transitions there exists $\mathbf{x}'$ such that $\mathbf{x}[i] = 0$, $\mathbf{x}[j] = \mathbf{x}'[i] \vee \mathbf{x}'[j]$, and $\mathbf{x}[\ell] = \mathbf{x}'[\ell]$ for $\ell \notin \{i, j\}$, and such that $d[t, \mathbf{x}] = d[t', \mathbf{x}']$. By induction hypothesis, there exists a triple (X, A, B) compatible with $(t', \mathbf{x}')$ such that $c(X) = d[t', \mathbf{x}']$, and hence such that $c(X) = d[t, \mathbf{x}]$. As G_t and $G_{t'}$ are isomorphic, X is also an OCT of G_t, and (A, B) also defines a bipartition of $G_t - X$. As $\mathbf{x}[i] = 0$

and $\mathbf{x}[j] = \mathbf{x}'[i] \vee \mathbf{x}'[j]$ are the only modifications made to $\mathbf{x}'$, it is easy to check that (X, A, B) is compatible with $(t, \mathbf{x})$. □

Lemma 2.10 *For every node $t \in V(T)$, for every OCT X of G_t and bipartition (A, B) of $G_t - X$, there exists $\mathbf{x} \in \{0, 1, 2, 3\}^k$ such that $(t, \mathbf{x})$ is reachable, compatible with (X, A, B), and satisfies $c(X) \geq d[t, \mathbf{x}]$.*

Proof. We proceed by induction on T.

Let t be a leaf node with $G_t = i(v)$ for some vertex v and label $i \in [k]$, then three different cases arise depending on whether v is placed in the OCT, or in one of the two sides of the bipartition. For each of these cases, a compatible state is constructed and a value satisfying $c(X) \geq d[t, \mathbf{x}]$ is initialized.

Let t be a disjoint union node with children t_1, t_2 such that $G_t = G_{t_1} \oplus G_{t_2}$, X be an OCT of G_t, and (A, B) be a bipartition of $G_t - X$. As $G_t = G_{t_1} \oplus G_{t_2}$, $X_1 = X \cap V(G_{t_1})$ and $X_2 = X \cap V(G_{t_2})$ are OCTs of G_{t_1} and G_{t_2}, and the ordered pairs $(A_1 = A \cap V(G_{t_1}), B_1 = B \cap V(G_{t_1}))$ and $(A_2 = A \cap V(G_{t_2}), B_2 = B \cap V(G_{t_2}))$ define bipartitions of $G_{t_1} - X_1$ and $G_{t_2} - X_2$, respectively. By induction hypothesis, there exist reachable states $(t_1, \mathbf{x}_1)$ and $(t_2, \mathbf{x}_2)$ with $d[t_1, \mathbf{x}_1] \leq c(X_1)$ and $d[t_2, \mathbf{x}_2] \leq c(X_2)$ that are compatible with (X_1, A_1, B_1) and (X_2, A_2, B_2), respectively. By the described transitions, the state $(t, \mathbf{x})$ defined by $\mathbf{x} := \mathbf{x}_1 \vee \mathbf{x}_2$ is computed, and it is compatible with (X, A, B). Since $c(X) = c(X_1) + c(X_2)$ we deduce $d[t, \mathbf{x}] \leq d[t_1, \mathbf{x}_1] + d[t_2, \mathbf{x}_2] \leq c(X)$ as desired.

Let t be a join node with child t' such that $G_t = \eta_{i \times j}(G_{t'})$, X be an OCT of G_t, and (A, B) be a bipartition of $G_t - X$. Since $E(G_{t'} - X) \subseteq E(G_t - X)$, X is also an OCT of $G_{t'} - X$, and (A, B) is also a bipartition of $G_{t'} - X$. By induction hypothesis, there exists a reachable state $(t', \mathbf{x})$ that is compatible with (X, A, B), and which satisfies $d[t', \mathbf{x}] \leq c(X)$. Since (A, B) is a bipartition of $G_t - X$, the edges that were added to $G_{t'}$ by the join operation either had an endpoint in X, or one in A and the other in B. Hence, either one of the labels i and j is empty (category 0), or i and j are on distinct sides of the partition. We conclude to a transition from $(t', \mathbf{x})$ to $(t, \mathbf{x})$, and hence $d[t, \mathbf{x}] \leq d[t', \mathbf{x}] \leq c(X)$.

Let t be a renaming label node with child t' such that $G_t = \rho_{i \to j}(G_{t'})$, X be an OCT of G_t and (A, B) be a bipartition of $G_t - X$. Since G_t and $G_{t'}$ are isomorphic, X is also an OCT of $G_{t'}$, and (A, B) is also a bipartition of $G_{t'} - X$. By induction hypothesis, there exists a reachable state $(t', \mathbf{x}')$ that is compatible with (X, A, B), and which satisfies $d[t', \mathbf{x}'] \leq c(X)$. The transition from this state gives us a reachable state $(t, \mathbf{x})$ such that $d[t, \mathbf{x}] \leq d[t', \mathbf{x}'] \leq c(X)$. The obtained state $(t, \mathbf{x})$ is still compatible with (A, B), as its modification exactly translates the change of label of the concerned vertices. □

We deduce the correctness of our algorithm.

Lemma 2.11 *The minimum size of an OCT for G is equal to the minimum, among $\mathbf{x} \in \{0,1,2,3\}^k$, of $d[r,\mathbf{x}]$.*

Proof. Let X^* be an OCT of G of minimum size, and (A,B) be a bipartition of $G - X^*$. By Lemma 2.10, there exists a reachable state $(r,\mathbf{x})$ compatible with (X^*,A,B) such that $c(X^*) \geq d[r,\mathbf{x}]$. By Lemma 2.9, for every other state $(r,\mathbf{y})$, either $(r,\mathbf{y})$ is not reachable and $d[r,\mathbf{y}] = +\infty$, or it is reachable and there exists an OCT Y of G such that $c(Y) = d[r,\mathbf{y}]$. We conclude by noting that $c(X^*) \leq c(Y)$ for every such set Y. □

We conclude the section with the time analysis.

Lemma 2.12 *There are at most $\mathcal{O}(4^k \cdot n)$ possible states in total, and computing all reachable states takes $\mathcal{O}(4^k \cdot k \cdot n)$ time.*

Proof. There are 4^k states per node of the k-expression, hence there are at most $\mathcal{O}(4^k \cdot n \cdot k^2)$ possible states in total in the algorithm.

The computation of a leaf node takes $\mathcal{O}(1)$ time, and the computations of a join or renaming label node is done within $\mathcal{O}(4^k)$ time. Let us show that the computation of a disjoint union node t takes $\mathcal{O}(4^k \cdot k)$ time. We first compute $\delta_i(\mathbf{x}) = \min\{d[t_i,\mathbf{x}'] : \mathbf{x}' \leq \mathbf{x}\}$ for $i \in [2]$ and every $\mathbf{x} \in \{0,1,2,3\}^k$. This is done in $\mathcal{O}(4^k \cdot k)$ time starting from $\delta_i(\{0\}^k) = d[t_i,\{0\}^k]$, and computing the other values from bottom to top, observing that, by monotonicity of δ_i, $\delta_i(\mathbf{x}) = \min\{d[t_i,\mathbf{x}], \delta_i(\mathbf{x}') \colon \mathbf{x}' \in \mathrm{pred}(\mathbf{x})\}$, where $\mathrm{pred}(\mathbf{x})$ denote the immediate predecessors of $\mathbf{x}$ in $\{0,1,2,3\}^k$ with respect to $\leq$. Now, note that by definition of the supremum, we have the following identity:

$$\min_{\mathbf{x}_1 \vee \mathbf{x}_2 \leq x} d[t_1,\mathbf{x}_1] + d[t_2,\mathbf{x}_2] = \min_{\mathbf{x}_1 \leq \mathbf{x}} d[t_1,\mathbf{x}_1] + \min_{\mathbf{x}_2 \leq \mathbf{x}} d[t_2,\mathbf{x}_2].$$

We deduce that $d[t,\mathbf{x}] = \delta_1(\mathbf{x}) + \delta_2(\mathbf{x})$.

Summing up, our algorithm runs in time $\mathcal{O}(4^k \cdot k \cdot n)$. □

The lower bound Our construction follows the line of the ones for Independent Set and Odd Cycle Transversal in [LMS11]. Let φ be an instance of SAT on n variable and m clauses. Note that we may assume that n is even, as otherwise we can add a single extra dummy variable to the instance with no impact on the satisfiability of φ. We denote by ρ the sum of the sizes of the clauses in φ, i.e., the total number of occurrences of literals in φ.

Let us first describe a gadget introduced in [LMS11, Section 7]. For two vertices u, v we call *arrow from u to v* and denote by $A(u,v)$ the graph consisting of a path $ua_1a_2a_3v$ on five vertices, together with the additional four vertices b_1, b_2, b_3, b_4 and eight edges $ub_1, b_1a_1, a_1b_2, b_2a_2, a_2b_3, b_3a_3, a_3b_4, b_4v$. This graph is illustrated in Figure 6. Note that such a graph has a unique smallest OCT of size 2, which is $\{a_1, a_3\}$: we call *passive OCT*

of the arrow such an OCT. In $A(u,v) \setminus \{u\}$, $\{a_2, v\}$ is a smallest OCT: we call it *active OCT* of the arrow.

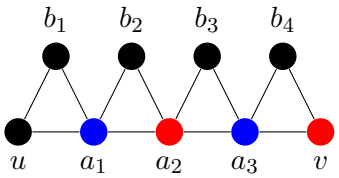


Figure 6.: The arrow $A(u,v)$. Vertices in blue form the passive OCT of $A(u,v)$, and those in red form the active OCT of $A(u,v) \setminus \{u\}$.

For a clause $C = \{\ell_1, \ldots, \ell_t\}$ with t an odd integer greater than 2, let us denote by $\widehat{C}$ the graph consisting of a simple cycle on vertex set $\{c_1, \ldots, c_t\}$ with c_i representing literal ℓ_i, and with edges linking c_i's in the natural way: $c_i c_{i+1}$ for every $i \in [t-1]$, and $c_t c_1$. For a clause C of even size $t \geq 2$, we proceed similarly but subdivide an arbitrary edge so that the obtained graph $\widehat{C}$ is an odd cycle. For a clause C of size one, we simply create a triangle $\widehat{C}$, one of its vertices representing the unique literal of the clause.

We construct a graph G_1 as follows. We create n paths $P_1, \ldots, P_n$ each of length $2m$, and denote by $p_{i,j}$ the j^{th} vertex on path P_i. Each P_i will be referred to as the path of variable x_i in the following. We add "crossing" edges as follows: for every odd $i \in [n-1]$ and even $j \in [2m-1]$, we add edges $p_{i,j}p_{i+1,j+1}$ and $p_{i+1,j}p_{i,j+1}$; this is illustrated in Figure 7. Crossing edges will serve no other purpose than allowing to construct these paths with $n/2$ labels. At this stage, paths of G_1 may be seen as rows representing variables of φ, and vertices $\bigcup_{i\in[n]}\{p_{i,2j}, p_{i,2j-1}\}$ for $j \in [m]$ may be seen as columns grouped two by two representing the two possible values variables x_i may take in clause C_j. For every clause $C_j = \{\ell_1, \ldots, \ell_{|C_j|}\}$ in φ, we add cycle $\widehat{C}_j$ to G_1, and make the following connections. If variable x_i appears positively in C_j on position p, i.e., $\ell_p = x_i$, then we add arrow $A(p_{i,2j-1}, c_p)$ to G_1. If x_i appears negatively in C_j on position p, then we add arrow $A(p_{i,2j}, c_p)$ to G_1. This concludes the construction of the graph G_1.

We now consider $n+1$ copies of G_1 that we name $G_1, \ldots, G_{n+1}$. Let us denote by $P_1^k, \ldots, P_n^k$ the n paths of G_k, $p_{i,j}^k$ the j^{th} vertex of P_i^k, and C^k the cycle representing clause C in G_k, for every $k \in [n+1]$. We link consecutive graphs $G_k, G_{k+1}, k \in [n]$ by adding edges $p_{i,2m}^k p_{i,1}^{k+1}$ for every $i \in [n]$, and denote by P_i^* the resulting path of length $2m \cdot (n+1)$ obtained by concatenating $P_i^1, \ldots, P_i^{n+1}$ in such a way. In addition, we add crossing edges $p_{i,2m}^k p_{i+1,1}^{k+1}$ and $p_{i+1,2m}^k p_{i,1}^{k+1}$ for every $k \in [n]$ and odd $i \in [n]$. We complete our construction by creating a biclique of bipartition $\{A, B\}$ with A and B of size $\alpha + 1 = (n+1)(nm + 2\rho) + 1$, by connecting every vertex of A to every vertex of P_i^* for odd $i \in [n]$, and by connecting every vertex of B to every vertex of P_i^* for even $i \in [n]$. We call G the obtained graph.

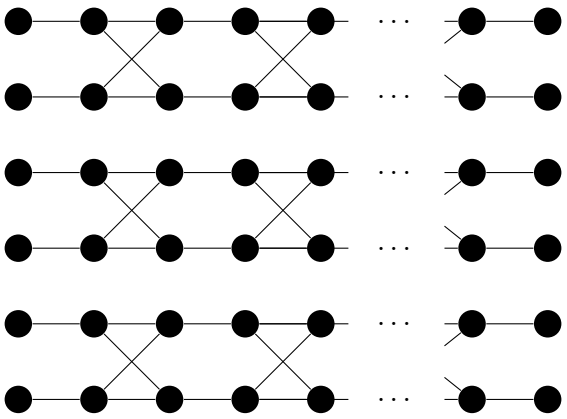


Figure 7.: The crossing edges on paths $P_1, \ldots, P_n$ in G_1.

Lemma 2.13 *If φ is satisfiable, then there exists an OCT of size $\alpha = (n+1)(nm+2\rho)$ in G.*

Proof. Let τ be a satisfying truth assignment of φ. We construct an OCT T of G as follows. If $\tau(x_i)$ is true, then we add $p^k_{i,j}$ to T for every $k \in [n+1]$ and odd $j \in [2m]$. Otherwise we add $p^k_{i,j}$ to T for every $k \in [n+1]$ and even $j \in [2m]$. For each of the $\rho \cdot (n+1)$ arrows $A(u,v)$ in G, we add its active OCT if u is part of T, and its passive OCT otherwise, each of size two. The obtained set T is of size α.

Since each clause C is satisfied by τ, at least one of its literals is satisfied. Thus for each cycle in $\widehat{C}^k, k \in [n+1]$ there exists an arrow $A(u,v)$ having an endpoint u being part of T. Hence that arrow is active and we conclude that v as well is part of T. We conclude that every (odd) cycle $\widehat{C}^1, \ldots, \widehat{C}^{n+1}$ corresponding to C is hit by T. Since T is a vertex cover of the paths $P^*_i, i \in [n]$ every remaining edge in $G - T$ that is not part of a clause cycle nor an arrow is either a crossing edge, or an edge incident to a vertex of A or B. Since these edges do not induce odd cycles, we conclude that T is an OCT of G, as desired. □

Lemma 2.14 *If there exists an OCT of size $\alpha = (n+1)(nm+2\rho)$ in G, then φ is satisfiable.*

Proof. First note that an OCT of G must either contain a vertex cover of each $P^*_i, i \in [n]$ or contain all of A or B. Since $|A| = |B| > \alpha$, we can assume the existence of an OCT T of size α satisfying the first condition. Hence T contains at least half of the vertices of paths $P^*_i, i \in [n]$, that is, at least $nm \cdot (n+1)$ vertices in total. Since we need at least two additional vertices to intersect the odd cycles of every arrow (even if an endpoint of the arrow is part of T), and that there are ρ of them, we conclude that T contains precisely $nm \cdot (n+1)$ vertices from $P^*_1, \ldots, P^*_n$, and $2\rho \cdot (n+1)$ vertices from the arrows. Now, observe that a vertex cover of size k in a path of size $2k$ may contain at most once

two consecutive vertices. Since P_i^*'s are made from $n+1$ repetitions of n paths, there exists $G_j, j \in [n+1]$ for which none of the paths $P_i^j, i \in [n]$ contains consecutive vertices from T. We construct a truth assignment τ for φ out of these paths as follows. For a variable $x_i, i \in [n]$ we set $\tau(x_i)$ to true if the vertices of T coincide with odd vertices of P_i^j, and $\tau(x_i)$ to false otherwise.

Since every clause cycle in G_j has to be hit by T, and T already contains α vertices from $P_1^*, \ldots, P_n^*$ and the arrows, we conclude that each clause cycle in G_j contains at least one incoming arrow $A(u,v)$ with $v \in T$. Hence T contains at least three vertices from that arrow; see Figure 6. Since T intersects $A(u,v) - \{u\}$ on at most two vertices, the only way to hit every cycle of $A(u,v)$ that way is for T to contain $\{u, a_2, v\}$: the arrow is active and T contains u. By construction, the variable corresponding to vertex u satisfies the clause in φ. We conclude to a satisfying truth assignment of φ as desired. □

Lemma 2.15 *The constructed graph G satisfies $\boldsymbol{cw}(G) \leq \boldsymbol{lcw}(G) \leq n/2 + 10$.*

Proof. We describe a construction of G by a linear K-expression with $K = n/2 + 10$ labels. This proves $\mathbf{lcw}(G) \leq K$, and $\mathbf{cw} \leq \mathbf{lcw}$ follows from definition.

During the construction, $n/2$ labels will be used to maintain the current path extremities, grouped two by two. These labels are indexed by $[n/2]$. Two labels will be dedicated to build clause cycles, denoted a and b. No more than five labels will be used to build arrows, and to prolongate current paths extremities. These labels are referred to as *working labels* in the following. At last, two labels will be used for the biclique induced by A and B, and one *final* label indexed by 0 will contain the rest of the graph: no further modifications will be made to vertices once they are put in that label. We first construct the biclique with its two dedicated labels. Then, we will construct the rest of G from G_1 to G_{n+1}, column after column (recall that columns consist of groups of two vertices from each $P_i^*, i \in [n]$), with each column constructed from top to bottom two rows at a time. At each step, four vertices are added to the P_i^*'s, and the gadgets are constructed along the way.

Let us focus on the graph G_1 and assume that we are currently constructing the column described by indices $j, j+1$ for an odd $j \in [2m]$, and are willing to construct rows $i, i+1$ of that column for an odd $i \in [n]$. First, we create vertices $p_{i,j}$ and $p_{i+1,j}$ using one distinct working label for each, call them y_1 and y_2, respectively. If one of $p_{i,j}$ and $p_{i+1,j}$, name it u, participates to an arrow $A(u, c_k)$ linking u to a clause cycle $\widehat{C}$ on vertex $c_k, k \in [|C|]$, we construct that arrow using other working labels. Clearly, three additional labels suffice to proceed with the construction of the arrow (not counting label 0). The vertices of the clause cycles will in fact be constructed by adding arrows one after the other. When creating arrow $A(u, c_k)$, c_k is joined with label a, the vertex labeled a is relabeled 0, and c_k is relabeled a. We point out that the order of the vertices in the cycle does not matter. Note that to this extent, we only need three labels for $\widehat{C}$: two labels a and b for the extremities of paths that constitute the cycle for now, and label 0 for internal vertices in these path (they will not be joined anymore). We point

out that during the construction of the column described by $j, j+1$, only the clause cycle $\widehat{C}$ is constructed. When finishing to build the column, the possible extra vertices that were added to $\widehat{C}$ so that it is of odd size are added by joins between a and a working label, emptying label a, and labeling them a. Once all vertices of $\widehat{C}$ have been added, the cycle is closed by joining labels a and b, which are then relabeled to 0 and hence, labels a and b are free to be used for the next column.

When constructing an arrow, we move every internal vertex in $A(u, v) \setminus \{u, v\}$ that has edges to all of its neighbours into label 0. Then, we join $p_{i,j}$ and $p_{i+1,j}$ to the extremities of paths P_i^*,P_{i+1}^* if they exist by joining label $\lceil i/2 \rceil$ (containing the extremities of these paths only) to y_1 and y_2. This yields the crossing edges described in Figure 7. We then rename label $\lceil i/2 \rceil$ to 0. Lastly, we join $p_{i,j}$ and $p_{i+1,j}$ accordingly to A and B.

The creation of vertices $p_{i,j+1}$ and $p_{i+1,j+1}$ follows the same principle regarding the gadgets. Only we join them to $p_{i,j}$ and $p_{i+1,j}$ using working labels z_1 and z_2 for $p_{i,j+1}$ and $p_{i+1,j+1}$, respectively, and joining y_1 to z_1 and y_2 to z_2. This is followed by renaming y_1 and y_2 to 0. The vertices labeled z_1 and z_2 can each be renamed to $\lceil i/2 \rceil$ as soon as joins to y_1 and y_2 respectively, and to neighbours in the possible arrow are done.

Repeating this process to the next row, column, until G_{n+1} is constructed yields our graph G with the desired number of labels. □

Bibliography

[BBBK20] Benjamin Bergougnoux, Édouard Bonnet, Nick Brettell, and O-joung Kwon. Close Relatives of Feedback Vertex Set Without Single-Exponential Algorithms Parameterized by Treewidth. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, volume 180 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[BCKN15] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015.

[BK19] Benjamin Bergougnoux and Mamadou Moustapha Kanté. Fast exact algorithms for some connectivity problems parameterized by clique-width. *Theor. Comput. Sci.*, 782:30–53, 2019.

[BKN+18] Marthe Bonamy, Lukasz Kowalik, Jesper Nederlof, Michal Pilipczuk, Arkadiusz Socala, and Marcin Wrochna. On directed feedback vertex set parameterized by treewidth. In Andreas Brandstädt, Ekkehard Köhler, and Klaus Meer, editors, *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, volume 11159 of *Lecture Notes in Computer Science*, pages 65–78. Springer, 2018.

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

[BST19] Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. IV. an optimal algorithm. *CoRR*, abs/1907.04442, 2019.

[BST20a] Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. I. general upper bounds. *SIAM J. Discret. Math.*, 34(3):1623–1648, 2020.

[BST20b] Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. II. single-exponential algorithms. *Theor. Comput. Sci.*, 814:135–152, 2020.

[BST20c] Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. III. lower bounds. *J. Comput. Syst. Sci.*, 109:56–77, 2020.

[BSTV13] Binh-Minh Bui-Xuan, Ondrej Suchý, Jan Arne Telle, and Martin Vatshelle. Feedback vertex set on graphs of low clique-width. *Eur. J. Comb.*, 34(3):666–679, 2013.

[CHL+12] Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce Reed, and Udi Rotics. Polynomial-time recognition of clique-width at most 3 graphs. *Discrete Applied Mathematics*, 160(6):834–865, 2012. Fourth Workshop on Graph Classes, Optimization, and Width Parameters Bergen, Norway, October 2009.

[CKN18] Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018.

[CMPP17] Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. Hitting forbidden subgraphs in graphs of bounded treewidth. *Inf. Comput.*, 256:62–82, 2017.

[CMR00] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[CNP+11] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011.

[CO00] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[HK21] Falko Hegerfeld and Stefan Kratsch. Towards exact structural thresholds for parameterized complexity. *ArXiv*, abs/2107.06111, 2021.

[Klo94] Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.

[Kor21] Tuukka Korhonen. Single-exponential time 2-approximation algorithm for treewidth. *CoRR*, abs/2104.07463, 2021.

[LMS11] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 777–789. SIAM, 2011.

[MRRS12] Pranabendu Misra, Venkatesh Raman, MS Ramanujan, and Saket Saurabh. Parameterized algorithms for even cycle transversal. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 172–183. Springer, 2012.

[Oum05] Sang-il Oum. Approximating rank-width and clique-width quickly. In Dieter Kratsch, editor, *Graph-Theoretic Concepts in Computer Science*, pages 49–58, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[Pil11] Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011.

[SdSS20] Ignasi Sau and Uéverton dos Santos Souza. Hitting forbidden induced subgraphs on bounded treewidth graphs. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 82:1–82:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.