Exact Algorithms for Loop Cutset

Sjoerd T. Timmer

Supervisor: Hans L. Bodlaender

Master Thesis: ICA-3118479

Universiteit Utrecht

Abstract:

The LOOP CUTSET problem was historically posed by Pearl as a subroutine in Pearl's algorithm for computing inference in probabilistic networks. The efficiency of the algorithm that solves the probabilistic inference highly depends on the size of the smallest known LOOP CUTSET. This justifies the search for exact algorithms for finding a minimum LOOP CUTSET. In this thesis we are investigating the algorithmic complexity of the problem. We will look at both the unparameterized problem and the problem parameterized by the treewidth of the input graph. For both we give an exact exponential time algorithm. The running times of these algorithms are $\mathcal{O}^\star(1.7548^n)$ and $\mathcal{O}^\star(4^{tw})$ respectively, where tw is the treewidth of the input graph. Finally, we prove a lower bound of 3^{tw} for the parameterized problem.

Keywords:

LOOP CUTSET, MAXIMUM INDUCED FOREST, Exact Algorithms, Treewidth, Cut & Count, Branch & Bound, Branch & Reduce, FEEDBACK VERTEX SET

Contents

Contents **3**

1 Introduction **4**
1.1 Background work 5
1.2 In this thesis 5

2 Preliminaries **7**
2.1 Definitions 7
2.2 Exact algorithms 8
2.3 Branch & Reduce 9
2.4 Treewidth 10
2.5 A tree property 11
2.6 Isolation Lemma 12

3 An Exact Algorithm **13**
3.1 Introduction 13
3.2 Transformation 13
3.3 Analysis 15

4 Using Tree Decompositions **17**
4.1 Introduction 17
4.2 Cut & Count 19
4.3 Lower bound 25

5 Discussion and Conclusions **28**
5.1 Exponential Complexity of the Loop Cutset problem 28
5.2 FPT algorithm for Loop Cutset 28
5.3 Future work 28

Bibliography **30**

Chapter 1

Introduction

The origin of the LOOP CUTSET problem resides with Pearl's algorithm [20] for calculating probabilistic inference in Bayesian Network models for probability distributions. Since the complexity of Pearl's algorithm depends exponentially on the size of the LOOP CUTSET that is used by the algorithm, it is highly desirable to find an optimal solution to the LOOP CUTSET problem. Even a constant addition to the size of the LOOP CUTSET increases the running time by a factor that is exponential in the size of that addition. A loop in a directed graph is a cycle in the underlying undirected graph. In particular the edges in a loop need not be directed along the directions of the edges in the graph. The LOOP CUTSET problem is concerned with finding a subset of vertices in a graph such that all loops are *cut*, with the restriction that a vertex only cuts a loop if it is not a *head-to-head* vertex with respect to that loop. A vertex is called a head-to-head vertex in a particular loop, if it is connected to both neighbors in that loop through incoming edges.

LOOP CUTSET
Input: A directed, acyclic graph $G = (V, E)$ and an integer k
Question: Is there a set $S \subseteq V$ of size at most k such that any loop in G contains a vertex $v \in S$ that is not a head-to-head vertex in that loop?

The LOOP CUTSET problem can also be seen as a directed variation on FEEDBACK VERTEX SET where only outgoing edges of the selected vertices are removed. It is intuitively true that removing outgoing edges leaves head-to-head connections intact and breaks any other kind of connection. This is the representation that we will mostly use throughout the rest of this thesis.

LOOP CUTSET
Input: A directed, acyclic graph $G = (V, E)$ and an integer k
Question: Is there a set $S \subseteq V$ of size at most k such that $G' = (V, \{(u, v) \in E | u \notin S\})$ is a forest?

1.1 Background work

The LOOP CUTSET problem has obvious similarities to FEEDBACK VERTEX SET and DIRECTED FEEDBACK VERTEX SET, problems that are both well studied [12, 14, 21]. The (UNDIRECTED) FEEDBACK VERTEX SET problem is one of the problems on Karp's original list of NP complete problems [15]. Exact algorithms and FPT algorithms for this problem have been developed over the years [7, 10, 12, 14, 21].

The first exact algorithm for FEEDBACK VERTEX SET is due to Razgon [21] in 2006. This was improved by Fomin et al. [12] who provide a $\mathcal{O}(1.7548^n)$ algorithm for FEEDBACK VERTEX SET and later by Fomin and Villanger [14] who reduce the upper bound to $\mathcal{O}(1.7348^n)$ using potential maximal cliques.

Another well studied approach on combinatorial graph problems are FPT algorithms [11], where the question is to find fast algorithms parameterized by the size of the solution. Well known results include the work of Fomin et al. [12] and Chen et al. [7]. The latter provides a $O(5^k kn^2)$ algorithm that solved FEEDBACK VERTEX SET to optimality. Recently the best known FPT bound for FEEDBACK VERTEX SET was improved by Cao, Chen and Liu [6] to $\mathcal{O}(3.83^k kn^2)$. Recently also DIRECTED FEEDBACK VERTEX SET (where it is the goal the break all directed cycles) has been solved in $4^k k!n^{\mathcal{O}(1)}$ time by Chen et al. [8].

The LOOP CUTSET problem itself is less well known. In the context of exact algorithms there are no previous results on the running time of exact algorithms for this problem. It is known to be in NP for a long time [24] but little effort has been made to provide fast exact algorithms. A lot of work was done on heuristic approaches. See for example the work of Stillman [23]. Later the focus of research has shifted towards approximation algorithms [2,3].

1.2 In this thesis

In this thesis we present exact algorithms for the LOOP CUTSET problem, parameterized by the treewidth as well as the size of the input. The notion of treewidth has been studied for a long time [22], also in combination with exact exponential time algorithms [4, 13]. Although the computation of treewidth and tree decompositions is a generally hard problem itself, it is still a useful measure for the *complexity* of a graph. Besides that, we do not necessarily need an optimal tree decomposition to find an optimal LOOP CUTSET.

More specifically, we have investigated the application of the Cut & Count technique to the LOOP CUTSET problem. As we will describe in Chapter 4, it turns out that the problems to which it applies are very well defined to be the class of connectivity problems. For many of these problems the implications of the Cut & Count approach have been studied [9]. This is however not true for LOOP CUTSET.

Investigating further into the exponential complexity of the LOOP CUTSET problem, I have also studied Exact Algorithms for LOOP CUTSET. Based on the work of Fomin [12] we will present an $\mathcal{O}^\star(1.7548^n)$ time algorithm for LOOP CUTSET. This algorithm is a basic application of Branch & Reduce. We also provide an algorithm that solves the problem in $\mathcal{O}^\star(4^{tw})$ time, based on the work of Cygan et al. [9]. Furthermore, when the problem is parameterized by the treewidth of the input we show that no algorithm solves the problem in $(3-\epsilon)^{tw} \cdot n^{\mathcal{O}(1)}$ time for any positive constant $\epsilon > 0$.

In Chapter 2 we will introduce the preliminary theory and establish notational conventions. We will provide an exact exponential time algorithm for LOOP CUTSET in Chapter 3.

In Chapter 4 follows the algorithm for the problem parametrized by the treewidth and the lower bound proof for that problem. Finally, in Chapter 5 we discuss the results and some concluding remarks.

Chapter 2

Preliminaries

2.1 Definitions

Throughout this thesis we will denote by $G = (V, E)$ the input graph. Mostly this will be a directed acyclic graph on vertices V and directed edges $E = \left\{(v_i, v_j) \middle| v_i, v_j \in V\right\}$ (directed from v_i to v_j). In a directed graph a cycle consists of a closed, circular sequence such that for every consecutive pair of vertices in this sequence there is an edges in the graph directed from the first to the latter.

Definition 2.1. *A directed graph is a* directed acyclic graph or DAG *iff it contains no directed cycles.*

In undirected graphs a cycle is just a circular sequence of vertices connected by edges.

A loop in a directed acyclic graph is a cycle in the underlying undirected graph. I.e, a loop in a directed graph becomes a cycle if we drop the directions of the edges. Note that this definition of loops includes, but is not restricted to cycles.

Many problem definitions will also contain a special subset $F \subseteq V$ of forbidden vertices as part of the input (or $R \subseteq V$ of required vertices) that must be (or may not be) present in the solution.

Besides the fact that this is sometimes necessary for the algorithm, it is also advantageous because it allows us to formulate a constructive algorithm that returns a solution of size k rather than just the (non-)existence of such a solution. This is done by iteratively adding or removing a vertex to the solution and using the decision algorithm to determine whether a feasible solution still exists. If so we recurse on the smaller problem, if not we can remove the vertex from the graph because we know that it does not occur in any minimum solution.

On various occasions, we will be reasoning about the *underlying undirected* graph. This is the graph with the same vertex set but where the direction of the edges is dropped.

By $N(v)$ we will denote the (open) neighborhood of a vertex v: $N(v) = \{v' \in V | (v', v) \in E \vee (v, v') \in E\}$. The closed neighborhood $N[v]$ contains v itself as well and is defined as $N(v) \cup \{v\}$. With the neighborhood of a set $X \subseteq V$ we will refer to the union of the neighbor-

hoods of that set; $N(X) = \bigcup_{x \in X} N(x)$ and $N[X] = \bigcup_{x \in X} N[x]$ respectively for open and closed neighborhoods.

For any subset of vertices X the graph $G[X]$ denotes the subgraph of G induced by X. That is, $G[X] = \left(X, \left\{E = (v_i, v_j) \middle| (v_i, v_j) \in E \wedge v_i, v_j \in X\right\}\right)$

2.2 Exact algorithms

The field of Exact Algorithms is concerned with the construction of algorithms that solve their problem in an optimal way. Many of these problems are known to be NP Hard and therefore it is assumed that these algorithms will always be exponential in their worst case running time. The goal of the research into exact algorithms is to minimize the base of the exponent, and therefore increase the size of the largest instance that can be solved within limited computation time. The problem at hand is the LOOP CUTSET problem that was already defined in the Introduction.

We will very often be dealing with FEEDBACK VERTEX SET instances as well, since on multiple occasions we will transform our LOOP CUTSET into a FEEDBACK VERTEX SET problem and solve that instead. The Minimum FEEDBACK VERTEX SET problem is equivalent to the MAXIMUM INDUCED FOREST problem. The definitions of these problems are:

FEEDBACK VERTEX SET
Input:
An undirected graph $G = (V, E)$ and an integer k
Question:
Is there a set $S \subseteq V$ of size at most k such that $V \setminus S$ is acyclic?

MAXIMUM INDUCED FOREST
Input:
An undirected graph $G = (V, E)$ and an integer l
Question:
Is there a set $X \subseteq V$ of size at most l such that $G[X]$ is a forest?

It should be clear that these problems are equivalent, since any FEEDBACK VERTEX SET of size k is the complement of a Maximum Induced Forest of size $n - k$ and vice versa. We will use both representations in the rest of this thesis.

Now for the LOOP CUTSET problem, we need a definition of the notion of a head-to-head vertex. We will say that:

Definition 2.2. *Given a loop of vertices $(c_1, c_2, \ldots, c_j)$ and a vertex c_i on that loop, this vertex is a* head-to-head vertex *if $(c_{i-1}, c_i) \in E$ and $(c_{i+1}, c_i) \in E$.*

Something similar to the transformation to a maximum tree problem can be done for LOOP CUTSET. In the LOOP CUTSET problem, however, we are not just interested in the forests induced by the remainder of some set S. We specifically want that the removal of all outgoing edges of the set S makes the graph a forest. We define the operation of cutting all out-edges from the graph as $G \rangle S \langle$:

Definition 2.3. *The graph $G \rangle S \langle$ is the graph $G' = (V, E')$ where $E' = \{(v, w) \in E | v \notin S\}$ are the edges from the original graph minus the outgoing edges from S.*

Using this notation we can then write down an even more straight forward definition of the LOOP CUTSET problem:

LOOP CUTSET
Input: A directed acyclic graph $G = (V, E)$ and an integer k
Question: Is there a set S of size at most k such that $G\rangle S\langle$ is a forest?

The operation $G\rangle S\langle$ is demonstrated in Figure 2.1. The inverse operation $G\langle X\rangle$ is the equivalent to LOOP CUTSET of taking the induced subgraph for FEEDBACK VERTEX SET. So Figure 2.1 is also an example of the operation $G\langle X\rangle$ on the set of white vertices $X = V \setminus S$.

Definition 2.4. $G\langle X\rangle = G\rangle V \setminus X\langle$

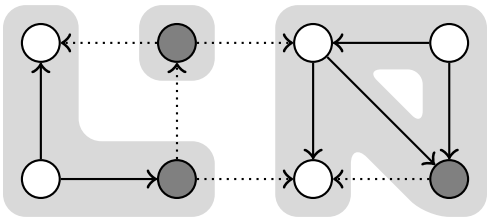


Figure 2.1: Example of the cutout operation $G\rangle S\langle$. The set S is filled in dark gray. Edges that got removed are shown with dotted lines. The number of connected components will become important later, so we have shown the connected components of $G\rangle S\langle$ in light gray. Note that the elements of S itself are also part of a connected component. In this particular case the set S is not a LOOP CUTSET since there is a loop in the right most component.

2.3 Branch & Reduce

The exact algorithm presented in Chapter 3 uses a branch & reduce technique. This technique was introduced by Land and Doig [17] in 1960. Branching algorithms in general explore the tree of possible solutions by splitting the problem in two subproblems in which the choice of one of the vertices is fixed. In one subproblem some vertex is required in the solution and in the other this vertex is forbidden to be in the solution. We refer to vertices that are in this way required to be in the solution as *selected* or *chosen* vertices.

For FEEDBACK VERTEX SET selecting a vertex in the solution or forbidding it in the remainder(the maximum induces forest) is equivalent to removing the vertex and all connected edges from the graph. For LOOP CUTSET however we need to keep track of deleted vertices and in particular the head-to-head connections in that vertex.

The main idea behind branch & reduce algorithms is that in the selection or non-selection of a vertex we not only make progress for that vertex but it also has implications for the possibilities of other vertices. This is measured by assigning a weight to every vertex, that not only depends on the state (selected or not) but also on some property of the neighbors. From the initial weight, the reduction of weight per branching and the number of branches we can calculate the running time in which α is still a variable. Finally is it possible to solve for which α the running time of the branching algorithm is minimal.

2.4 Treewidth

One way to express the complexity of a graph is by looking at the treewidth [22] of the graph instead of the number of vertices. In many practical applications this treewidth is relatively low compared to the number of vertices.

A tree decomposition $(\mathbb{T}, \mathbb{B})$ of a graph $G = (V, E)$ is a rooted tree $\mathbb{T} = (X, F)$ in which every node $x \in X$ has a bag $B_x \in \mathbb{B}$ associated to it that contains a subset of the vertices from G; $B_x \subseteq V$. This assignment of vertices to the bags must satisfy two additional properties:

- Every vertex and every edge occurs in the induced subgraph $G[B_x]$ of at least one $x \in \mathbb{X}$.
- For every vertex $v \in V$, the nodes of the bags that contain v induce a subtree in the tree decomposition.

The width of a tree decomposition is the cardinality of the largest bag in $\mathbb{T}$, minus 1. The treewidth of a graph is the minimum width over all possible tree decompositions of that graph. Any tree has treewidth 1, hence the name.

Furthermore, we may refer to V_x as the union of B_y over all descendants y of x (including x itself):

$$V_x = \bigcup_{y \text{ 'descendant of' } x} B_y$$

2.4.1 Pathwidth

In the proof of the lower bound for LOOP CUTSET we use the notion of pathwidth instead of treewidth. The pathwidth is the minimum width of a tree decomposition in which $\mathbb{T}$ is a path. Such a decomposition is called a path decomposition instead of a tree decomposition. In our proof for the lower bound we restrict ourselves to path decompositions because known lower bounds on FEEDBACK VERTEX SET are defined in terms of pathwidth.

2.4.2 Nice tree decomposition

Common practice when working with tree decompositions, is to transform the decomposition into a *nice tree decomposition*. In a nice tree decomposition only one of the following operations takes place between a vertex x and the children of x. Note that we use a different, larger set of (smaller) operations than the commonly used notion of nice tree decompositions. Besides the finer grained operations, we also require that leaf bags and the root bag are empty. Finally, besides the subset of vertices we associate a subgraph of G with every bag. This subgraph contains the vertices from B_x and some of the edges between those vertices, but not necessarily all. Edges will be introduced explicitly. The nice tree decompositions that we will use exists of the following types of bags:

- x is a **leaf**: x has no children and B_x has no vertices: $B_x = \emptyset$.
- x **introduces** a new **vertex** v: x has only one child y and it contains $B_x = B_y \cup \{v\}$. The associated subgraph also contains exactly the same edges so the new vertex is of degree 0 in this subgraph.
- x **introduces** a new **edge** (u, v): x has one child with exactly the same vertices $B_x = B_y$, but this bag is labeled with the edge (u, v). Every edge is introduced exactly once. The associated subgraph now contains this edge.

- x **joins** two other bags y and z: these are the only two children of x. They both have exactly the same vertex sets and the same edge set, $B_x = B_y = B_z$.
- x **forgets** a vertex $v \in B_y$: x has one child y that contains v, $B_x = B_y \setminus \{v\}$.

This kind of nice tree decomposition was introduced by Cygan et al. [9].

Every tree decomposition of width p can be transformed into a nice tree decomposition of equal width [16]. The same holds for path decompositions. This is done by expanding every parent child pair in the tree decomposition into a chain of operations until all operations are of one of the atomic kinds.

2.4.3 Dynamic programming on tree decompositions

The algorithm in the Chapter 4 works by performing a dynamic programming routine on the bags of the tree decomposition. It tabulates all possible assignments of some state $S \in \{s_1, s_2, \ldots\}$ to the vertices. When we count the number of possible state assignments to all vertices this would amount to $|S|^n$ records; one for every possible combinations of states. If, however, we can limit the number of vertices for which we tabulate the state assignments to the vertices in one bag, we can reduce this to a number of tables of size $|S|^{tw}$, which reduces the running time of the algorithm to $\mathcal{O}^\star(|S|^{tw})$.

2.5 A tree property

In the argumentation in Chapter 4 we will use the following lemma on the number of connected components of a tree.

Lemma 2.1. *(Folklore) Considering a graph $G = (V, E)$. Call the number of vertices $n = |V|$ and the number of edges $m = |E|$. Let c be the number of connected components of G. Then, G is a forest if and only if $c = n - m$.*

Proof. We prove the lemma by induction to $|E|$, first showing that the graph without edges satisfies the condition and then showing that adding an edge does not change the validity.

- Any graph $G = (V, \emptyset)$ with n vertices has $c = n$ connected components, since every vertex is a connected component by itself. There are no edges, therefore $m = |\emptyset| = 0$. The equality $c = n - m$ follows.
- Assuming we are given a forest $G = (V, E)$ with n vertices and m edges for which the equality $c = n - m$ holds. Adding one edge to the graph increases the number of edges to $m' = m + 1$ and the number of vertices remains $n' = n$. This operation can either connect two components into one, or connect one component to itself, which introduces a loop. In the first case the number of connected components decreases by one so the new number of connected components is $c' = c - 1$, so we can derive that $c' = c - 1 = (n - m) - 1 = (n' - (m' - 1)) - 1 = n' - m'$. If, on the other hand, we introduce a loop, the number of connected components remains the same and the equality is violated.

The above argumentation ensures that all forests have this property. For the implication to hold both ways we have to show that only forests have this property, i.e. no cyclic graph has $n - m$ connected components. We can see that this is true if we look at the procedure above.

Once the first loop has been introduced the graph has $c = n - m + 1$ connected components. In every step from then on the number of connected components can only increase by zero or one, while m increases by 1, therefore $c = n - m$ can never be reached again. □

2.6 Isolation Lemma

A very useful concept that we will need in the analysis in Chapter 4 is the Isolation Lemma [19]. A weight function $\omega : U \to \mathbb{Z}$ is used to assign integer weights to vertices (or more generally, a universe U). We denote by $\omega(v)$ the weight of vertex v and by $\omega(X)$ the total weight $\sum_{v \in X} \omega(v)$ of some subset of vertices $X \subseteq V$.

Now we consider a collection of subsets of U. This a called the set family $\mathscr{F} \subseteq 2^U$. In our case this will be the family of solutions.

A weight function $\omega(v \in U) = \{0, 1, \ldots\}$ is said to be isolating a set family $\mathscr{F}$ if there is only one unique set in $\mathscr{F}$ of minimum weight according to that function. Mathematically we can express this as:

Definition 2.5. *Given a weight function $\omega : U \to \mathbb{Z}$ and a set family $\mathscr{F} \subseteq 2^U$, ω* isolates *$\mathscr{F}$ if and only if there is a set $S \in \mathscr{F}$ with $\omega(S) < \omega(S')$ for every other set $S' \in \mathscr{F} \setminus \{S\}$.*

The Isolation Lemma now states that:

Lemma 2.2 (Isolation Lemma [19])**.** *Suppose we are given a universe U and a set family over that universe $\mathscr{F} \subseteq 2^U$ of size at least one. If we assign independent random weights $\omega(u) \in \{1, 2, \ldots, N\}$ to all elements $u \in U$ uniformly, then this weight function ω separates that set family with probability at least $1 - \frac{|U|}{N}$.*

More specifically, we can use the set of solutions to some graph problem as the set family. If we are given a graph with n vertices and we assign them weights in the range $\{1, 2, \ldots, 2n\}$ then the set of minimum cardinality solutions to the problem contains one unique solution of minimum weight with probability at least $\frac{1}{2}$.

Chapter 3

An Exact Algorithm

3.1 Introduction

Fomin et al. [12] have developed an exact algorithm to compute a minimum FEEDBACK VERTEX SET of a graph in $\mathcal{O}^\star(1.7548^n)$ time. This algorithm uses branch & reduce as well as branch & bound to achieve this fast running time. We will use this algorithm as a black box subroutine to obtain an equally fast algorithm for LOOP CUTSET. However, in order to prove that the worst case running time is indeed equally low, we will show that a problem arises in the analysis of the new running time. Therefore we cannot approach the known algorithm as black box when analysing the worst case running time. We will describe the original analysis and how it can be translated to the LOOP CUTSET problem.

Given an instance of LOOP CUTSET we can transform this into an instance of BLACKOUT FEEDBACK VERTEX SET (the name was given by van Dijk [5], but the notion was implicitly used by Fomin et al. as well). On this transformed instance we can run the branching algorithm for FEEDBACK VERTEX SETS, and transform the result back into a solution for the LOOP CUTSET problem. Unfortunately, we will see this transformation doubles the number of vertices in the graph and therefore the running time would increase to $\mathcal{O}^\star\left(1.7548^{2\cdot n}\right) = \mathcal{O}^\star(3.0793^n)$. However, when we consider the graphs that can result from this particular transformation we show that the the running time of this algorithm is still bounded by $\mathcal{O}^\star(1.7548^n)$.

3.2 Transformation to BLACKOUT FEEDBACK VERTEX SET

Let us first define the BLACKOUT FEEDBACK VERTEX SET problem. This is the variant of the FEEDBACK VERTEX SET problem where part of the input is excluded from the solution. I.e. some subset F of vertices is *forbidden*. We will also refer to these vertices as *blacked out*. The

formal definition of the problem is:

BLACKOUT FEEDBACK VERTEX SET
Input: An undirected graph $G=(V,E)$, a set of forbidden vertices $F \subseteq V$ and an integer k
Question: Is there a set $S \subseteq V$ of size at most k with $S \cap F = \emptyset$, such that $V \setminus S$ induces a forest in G?

Note that the blacked out vertices are forbidden in the FEEDBACK VERTEX SET. This means that they are a mandatory part of the solution to the equivalent MAXIMUM INDUCED FOREST problem.

Given a LOOP CUTSET instance we can transform it in the following fashion [2]. Every vertex v_i is split in two vertices v_i^{in} and v_i^{out} connected by an edge. All incoming edges in the original graph are directed to v_i^{in} while outgoing edges are transfered to v_i^{out}. In this transformed graph we then emphblackout all v^{in} vertices. The intuition behind this, is that it prevents the cutting of head-to-head connections. See Figure 3.1 for an example.

We will capture the described transformation as follows:

Definition 3.1. *The* transformation *T applied to a directed graph $G=(V,E)$ yield a new undirected graph $T(G)=G'=(V',E',F)$ for which:*

- $V' = \bigcup_{v_i \in V} \left\{ v_i^{in}, v_i^{out} \right\}$
- $E' = \left\{ (v_i^{in}, v_i^{out}) \middle| v_i \in V \right\} \cup \left\{ (v_i^{out}, v_j^{in}) \middle| (v_i, v_j) \in E \right\}$
- $F = \left\{ v_i^{in} | v_i \in V \right\}$

The transformation T also maps solutions: $T(S)$ is, for a solution $S \subseteq V$, the set

$$T(S) = \left\{ v_i^{out} \middle| v_i \in S \right\}$$

Suppose that we find a solution S' to the BLACKOUT FEEDBACK VERTEX SET on $T(G)$, we can find the transformed solution to the LOOP CUTSET Problem by the simple construction of $S = \left\{ v_i \middle| v_i^{out} \in S' \right\}$. We now claim that this set S is a solution to the LOOP CUTSET problem on G.

Lemma 3.1 (Bar-Yehuda and Geiger [2])**.** *S' is a solution to the* BLACKOUT FEEDBACK VERTEX SET *on $T(G)$ if and only if S is a solution to the* LOOP CUTSET *problem on G.*

Proof. Suppose there exists a LOOP CUTSET A for G that does not have a corresponding FEEDBACK VERTEX SET $T(A)$ in $T(G)$ that solves the transformed FEEDBACK VERTEX SET instance. Then there still exists a cycle in $T(G \setminus A)$. If we contract the vertices v_i^{in} and v_i^{out} together again, we obtain a loop in the original graph G that does not include any of the removed vertices and therefore such a solution A cannot exist.

Now for the other way around; suppose S is not a solution on G, then there exists a loop $C=(c_1,c_2,\ldots,c_1)$ in G such that every vertex $c_i \in A$ is a head-to-head vertex in that loop. This loop correspond to a cycle in the transformed graph $T(G)$. For head-to-head vertices, this transformed cycle only contains v_{in}. Thus, the transformed cycle contains only out-vertices that are not in the solution and in-vertices that can by definition not be in the solution so none of the vertices in this cycle is in S. This is in contradiction with the fact that the set S' was a solution to the FEEDBACK VERTEX SET problem.

□

Now that we know that the solutions are mapped one on one, we can be sure that a minimum solution in one problem is also minimum in the other, otherwise a smaller solution would exists, which could be mapped back, which then would yield a solution smaller than the current one that we supposed to be a minimum solution in the first place.

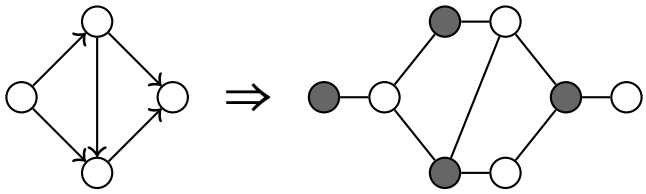


Figure 3.1: Transformation from LOOP CUTSET to BLACKOUT FEEDBACK VERTEX SET; Blackout vertices are indicated in gray.

3.3 Analysis

The above transformation duplicates the number of vertices in the graph. Therefore, if the LOOP CUTSET problem had n vertices, we now have a BLACKOUT FEEDBACK VERTEX SET with $2n$ vertices. That means that if we run the BLACKOUT FEEDBACK VERTEX SET algorithm by Fomin et al. on this graph it will spend time in the order of $\mathcal{O}^\star\left(1.7548^{2n}\right) = \mathcal{O}^\star(3.0792^n)$ to solve the problem. However, we know that the BLACKOUT FEEDBACK VERTEX SET instances that we obtain, are the result of this transformation and we can exploit this fact in the analysis.

The property that we use to improve the analysis, is the fact that half of the vertices in the obtained graph is already blacked out. There are only n vertices for which we need to make the decision and the other n vertices need to be in the MAXIMUM INDUCED FOREST. We will see that these vertices do not contribute to the complexity of the problem.

We will now review the way Fomin et al. obtained the upper bound on the running time for their FEEDBACK VERTEX SET algorithm. They solve the FEEDBACK VERTEX SET problem in $\mathcal{O}(1.7548^n)$ by solving BLACKOUT FEEDBACK VERTEX SET with a non-standard measure μ in $\mathcal{O}^\star(1.3328^\mu)$ time, where μ is the total *weight* of the graph. The total weight is defined as the sum of the weights of all vertices in the input. For vertices the following weights apply:

$$\mu(v) = \begin{cases} 0 & \text{if } v \in F \\ 1 & \text{if } v \in N(t), \text{ the neighborhood of some vertex } t \\ 1+\alpha & \text{otherwise} \end{cases}$$

The bound of $\mathcal{O}^\star(1.3328^\mu)$ is obtained by an extensive case analysis, in which every case results in a sufficiently high weight reduction. These reductions are dependent on α. The upper bound on the running time is then a function of α, minimizing this functions yields the bound of $\mathcal{O}^\star(1.3328^\mu)$ for $\alpha = 0.995$. From the fact that any vertex can have weight at most $1+\alpha$ it follows that the weight of any graph is at most $n(1+\alpha)$. It can then be concluded that the running time is bounded by $\mathcal{O}^\star\left(1.3328^{1.995n}\right) = \mathcal{O}^\star(1.7548^n)$.

Now, consider again how instances of the LOOP CUTSET problem are transformed to instances of BLACKOUT FEEDBACK VERTEX SET. In particular, we observe that when a directed acyclic graph $G = (V, E)$ with n vertices is transformed to an BLACKOUT FEEDBACK VERTEX SET instance G', then G' has n blacked out vertices, which have weight 0, and n vertices that

are not blacked out, which have weight $1+\alpha$. The total weight will therefore still be at most $n(1+\alpha)$ and the running time of the whole procedure is still bounded by $\mathcal{O}^\star(1.7548^n)$.

Thus, applying the algorithm of Fomin et al. [12] to G' costs $\mathcal{O}^\star(1.7548^n)$ time. As the transformation can be done in linear time, we obtain the following result.

Theorem 3.1. LOOP CUTSET *can be solved in* $\mathcal{O}^\star(1.7548^n)$ *time.*

Chapter 4

Using Tree Decompositions: Cut & Count

4.1 Introduction

In this chapter we will discuss a randomized algorithm that finds a LOOP CUTSET of size k in a graph for which we know a tree decomposition. This algorithm has a worst case running time of $\mathcal{O}^\star(4^{tw})$ for a tree decomposition of width tw and it is randomized in the sense that it may conclude false negatives with probability at most $\frac{1}{2}$. I.e. it may conclude that no LOOP CUTSET of size k exists, while such a solutions does exist. On the other hand, if a solution does not exists, the algorithm will always conclude so. The probability of false negatives may be decreased by running the algorithm multiple times or by choosing the random weights from a larger interval. The algorithm below is tailored to obtain a minimal chance of correctness of $\frac{1}{2}$, but any target probability strictly below one can be achieved.

Since we are dealing with problem instances where we are given a tree decomposition of the graph, we need to redefine the problem accordingly:

LOOP CUTSET
Input: A directed acyclic graph $G = (V, E)$, a tree decomposition of width tw, a set of required vertices $R \subseteq V$ and an integer k.
Question: Is there a set S with $R \subseteq S \subseteq V$ of size at most k such that $G \rangle S \langle$ is a forest?

In other words the graph where all outgoing edges of S are deleted must be a forest. That is equivalent to the requirement that any loop in G is cut by at least one vertex in S that is not a head-to-head vertex in that loop. Recall that with respect to a certain loop, a vertex v is a head-to-head vertex iff the arcs from both neighbors are directed towards v, i.e. if $(u, v), (v, w) \in E$ then v is a head-to-head vertex with respect to any loop $(\ldots, u, v, w, \ldots)$.

In the above definition a set R is required to be in the solution. Note that we cannot simply remove those vertices from the graph as we would have done for FEEDBACK VERTEX SET. For LOOP CUTSET, removed vertices can still contribute to loops in which they are a head-to-head vertex. The vertices of R are forbidden to be in the remaining forest. This can also be used to find a concrete solution and not only the existence of a solution of a particular size. This is done by iteratively adding vertices to the required part of the solution and checking whether a solution of the requested size still exists.

This problem is remarkably similar to the FEEDBACK VERTEX SET problem with the difference that parents of a head-to-head vertex are not disconnected by selecting that vertex. The described algorithm is based on the algorithm for FEEDBACK VERTEX SET by Cygan et al. [9].

In the second part of this chapter we will also show that under the Strong Exponential Time Hypothesis no algorithm solves LOOP CUTSET in $(3-\epsilon)^p \cdot \text{poly}(n)$ for any $\epsilon > 0$ where p is the pathwidth of the input graph.

4.1.1 A Failing Attempt

After the previous chapter in which we extensively leaned on a simple transformation from LOOP CUTSET to FEEDBACK VERTEX SET, we could be tempted to try and do the same for the problem parameterized by treewidth. However, it turns out that we cannot easily transform LOOP CUTSET into a FEEDBACK VERTEX SET while maintaining the treewidth. This would not be a problem if we could prove that the treewidth increases by a constant additive term for the whole graph, but this turns out to be difficult to prove. In all our attempts the treewidth increases by a multiplicative factor in the worst case. We demonstrate this by means of a simple example in which the treewidth increases from three to four in the operation of splitting a single vertex v. See Figure 4.1. This Figure shows that the treewidth can increase in a single splitting operation. Since the transformation of the graph splits all vertices, the treewidth could potentially increase by a factor two. That means we cannot use this transformation to obtain an equally fast algorithm for LOOP CUTSET. Whether a transformation exists that changes the treewidth by at most an additive term, remains an open question.

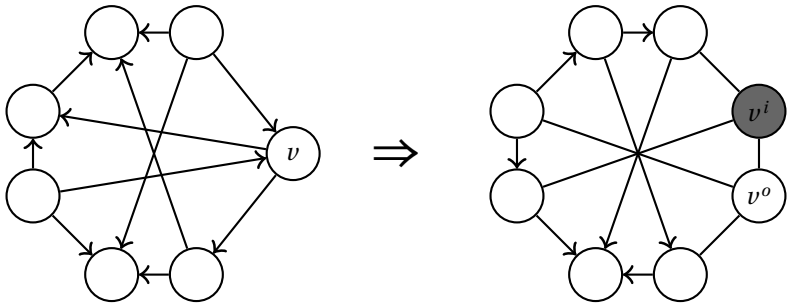


Figure 4.1: One step in the transformation from LOOP CUTSET (left) to FEEDBACK VERTEX SET (right); The vertex v is split in v^i and v^o. In this single operation the treewidth increases from 3 to 4. The vertex v^i gets blacked out, which is indicated in gray.

4.1.2 Catching global constraints in local properties

Tree decompositions are extremely useful for many problems with local constraints. Problems like INDEPENDENT SET and DOMINATING SET can be solved to optimality with a dynamic programming algorithm on the bags of the tree decomposition [1, 4]. For some problems the constant factor can be improved using convolutions [26], but this is not the case for the algorithm presented here.

For INDEPENDENT SET for instance, we can work from the bottom up, tabulating for every bag x in the tree decomposition and every target size k, the number of sets in B_x that allow an INDEPENDENT SET of size k in the graph that is induced by V_x, all vertices that have been introduced so far (and have possibly been forgotten again). A key idea in this method is that

we calculate the number of solutions in the subsets rather than that we calculate all solutions. Finally the answer of the algorithm is true if that number is more than 0 for the root bag and the requested value of k, and false otherwise. We will use something similar in the dynamic programming for LOOP CUTSET.

For each type of bag in a nice tree decomposition the table of values can be computed from the tables of the child bag(s). In the case of INDEPENDENT SET again, there are 2^b possible sets that we need to tabulate for a bag of size b and since the largest bag has size $tw+1$ we tabulate $\mathcal{O}(2^{tw})$ sets per bag. The number of bags in a nice tree decomposition is polynomial in the size of the graph as is the target size k and therefore we tabulate $\mathcal{O}^\star(2^{tw})$ entries in total.

We observe that in this case there is a global constraint that says that adjacent vertices cannot be in the solution simultaneously. Since every edge occurs in at least one bag, we can guarantee the correctness of the final answer by filtering the partial solutions that violate the constraint locally. In this way we check the constraint locally and still find all (and only) correct solutions.

For problems like FEEDBACK VERTEX SET and LOOP CUTSET a similar approach is more complicated because the globally required property for the solutions to these problems cannot simply be checked by verifying a local constraint. The required global properties for such problems are connectivity constraints, and therefore these problems are called connectivity problems. In this case the problem specification requires that the remainder of a solution induces a forest, which is a global structure that cannot easily be guaranteed by looking at the properties of individual vertices, because loops are structures that can span multiple vertices and edges. In a naïve application of dynamic programming we would need to know for every partial solution and every pair of vertices in a bag whether there is a path or not between those vertices in that partial solutions.

Now that we have seen how dynamic programming on tree decompositions works and why it does not work for connectivity problems, we will introduce the Cut & Count technique which will prove to solve the problem by converting the global constraint to a local constraint, albeit a more relaxed constraint that allows some *fake* solutions as well. The details will be described later, but the local constraint that we are going to introduce is that neighboring vertices obey some rules with respect to an assignment of states to the vertices. In the above description of the INDEPENDENT SET these states would be *in* and *out*, representing the choices for taking this vertex in the set and leaving it out respectively. For LOOP CUTSET we will consider four possible states.

4.2 Cut & Count

We have seen that the answer to the question whether a solution of size k exists, can be answered by counting the number of solutions in partial structures. The Cut & Count technique extends this approach by relaxing the structures that are being counted, but always such that the number of them increases by an even amount. The idea is that we count the combination of a solution, a marker assignment, and a consistent cut. We can prove that the number of counted objects differs an even amount from the number of real solutions. The combined objects that we do count but do not contain an actual solution will be referred to as *fake* or *false* candidates. We are thus no longer interested in the question if at least one solution exists, but whether the number of solutions is odd or even. We will therefore only tabulate the

parity of the number of counted objects. Furthermore we will see that if we choose weight uniformly and at random, there is a chance of getting a single minimum weight solution of at least $\frac{1}{2}$. That means that with probability at least $\frac{1}{2}$ we can detect that there is a solution, and with probability at most $\frac{1}{2}$ we get an even number of counted objects and we do not know whether a solution exists.

4.2.1 Global approach

Let us now formalize the approach described above. Suppose that $\mathscr{S} \subseteq 2^V$ is the family of solutions, i.e. 2^V is the power set of V that includes all possible subsets of vertices and $\mathscr{S}$ is the subset of those that actually fulfill the requirement that they cut all loops. For every $S \in \mathscr{S}$ we have that $G\rangle S\langle$ is a forest. Since we concluded in the previous subsection that we cannot check this by looking at the the partial solutions, we define the notion of *markers*. As we mentioned, the counted objects contain a marker assignment. In the dynamic program we keep track of the number of markers that was used. A key observation is that in the dynamic program we can exclude partial solutions that have a marker partitioned on the wrong side (to be defined later). This marker assignment can be verified locally and we will see that it can be used to check if the graphs are acyclic.

Now, instead of counting solutions in $\mathscr{S}$ we are going to count the elements of $\mathscr{C}$. This set $\mathscr{C}$ contains *consistently cut, marked, relaxed solutions*. We will now define this set, step by step.

Definition 4.1. *The set of* relaxed *or* candidate *solutions $\mathscr{R}$ contains all possible selections of vertices with all possible assignments of markers.*

Definition 4.2. *The set $\mathscr{C}$ of* consistently cut subgraphs, *contains elements of $\mathscr{R}$ together with a marker assignment and a consistent cut of V. All together this amounts to elements of the form $((S,M),(C_L,C_R))$ where $S \in \mathscr{R}$ is the set of vertices in this candidate solution, $M \subseteq V$ is a set of marked vertices and (C_L,C_R) is a* consistent *cut in V.*

The use and purpose of markers will be explained in more detail when we prove that the number of fake elements of $\mathscr{C}$ is even.

The sets C_L and C_R are the *left* and *right* part of an *empty cut* in V. I.e. Together they must span V and they cannot overlap. Note that we cut all vertices in V, not just the vertices in S.

Definition 4.3. *A cut (C_L,C_R) is called* consistent *iff:*

- *there are no edges (u,v) from $u \in C_L$ to $v \in C_R$ with $u \in S$ and $v \in V$ or vice versa, i.e. all vertices of any connected component of $G\rangle S\langle$ are partitioned on the same side.*
- *and all markers are in the left part of the cut; $M \subseteq C_L$.*

From now on we will consider the sets $\mathscr{C}_W = \{c \in \mathscr{C} | \omega(c) = W\}$ and $\mathscr{S}_W = \{s \in \mathscr{S} | \omega(s) = W\}$ to be the selection of solutions with a certain total weight. Solving the problem for every weight is not considerably harder than solving the problem for one weight W because there is only a polynomial number of total weights possible. Every vertex has an integer weight at most $4n$, so the total weight can at most be $4n^2$ and the total weight must be integer as well.

Now we can proceed with the main theorem of Cut & Count, applied to the LOOP CUTSET problem:

Lemma 4.1 (Cygan et.al. [9])**.** *Suppose that we can provide the following two ingredients for the* LOOP CUTSET *problem:*

1. *The number of* fake *candidate solutions is even:* $\mathscr{C}_W \equiv \mathscr{S}_W \pmod 2$
2. *A dynamic program to count the elements of* $\mathscr{C}_W$ *(modulo 2);*

then this dynamic program can be used to obtain a randomized algorithm that returns false positives with probability at most $\frac{1}{2}$.

Proof. We know that if the dynamic program returns an odd number of minimum weight solutions, there must have been an odd number of actual solutions of the desired size, which implies that there was at least one solution. If on the other hand we find an even number of consistent cuts we do not know if these all correspond to *fake* solutions or if there were any actual solutions. In the latter case we return false wrongly, since we may have missed some solutions (but always an even number of them) in this conclusion. However we can invoke on the Isolation Lemma to show that if the weights were chosen uniformly and at random, this cannot happen too often. In particular, a minimum weight solution is unique amongst the minimum cardinality solutions with probability at least $\frac{1}{2}$. Since one is odd by definition the number of minimum weight minimum cardinality solution is odd with probability at least $\frac{1}{2}$. □

Notice that we only need to know the parity of the number of consistent cuts. This means we need to store for every table entry in the dynamic program only whether it has an odd or an even number of possible consistent cuts. That means we can use a dynamic program that works modulo two. In the following sections we will provide the two required components of the Cut & Count technique separately in 4.2.2 and 4.2.3. In section 4.2.4 we state the formal definition of the dynamic programming algorithm.

4.2.2 Defining the Cuts

In order to maintain consistency with existing work on FEEDBACK VERTEX SET we will from now on consider X to be the complement of the set S, where S is the cutset and X the vertices that remain. The set X is the set that we are going to construct. For FEEDBACK VERTEX SET this set must induce a forest, but for LOOP CUTSET it must also induce a forest together with all incoming edges from S to X, i.e. $G\langle X\rangle$ must be a forest.

Since the Cut and Count technique requires us to know the maximum number of connected components in the tabulated solutions so far, we need a way to identify those. This is done by putting a *marker* on some of the vertices. In order to make sure that identical solutions with different marker-configurations have different weights we also assign an additional weight to all vertices that is only added to the total weight of a solution if that vertex is marked in that solution.

We will now give the formal definition of the weights. We first define the universe $U = V \times \{\mathbf{W}, \mathbf{M}\}$. Elements $(v, \mathbf{W})$ indicate the *normal* wight of vertex v, and elements $(v, \mathbf{M})$ are used to assign weights to marked vertices. We assign weights to vertices with a function $\omega : U \to \{1, \ldots, N\}$ where $N = 2|U| = 4|V|$. Every vertex has a weight that is added when it is in X and a weight that is added when it is marked.

We can then use the Isolation Lemma [19] to derive that this function ω isolates the family of minimum cardinality solutions with probability at least $\frac{1}{2}$, which means that there is exactly one minimum weight solution, which will be found by a dynamic program.

Now we define the following sets:

Definition 4.4. $\mathscr{R}_W^{A,B,C}$ *is the family* of solution candidates. *A candidate solution is a marked subset that excludes R that has A vertices, B edges in* $G\rangle S\langle$ *and C markers. We denote the elements of this set by tuples* (X,M) *where* $M,X \subseteq V$ *where X are the selected vertices and M the marked vertices.*

The elements of $\mathscr{R}_W^{A,B,C}$ are candidates for the forest, not for the LOOP CUTSET. Note also that the number of vertices refers to the number of vertices in some set X, but the number of edges refers to all outgoing edges from X, not only the edges between vertices in X.

Definition 4.5. $\mathscr{S}_W^{A,B,C}$ *is the family* of real, marked solutions. *These are the sets in* $\mathscr{R}_W^{A,B,C}$ *that satisfy the additional constraint that every connected component in* $G\rangle S\langle$ *is a tree that contains at least one marker.*

It is easy to verify that any actual solution leads to at least one element in one of the sets $\mathscr{S}_W^{A,B,C}$, but could lead to more solutions where the markers are places differently. We can now define the family of consistently cut subsets:

Definition 4.6. $\mathscr{C}_W^{A,B,C}$ *is the family of* consistently cut subsets $((X,M),(C_L,C_R))$ *where* $(X,M) \in \mathscr{R}_W^{A,B,C}$ *is a candidate solution and* (C_L,C_R) *is an empty cut in V that satisfies* $M \subseteq C_L$ *i.e. there are no edges from* C_L *to* C_R *or vice versa in* $G\rangle S\langle$ *and all markers are partitioned on the left.*

4.2.3 Counting Consistent Cuts

We are now going to count the number of consistent cuts of candidate solutions by means of dynamic programming. We will first show that the number of *fake* candidate solutions is even. This fact follows from the following two simple lemmas:

Lemma 4.2. *Every candidate solution with* $C \leq n - B$ *for which* $G\rangle S\langle$ *is not a forest has at least one unmarked component, i.e. a connected component that does not contain any marked vertices.*

Proof. Forests with n vertices and m edges have exactly $n-m$ connected components(see Lemma 2.1). Every candidate solution has n vertices since the operation $G\rangle S\langle$ does not remove vertices. Suppose some candidate solution $G\rangle S\langle$ has B edges but it is not a forest. Not to be a forest means that the graph $G\rangle S\langle$ must have more than $n-B$ connected components. Since there are only $C \leq n-B$ markers, and more than $n-B$ connected components there must be connected components without a marker. □

Lemma 4.3. *Every element in the solution set that contains unmarked components appears an even number of times in the set of consistent cuts.*

Proof. Since any unmarked component can be partitioned either left or right, there are 2^{cc} possibilities to divide the components over X_1 and X_2 where cc is the number of unmarked connected components. The lemma follows from the simple fact that 2^x is even for every $x \geq 1$. The number of cuts 2^{cc} can only be odd for $\text{cc} = 0$. □

Lemma 4.4. *the number of* fake *candidate solutions with $C \leq n - B$ is even:* $\mathscr{S}_W^{A,B,C} \overset{mod\,2}{=} \mathscr{C}_W^{A,B,C}$

Proof. This follows immediately from the combination of Lemma 4.2 and Lemma 4.3. □

Candidate solutions for which $C > n-B$ are not relevant, since they either are not a forest and therefore not a real solution, or they are a forest, but that means that the induces graph of that solution could also have been marked with $n-B$ markers, thus there must be another solution that is just as good that is in the set that we do consider. Since we are answering the question whether at least one solution exists we can thus ignore candidate solutions for which $C > n-B$. Note that we do not look at the size of the solutions A here(in contrast with the Cut & Count technique for FEEDBACK VERTEX SET), but we do need it to keep track of the size of partial solutions in order to check the final size of solution.

4.2.4 Dynamic Programming

We will now formulate a dynamic program to count the number of consistently cut, marked candidate solutions. Note that we pose the dynamic program as is, without a formal correctness proof.

4.2.4.1 The Tabulated elements

We are going to maintain a table of values $A_x(a,b,c,w,\mathbf{s})$ that represent, for some bag x in $\mathbb{T}$, the number of consistently cut candidate solutions of size a with b edges and c markers and a total weight of w that satisfies the assignment of states $\mathbf{s}$ to the vertices in the bag B_x. The state assignment $\mathbf{s}$ maps every vertex to one of the states $\{\mathbf{F}_L, \mathbf{F}_R, \mathbf{S}_L, \mathbf{S}_R\}$ where the states $\mathbf{F}_i$ represent the fact that this vertex should be part of the remainder forest and $\mathbf{S}_i$ indicates that this vertex is part of the LOOP CUTSET. The subscripts $_L$ and $_R$ refer to the *Left* and *Right* partitions of the cut.

We define the relevant sets as:

$$R_x(a,b,c,w) = \left\{ \begin{array}{lll} (X,M) & | & X \subseteq V_x \\ & \wedge & M \subseteq V_x \setminus B_x \\ & \wedge & X \cap R = \emptyset \\ & \wedge & |X| = a \\ & \wedge & |\{(v,w) \in E_x | v \notin X\}| = b \\ & \wedge & |M| = c \\ & \wedge & \omega(X \times \{\mathbf{W}\}) + \omega(M \times \{\mathbf{M}\}) = w \end{array} \right\}$$

$$C_x(a,b,c,w) = \left\{ \begin{array}{lll} ((X,M),(C_L,C_R)) & | & (X,M) \in R_x(a,b,c,w) \\ & \wedge & M \subseteq C_L \\ & \wedge & (C_L,C_R) \text{ is a consistently cut in } G[V_x] \end{array} \right\}$$

$$A_x(a,b,c,w,\mathbf{s}) = \left| \left\{ \begin{array}{lll} ((X,M),(C_L,C_R)) & \in & C_x(q,b,c,w) \\ & | & s(v) = \mathbf{F}_j \Rightarrow v \in X_j \\ & \wedge & s(v) = \mathbf{S}_j \Rightarrow v \in X_j \end{array} \right\} \right|$$

There are thus four states per vertex. So when we do dynamic programming on the bags of the tree decomposition, we know that there are 4^y combinations of states possible for a bag

of size y, and therefore $\mathcal{O}^\star(4^y)$ records in the dynamic program for every bag. All possible values of a, b,c and w are polynomial in the size of the instance. The number of bags is polynomial as well.

4.2.4.2 Update rules

We can now define the actual dynamic programming rules. Note that we will just state the update rules here without further proof of correctness. As described, we slightly deviate from the standard nice tree decomposition. We assume that the tree decomposition has empty leaves and an empty root. There is also a separate introduce edge bag. Every edge is introduced exactly once, namely in a special bag that is inserted just above the bag where both endpoint appear for the first time together, as seen from the root.

Leaf bag: Leave bags can only have solutions of size and weight zero, and in that case there is exactly one solution:

$$A_x(0,0,0,0,\emptyset) = 1$$

Introduce vertex bag: Initially a vertex is not connected to any other vertex because the edges are introduces separately. There are two options for these new vertices; they are either in or out of the solution.

$$\begin{aligned} A_x(a,b,c,w,\mathbf{s}\cup\{v\to\mathbf{S}_j\}) &= A_y(a,b,c,w,\mathbf{s}) && \forall j\in\{L,R\} \\ A_x(a,b,c,w,\mathbf{s}\cup\{v\to\mathbf{F}_j\}) &= [v\notin R]A_y(a-1,b,c,w-\omega((v,\mathbf{W}),s),\mathbf{s}) && \forall j\in\{L,R\} \end{aligned}$$

Introduce edge bag: Suppose we want to introduce the edge from u to v. By $()^{-1}$ we denote the function that maps L to R and vice versa.

$$A_x(a,b,c,w,\mathbf{s}) = \begin{cases} 0 & \text{if } \mathbf{s}(u)=\mathbf{F}_L \wedge \mathbf{s}(v)=\mathbf{v}_R \\ 0 & \text{if } \mathbf{s}(u)=\mathbf{F}_R \wedge \mathbf{s}(v)=\mathbf{v}_L \\ A_y(a,b-1,c,w,\mathbf{s}) & \text{otherwise if } \mathbf{s}(u)=\mathbf{F}_j \\ A_y(a,b,c,w,\mathbf{s}) & \text{otherwise } (\mathbf{s}(u)=\mathbf{S}_j) \end{cases} \qquad \forall j\in\{L,R\}\forall \mathbf{v}\in\{\mathbf{T},\mathbf{S}\}$$

Forget bag: When we forget a vertex, we have the option to mark it:

$$A_x(a,b,c,w,\mathbf{s}) = A_y(a,b,c-1,w-\omega((v,\mathbf{M})),\mathbf{s}[s(v)\to\mathbf{F}_L\cup\mathbf{S}_L]) + A_y(a,b,c,w,\mathbf{s})$$

Join bag: We can only join two bags when they have exactly the same state. The weight and cardinality of overlap is subtracted from the new values:

$$A_x(a,b,c,w,s) = \sum_{\substack{a_1+a_2-|s^{-1}(\mathbf{F}_j)|=a \\ b_1+b_2=b \\ c_1+c_2=c \\ w_1+w_2-\omega(s^{-1}(\mathbf{F}_j)\times F)=w}} A_y(a_1,b_1,c_1,w_1,s)A_z(a_2,b_2,c_2,w_2,s) \quad \forall i,j\in\{R,L\}$$

Theorem 4.1. *There exists a randomized algorithm for* LOOP CUTSET *that solves the problem in* $\mathcal{O}^\star(4^{tw})$ *for graphs of treewidth tw that reports false negatives with probability at most* $\frac{1}{2}$.

Proof. We refer to Lemma 4.1 to ensure that both ingredients are now present. We just formulated the dynamic program to count the number of candidate solutions modulo 2, and Lemma 4.4 ensures that the number of candidate solutions and the number of actual solutions differ an even number. □

4.3 Lower bound on LOOP CUTSET in terms of Treewidth

Due to Cygan et al. [9] it is known that under the Strong Exponential Time Hypothesis no algorithm solves FEEDBACK VERTEX SET in $(3-\epsilon)^p \cdot n^{\mathcal{O}(1)}$ time for a graph with a known path decomposition of width p and any constant $\epsilon > 0$. We will show that the same lower bound holds for LOOP CUTSET by giving a reduction from FEEDBACK VERTEX SET to LOOP CUTSET. First we presume the existence of a *black box algorithm* that does solve LOOP CUTSET instances in $(3-\epsilon)^p \cdot n^{\mathcal{O}(1)}$ time, and then we show that any such algorithm can be used to solve FEEDBACK VERTEX SET instances. Since Cygan et al. proved the latter to be impossible assuming the Strong Exponential Time Hypothesis, this proves that no algorithm can solve LOOP CUTSET in $(3-\epsilon)^p \cdot n^{\mathcal{O}(1)}$ time for any $\epsilon > 0$ assuming the Strong Exponential Time Hypothesis.

We will first discuss a transformation from FEEDBACK VERTEX SET to LOOP CUTSET and argue that it preserves solutions. In the second part we show that this transformation increases the pathwidth by at most one, and therefore maintains the exponential lower bound.

4.3.1 Transformation

The transformation from FEEDBACK VERTEX SET to LOOP CUTSET is as follows: Given an FEEDBACK VERTEX SET instance (G,k), we create an instance $(T(G),k)$ for the LOOP CUTSET problem by transforming the graph by some transformation T.

This transformation replaces every edge (v_i,v_j) by a vertex x_{ij} and two directed edges (v_i,x_{ij}) and (v_j,x_{ij}). See Figure 4.2a for an example. In Section 4.3.2 we will show that this transformation increases the pathwidth by at most one. Let us first prove the correctness of this transformation. I.e. the sets of solutions to the problem and the transformed problem are exactly equal. We start with the following observation:

Lemma 4.5. *Let $T(G)$ be the (directed) transformation of the (undirected) input graph G. Then, in $T(G)$ the following two statements hold:*

1. *when selected, all vertices v_i break any loop they are on.*
2. *all vertices x_{ij} do not break any loop when selected.*

Proof. For part 1, the vertices v_i have only outgoing edges in the transformed graph, and selecting a vertex is equivalent to removing outgoing edges. For part 2, these vertices have only incoming edges, and therefore they cannot break loops. □

Now the following theorem states the correctness of the transformation:

Theorem 4.2. *A set S is a solution to the* FEEDBACK VERTEX SET *problem on G if and only if it is a solution to the* LOOP CUTSET *problem on $T(G)$.*

Proof. The proof consists of two parts. For the first part, consider a solution set S and the LOOP CUTSET problem on $T(G)$. According to part 2 of Lemma4.5 the vertices x_{ij} will not break any loop. Therefore they do not contribute anything to the solution and will thus never

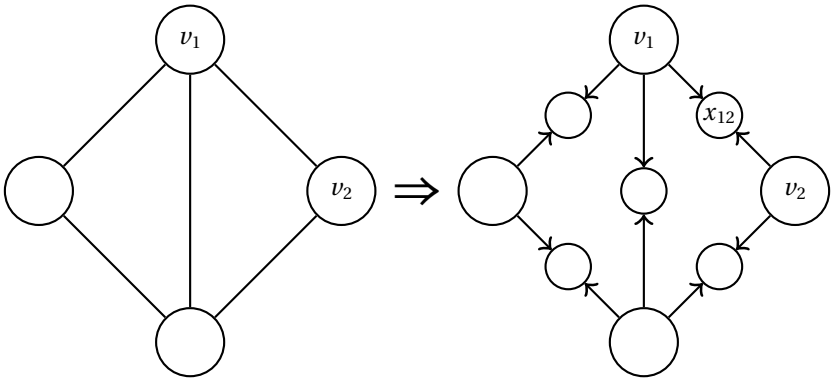
be part of a minimum solution S. Any loop in G can be mapped to a loop in $T(G)$. The set S can only be a solution to the LOOP CUTSET problem on $T(G)$ if it cuts this loop, but since in can only do so by selecting one of the vertices v_i on the loop, this also cuts the loop in G. Therefore, any solution to the LOOP CUTSET problem is also a solution to the FEEDBACK VERTEX SET problem.

For the second part we need to show that any solution to the FEEDBACK VERTEX SET problem can indeed be found by finding a minimum LOOP CUTSET in $T(G)$. We call on part 1 of Lemma 4.5 to assure that any vertex v_i that cuts a loop in G also cuts the corresponding loop in $T(G)$ because it can –by definition– not be a head-to-head vertex, since it has only outgoing edges. □

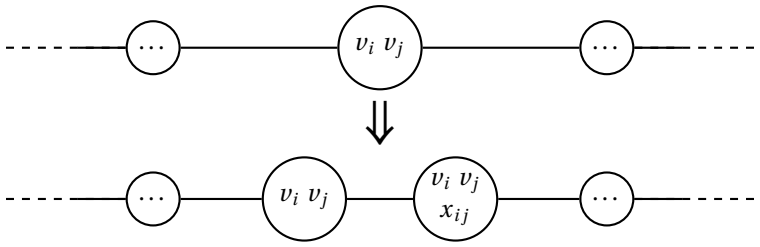
4.3.2 Transformation of path decompositions

We will now show that this transformation increases the pathwidth by at most one. For every new vertex x_{ij} in the transformed graph we duplicate one of the bags that was already present in the original path decomposition and that contains both endpoints of the edge (v_i, v_j) in the original graph. The definition of path decompositions states that at least one such bag exists. To this bag we then add x_{ij}. The duplicated bag is inserted immediately next to (either before or after) the original bag. So if a bag B_x was originally connected to B_y and B_z, the local order of bags will become $\ldots, B_y, B_x, B'_x, B_z, \ldots$.

Lemma 4.6. *Let $T(G)$ be the transformation of G. Then, the pathwidth p' of $T(G)$ is at most one more than the pathwidth p of G.*



(a) Transformation of the graph



(b) Transformation of a path decomposition

Figure 4.2: Transformation from FEEDBACK VERTEX SET to LOOP CUTSET

Proof. We prove this by showing that the described construction of a path decomposition for $T(G)$ is indeed correct and has width $p+1$. The latter part is trivial if we observe that every new bag has the same vertices as one of the original vertices plus one. For the correctness of the transformation we argue that this transformation maintains the properties of a path decomposition and introduces the bags required by the newly introduced vertices in the graph. A path decomposition must satisfy the following two criteria:

- Both endpoints of every edge occur together in at least one bag. The new vertices x_{ij} have two edges each. Since a bag is duplicated that already contained v_i and v_j, we know that there is now a bag that contains v_i and x_{ij}, thus fulfilling this requirement for a path decomposition for the new edge (v_i, x_{ij}). The same holds for the new edge (v_j, x_{ij}).
- The set of bags that contain some vertex v induce a sub path. The new vertex x_{ij} occurs only in bag and therefore this criteria is met by definition for those vertices. The original vertices v_i formed a sub path in the original path decomposition. But neither the operation of duplicating a bag, not the operation of adding vertices to a bag can break the induced sub path. Therefore, after the transformation this must still hold.

Thus the new path decomposition is correct for the transformed graph. □

Now we can easily conclude with the following theorem on the lower bound of the LOOP CUTSET problem.

Theorem 4.3. *Assuming the Strong Exponential Time Hypothesis, there cannot exist an algorithm that, given a graph G of pathwidth p, solves* LOOP CUTSET *on G in $(3-\epsilon)^p \cdot poly(n)$ for any constant $\epsilon > 0$.*

Proof. Suppose such an algorithm exists, we could take any instance for the FEEDBACK VERTEX SET problem and transform it as described above. The pathwidth of the transformed graph would be at most $p+1$, the algorithm will therefore take $(3-\epsilon)^{p+1} \cdot \text{poly}(n) = (3-\epsilon) \cdot (3-\epsilon)^p \cdot \text{poly}(n) = (3-\epsilon)^p \cdot \text{poly'}(n)$, which means we would have solved FEEDBACK VERTEX SET in that same running time, which is in contradiction with Strong Exponential Time Hypothesis. □

Chapter 5

Discussion and Conclusions

5.1 Exponential Complexity of the LOOP CUTSET problem

We have studies the exponential time complexity of the LOOP CUTSET problem. This problem has a very specific application in the inference of probabilities in probabilistic networks. It occurs as a black box subroutine in Pearl's algorithm for probabilistic inference.

Based on the exponential algorithms for FEEDBACK VERTEX SET by Fomin et al. we have given an algorithm that solves the LOOP CUTSET problem in $\mathcal{O}^\star(1.7548^n)$ time, where n is the number of vertices in the input graph.

We have also found an application of the Cut and Count technique for LOOP CUTSET resulting in an $\mathcal{O}^\star(4^{tw})$ time algorithm where tw is the treewidth of the input.

5.2 FPT algorithm for LOOP CUTSET

We also looked into FPT algorithms for LOOP CUTSET, but it turns out that the algorithms by Fomin [12], Chen [7] and Cao et al. [6] all apply to the LOOP CUTSET problem as well if we apply the simple transformation that we used in Chapter 3. This transformation made a BLACKOUT FEEDBACK VERTEX SET instance given a LOOP CUTSET instance. It follows straightforward from the definition of that transformation that the size of the solution remains equal. When we look at the implementation of the known FPT algorithm we see that all these algorithms actually work for BLACKOUT FEEDBACK VERTEX SET as well as FEEDBACK VERTEX SET. That means we can use them to find a LOOP CUTSET of a particular size just by transforming it into a BLACKOUT FEEDBACK VERTEX SET.

5.3 Future work

For the problem parameterized by treewidth we found a lower bound proof of $\mathcal{O}^\star(3^{tw})$. The actual algorithm that we provided solves the problem in $\mathcal{O}^\star(4^{tw})$ time. That means that there is a gap between the lower bound and the running time of the analysis. Either the lower bound or the algorithm could be improved. However, we suspect that this is very difficult.

The algorithm that we provided for Loop Cutset cannot easily be improved. In order to obtain a faster algorithm a whole new approach would have to be found. The same holds for the lower bound proof. We deducted the lower bound from the lower bound of the Feedback Vertex Set problem keeping the treewidth the same. If we want to increase the lower bound in this way, we would need to decrease the treewidth in that process which is very unlikely to be possible.

For the Feedback Vertex Set problem specialised algorithms for 3-regular graphs are known [18,25] that take only linear time in the size of the input. We speculate that it could be possible to adept such algorithms to the Blackout Feedback Vertex Set problem or at least to the Blackout Feedback Vertex Set instances that can arise from the transformation from Loop Cutset. In that way it would be possible to solve Loop Cutset in linear time for graph of maximum degree three.

Bibliography

[1] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989.

[2] R. Bar-Yehuda and D. Geiger. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and bayesian inference. *SIAM Journal on Computing*, 27(4):942–959, 1998.

[3] A. Becker and D. Geiger. Optimization of pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83(1):167–188, 1996.

[4] H. L. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.

[5] H. L. Bodlaender and T. C. van Dijk. A cubic kernel for feedback vertex set and loop cutset. *Theory of Computing Systems*, 46(3):566–597, 2010.

[6] Y. Cao, J. Chen, and Y. Liu. On feedback vertex set new measure and new structures. In H. Kaplan, editor, *Proceedings of SWAT 2010*, volume 6139 of *Lecture Notes in Computer Science*, pages 93–104, 2010.

[7] J. Chen, F. V. Fomin, Y. Liu, S. Lu, and Y. Villanger. Improved algorithms for feedback vertex set problems. *Journal of Computer and System Sciences*, 74(7):1188–1198, 2008.

[8] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55(5), 2008. Article No. 21.

[9] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In R. Ostrovsky, editor, *Proceedings of FOCS 2011*, pages 150–159, 2011.

[10] F. Dehne, M. Fellows, M. Langston, F. Rosamond, and K. Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. *Theory of Computing Systems*, 41(3):479–492, 2007.

[11] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[12] F. Fomin, S. Gaspers, A. Pyatkin, and I. Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. *Algorithmica*, 52(2):293–307, 2008.

[13] F. V. Fomin and D. Kratsch. *Exact Exponetial Algorithms*. Springer, 2010.

[14] F. V. Fomin and Y. Villanger. Finding induced subgraphs via minimal triangulations. In J.-Y. Marion and T. Schwentick, editors, *Proceedings of STACS 2010*, volume 5, pages 383–394, 2010.

[15] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[16] T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[17] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[18] D. Li and Y. Liu. A polynomial algorithm for finding the minimum feedback vertex set of a 3-regular simple graph. *Acta Mathematica Scientia*, 19(4):375–381, 1999.

[19] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[20] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[21] I. Razgon. Exact computation of maximum induced forest. In L. Arge and R. Freivalds, editors, *Proceedings of SWAT 2006*, volume 4059 of *Lecture Notes in Computer Science*, pages 160–171, 2006.

[22] N. Robertson and P. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49 – 64, 1984.

[23] J. Stillman. On heuristics for finding loop cutsets in multiply connected belief networks. In P. P. Bonissone, M. Henrion, L. N. Kanal, and J. F. Lemmer, editors, *Proceedings of UAI 1990*, pages 265–272, 1990.

[24] H. J. Suermondt and G. F. Cooper. Probabilistic inference in multiply connected belief networks using loop cutsets. *International Journal of Approximate Reasoning*, 4(4):283–306, 1990.

[25] S. Ueno, Y. Kajitani, and S. Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1–3):355–360, 1988.

[26] J. M. M. van Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In A. Fiat and P. Sanders, editors, *Proceedings of ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577, 2009.