University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Adam Starak
Student no. 361021

Application of parameterized techniques to finding spanning star forests in graphs

Master's thesis
in COMPUTER SCIENCE

Supervisor:
dr Michał Pilipczuk
Institute of Informatics

November, 2019

Supervisor's statement

Hereby I confirm that the presented thesis was prepared under my supervision and that it fulfils the requirements for the degree of Master of Computer Science.

Date | Supervisor's signature

Author's statement

Hereby I declare that the presented thesis was prepared by me and none of its contents was obtained by means that are against the law.

The thesis has never before been a subject of any procedure of obtaining an academic degree.

Moreover, I declare that the present version of the thesis is identical to the attached electronic version.

Date | Author's signature

Abstract

W pracy przedstawiono prototypową implementację blabalizatora różnicowego bazującą na teorii fetorów σ-ρ profesora Fifaka. Wykorzystanie teorii Fifaka daje wreszcie możliwość efektywnego wykonania blabalizy numerycznej. Fakt ten stanowi przełom technologiczny, którego konsekwencje trudno z góry przewidzieć.

Keywords

parameterized algorithm, kernelization, SETH, tree decomposition, cross-composition

Thesis domain (Socrates-Erasmus subject area codes)

11.3 Informatyka

Subject classification

D. Software
D.127. Blabalgorithms
D.127.6. Numerical blabalysis

Tytuł pracy w języku polskim

Zastosowanie technik algorytmów parametryzowanych dla problemu znajdowania gwiazd rozpinających grafy

Contents

Chapter 1

Introduction

A spanning star forest of a graph is a subgraph such that it contains all the vertices and its every connected component is a tree of depth 2. In the Spanning Star Forest problem, given a graph, we ask whether there exists a spanning star forest.

The goal of this paper is to apply numerous parameterized techniques to three different variants of the problem. We start with the most basic variant, denoted Spanning Star Forest where we only ask whether a graph has a spanning star forest. We show a simple condition, verifiable in linear time, that is necessary and sufficient for the existence of a spanning star forest in a graph. Later, we present an algorithm that constructs a spanning star forest. The following theorem summarizes these results:

Theorem 1.1. Spanning Star Forest *can be solved in linear time. Moreover, given a graph G, one can find a solution in linear time if it exists.*

Afterwards, we introduce the second variant. In Minimal Spanning Star Forest, we look for a spanning star forest with the minimum possible number of stars. We show that Minimal Spanning Star Forest and Dominating Set are essentially equivalent. That is, they are interreducible with respect to parameterized and polynomial-time reductions. Thus, we obtain with the following outcomes:

Theorem 1.2. Minimal Spanning Star Forest *is* W[2]*-complete.*

Theorem 1.3. Minimal Spanning Star Forest *is* NP-complete.

Based on reductions, we show a brute force algorithm. We point out that this running time is tight, using a lower bound proved for Dominating Set due to Pătraşcu and Williams [4].

Theorem 1.4. Minimal Spanning Star Forest *can be solved in $\mathcal{O}^*(N^{k+o_k(1)})$, where N is the number of vertices.*

Theorem 1.5. *Unless* CNF-SAT *cannot be solved in time $\mathcal{O}^*((2-\epsilon')^n)$ for some $\epsilon' > 0$, there does not exist constants $\epsilon > 0, k \geq 7$ and an algorithm solving* Minimal Spanning Star Forest *on instance with parameter equal to k in time $\mathcal{O}(N^{k-\epsilon})$, where N is the number of vertices of the graph.*

Finally, we introduce the last variant. In Spanning Star Forest Extension we are given a graph and a subset of forced edges. We ask whether there exists a spanning star forest in the graph that contains all the forced edges. In further, vertices that do not have any forced edge are called free vertices. We prove that this problem is essentially equivalent to CNF-SAT, where the number of forced edges corresponds to the number of variables while the number of free vertices corresponds to the number of clauses. Thus, we obtain:

Theorem 1.6. Spanning Star Forest Extension *is* NP-*complete.*

For the parametrization by the number of forced edges, our reductions yield the following:

Theorem 1.7. Spanning Star Forest Extension *parameterized by the number of forced edges can be solved in time* $\mathcal{O}^*(2^{|F|})$ *where* $|F|$*is the number of forced edges.*

Theorem 1.8. *There exists an algorithm solving* CNF-SAT *in time* $2^{o(n)}$*, where* n *is the number of variables, if and only if there exists an algorithm solving* Spanning Star Forest Extension *parameterized by the number of forced edges in time* $2^{o(n)}$*, where* n *is the number of forced edges.*

Also, we argue that there does not exist a polynomial kernel for Spanning Star Forest Extension when parameterized by the number of forced edges unless NP $\subseteq$ coNP/poly. We show two different approaches. The first one uses CNF-SAT. In the second approach, we use the composition framework proposed by Bodlaender et al. [10]. Namely, we prove that there exists a cross-composition of Spanning Star Forest Extension into itself. All in all, we obtain the following result:

Theorem 1.9. Spanning Star Forest Extension *parameterized by the number of forced edges does not admit a polynomial kernel unless NP* $\subseteq$ *coNP/poly.*

Recall that our reductions provides a link between free vertices and clauses. Therefore, when we parameterize the problem by the number of free vertices, we can immediately transform the algorithm proposed by ?? and obtain the following:

Theorem 1.10. Spanning Star Forest Extension *parameterized by the number of free vertices can be solved in ??.*

Furthermore, unlike in the case of the previous parametrization, we provide an algorithm that outputs a linear kernel.

Theorem 1.11. Spanning Star Forest Extension *parameterized by the number of free vertices admits a kernel with at most* k *clauses and* $2k$ *variables.*

Finally, we study the parameterization of the extension variant by the treewidth of the input graph. We propose a dynamic programming algorithm over a decomposition of a graph which uses fast cover product proposed by Björklund, et al. [12]. Our algorithm works in time $2^t \cdot poly(t) \cdot n$. In addition, we show that improving the running time would give a faster algorithm for a CNF-SAT.

Theorem 1.12. Spanning Star Forest Extension *parameterized by treewidth can be solved in time* $2^t \cdot poly(t) \cdot n$*.*

Theorem 1.13. *Unless* CNF-SAT*cannot be solved in time* $\mathcal{O}^*((2-\epsilon')^n)$ *for some* $\epsilon' > 0$*, there is no algorithm for* Spanning Star Forest Extension *parameterized by treewidth that achieves running time* $\mathcal{O}^*((2-\epsilon)^t)$ *for any* $\epsilon > 0$*, where* t *is the treewidth of the input graph.*

Outline. In Section 2 we settle notation. Section 3 contains results for Spanning Star Forest. In Section 4 we prove all the theorems dedicated for Minimal Spanning Star Forest. Section 5 give the algorithms and lower bounds for Spanning Star Forest Extension parameterized by three different values.

Chapter 2

Preliminaries

2.1. Structures

In a simple graph G we denote by $V(G)$ and $E(G)$ the sets of vertices and of edges, respectively. Let $\deg_G(v)$ denote degree of the vertex v in the graph G which is the number of adjacent vertices. An induced graph G' of G is a subgraph formed from a subset of vertices and all the edges between them that are present in G. For a set $X \subseteq V(G)$, by $G[X]$ we define the graph induced by vertices from X. Let $G \setminus v$ be the abbreviation for $G[V(G) \setminus \{v\}]$. G' is a *subgraph* of G, denoted by $G' \subseteq G$, if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. A *tree* T is a connected graph which has exactly $|V(T)| - 1$ edges. A *spanning tree* T of a graph G is a connected subgraph which includes all of the vertices of G, with the minimum possible number of edges. A *star* S is a tree of with at least 2 vertices for which at most one vertex has a degree greater than 1. A star of size at least 3 consists of a *center*, that is a vertex of the greatest degree, and *rays* — vertices of degree 1. Vertices of a star of size 2 are called *candidates*. For a given graph G, we say that S is a *spanning star forest* if $V(S) = V(G)$ and every connected component of S is a star.

2.2. Parameterized complexity

Parameterized complexity is a young branch of computational complexity theory. We refer the reader to textbooks of Downey and Fellows [1], Flum and Grohe [2] pr Cygan et al. [3], for an overview of the field.

We now introduce basic terminology. We begin with formally defining a parameterized problem. For the sake of clarity, all the definitions are taken from [3].

Definition 2.1. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbf{N}$, where Σ is a fixed, finite alphabet. For an instance $(x, k) \in L$, k is called the *parameter*.

Consider the example problems:

Definition 2.2. Independent Set: Given a graph G and a positive integer k, decide whether there exists a set I such that $|I| = k$ and $G[I]$ has no edges.

Definition 2.3. Dominating Set: Given a graph G and a positive integer k, decide whether there exists a set D such that $|D| \leq k$ and every vertex is either in D or is adjacent to one of the vertices from D.

There are multiple different parameters for a single problem. For example, Dominating Set can be parameterized by the sought size of dominating set k, or by the treewidth of the input graph.

Now, we want to introduce different complexity classes. The first one is called FPT (fixed parameter tractable). We say that a parameterized problem is in FPT if and only if it has an FPT algorithm defined below:

Definition 2.4. For a parameterized problem Q, an *FPT algorithm* is an algorithm $\mathcal{A}$ which, for any input (x, k), decides whether $(x, k) \in Q$ in time $f(k) \cdot n^c$ where c is a constant, independent of n, k, and f is a computable function.

Another important class of parameterized problems is XP. Similarly, a problem is in XP if and only if it has an XP algorithm defined below:

Definition 2.5. For a parameterized problem Q, an *XP algorithm* is an algorithm $\mathcal{A}$ which, for any input (x, k), decides whether $(x, k) \in Q$ in time $n^{f(k)}$ where f is a computable function.

Similar by polynomial-time reductions, we now introduce a *parameterized reduction*, that is, a notion of transforming instances of a certain parameterized problem to instances of another one.

Definition 2.6. Let $P, Q \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized languages. A *parameterized reduction* from P to Q is an algorithm $\mathcal{A}$ that given $(x, k) \in P$ outputs $(x', k') \in Q$ such that the following three conditions hold:

1. (x, k) is a YES-instance of P if and only if (x', k') is a YES-instance of Q.
2. $k' \leq g(k)$ for some computable function g.
3. The running time of $\mathcal{A}$ is $f(k) \cdot |x^c|$ for some computable function f and constant c.

Finally, we introduce the last family of complexity classes. *W-hierarchy* is an ascending chain of classes : W[1] $\subseteq$ W[2] $\subseteq$ W[3] $\subseteq$ .... For the purpose of this paper, we define W[1] as the closure of the Independent Set problem and W[2] as the closure of the Dominating Set problem. In other words, Independent Set parameterized by the size of independent set is W[1]-complete and Dominating Set parameterized by the size of dominating set is W[2]-complete with respect to parameterized reductions. There is a lemma that proves FPT $\subseteq$ W[1] and it is conjectured that this containment is strict.

Last but not least, we introduce a *kernelization algorithm* — a way of reducing the size of input instances in polynomial time:

Definition 2.7. A *kernel* for a parameterized problem Q is an algorithm $\mathcal{A}$ that, given an instance $(x, k) \in Q$, works in polynomial time and returns an equivalent instance $(x', k') \in Q$ such that $|x'| + k' \leq g(k)$ for a computable function g, called the *size* of the kernel.

2.3. Tree decomposition

Formally, a tree decomposition of a graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ where $\mathcal{T}$ is a tree whose every node t is assigned a vertex subset $X_t \subseteq V(G)$, called a *bag*, such that the following three conditions hold:

(T1) $\bigcup_{t \in V(T)} X_t = V(G)$.

(T2) For every $vu \in E(G)$ there exists a node t of $\mathcal{T}$ such that $v, u \in X_t$.

(T3) For every $v \in V(G)$ the set $T_v = \{t \in V(T) : v \in X_t\}$ induces a connected subtree of $\mathcal{T}$.

The *width* of a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, denoted $\mathrm{tw}(\mathcal{T})$, is equal to $\max_{t \in V(T)} |X_t| - 1$. The treewidth of a graph G, denoted $\mathrm{tw}(G)$, is the minimum width over all tree decompositions of G.

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

- $X_i = \emptyset$ if i is either the root or a leaf.
- Every non-leaf node is of one of the three following types:
  - **Introduce vertex node**: a node t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$ for some vertex $v \notin X_{t'}$.
  - **Introduce edge node**: a node t labeled with edge $vu \in V(G)$ such that $u, v \in X_t$ with exactly one child t' such that $X_t = X_{t'}$.
  - **Forget node**: a node t with exactly one child t' such that $X_t = X_{t'} \setminus \{v\}$ for some vertex $v \in X_{t'}$
  - **Join node**: a node t with exactly two children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$.

Note that every tree decomposition can be turned into a nice one without increasing the width in time $\mathrm{poly}(t) \cdot n$.

We distinguish one special case. If a tree $\mathcal{T}$ forms a path, we call it a *path decomposition*. Respectively, by $\mathrm{pw}(\mathcal{T})$ we denote a width of a path decomposition and by $\mathrm{pw}(G)$ we denote the minimum width over all path decompositions of G.

2.4. Satisfiability problem

In this section we present the CNF-SAT problem. We also show two hypotheses. They are fundamental to prove lower bounds for algorithms.

A *conjunctive normal form* (CNF) of a propositional formula ϕ on n Boolean variables $x_1, x_2, ..., x_n$ is a formula of form $\phi = C_1 \wedge C_2 \wedge ... \wedge C_m$, where C_i is a *clause*. Each clause C_i consists of disjunction of *literals*, that is, $C_i = l_1 \vee l_2 \vee ... \vee l_j$. A literal l_j corresponds to either a variable x_k or its negation $\neg x_k$. We also introduce an abbreviation $l_i \in C_j$ which means that a literal l_i occurs in C_j.

After this introduction, we are ready to finally formulate the CNF-SAT problem:

Definition 2.8. CNF-SAT: given a propositional formula ϕ on n Boolean variables $x_1, x_2, ..., x_n$ that is in *conjunctive normal form* decide whether there exists an evaluation of variables σ, such that $\sigma(\phi) = \text{True}$.

It is worth mentioning one more variant of a satisfiability problem. By restricting the number of literals in clauses to some constant q we arrive at the q-SAT problem. Observe that for $q \geq 3$, q-SAT is NP-Complete due to Cook-Levin Theorem. Let δ_q be a infinimum of the set of constants c for which there exists an algorithm solving q-SAT in time $\mathcal{O}^*(2^{cn})$. The *Exponential-Time Hypothesis* and *Strong Exponential-Time Hypothesis* are defined as follows:

Conjecture 2.1 (**Exponential-Time Hypothesis, ETH**).

$$\delta_3 > 0$$

Conjecture 2.2 (**Strong Exponential-Time Hypothesis, SETH**).

$$\lim_{q \to \infty} \delta_q = 1$$

Intuitively, ETH states that we need to browse through exponential number of assignments for q-SAT while SETH implies that as the number of literals in clauses grows, brute force check is inevitable. However, in this paper we do not refer to the above consequences, but to the following consequence of Conjecture 2.2

Conjecture 2.3. CNF-SAT *cannot be solved in time* $\mathcal{O}^*((2-\epsilon)^n)$ *for some* $\epsilon > 0$.

Chapter 3

Spanning Star Forest Problem

In this chapter we examine both decision and constructive variant of Spanning Star Forest. We propose an algorithm working in linear time that outputs a spanning star forest or concludes that the given instance is a NO-instance.

3.1. Decision variant

In the decision variant of Spanning Star Forest, all that we have to do is to answer whether there exists a spanning star forest of an input graph. As it turns out, every graph that does not contain any isolated vertex has a spanning star forest.

Lemma 3.1. *A graph G has a spanning star forest if and only if it does not contain any isolated vertices.*

Proof. If G has a spanning star forest S, then we have that for all $v \in V(G)$, $1 \leq deg_S(v) \leq deg_G(v)$. Thus, none of the vertices is isolated.

For the opposite direction, we prove the lemma by induction on $|V(G)|$. Assume $|V(G)| = 2$. The statement trivially holds because a graph consisting of one edge and two vertices is a correct spanning star forest. Let $|V(G)| > 2$. For the induction step, we split the proof into two parts.

Firstly, suppose that for all vertices $v \in V(G)$, it holds that $\deg_G(v) = 1$. Clearly, G is a matching. Hence, it is a spanning star forest by itself.

Now, suppose that there exists a vertex u such that $\deg_G(u) > 1$. Let $C \subseteq G$ be the connected component satisfying $u \in V(C)$. Based on the degree of u, we infer that $|V(C)| > 2$. Let T be an arbitrary spanning tree of C and v be one of its leaf. Observe that $T \setminus v$ is a spanning tree of $C \setminus v$. So, $C \setminus v$ does not have any isolated vertices and neither has the graph $G \setminus v$. Now, from the induction, let S be a spanning star forest of the graph $G \setminus v$, u be a vertex such that $uv \in E(G)$ and let $w \in N_S[u]$. Consider the two following cases:

1. Suppose u is a ray in S. This implies that w is a center and $deg_S(w) \geq 2$. Then, $S' = \big(V(S) \cup \{v\}, (E(S) \cup \{uv\}) \setminus \{uw\}\big)$ is a spanning star forest for the graph G.

2. Otherwise, u is either a candidate or a center. Then, $S' = \big(V(S) \cup \{v\}, E(S) \cup \{uv\}\big)$ is a spanning star forest of the graph G. □

Application of Lemma 3.1 yields the following result for Spanning Star Forest.

Corollary 3.1. *The decision variant of* Spanning Star Forest *can be solved in linear time.*

Proof. Given a graph $G = (V, E)$ the answer is YES if for all $v \in V(G)$ $\deg_G(v) \neq 0$ and NO otherwise. $\square$

3.2. Constructing a solution

In the previous section, we gave an algorithm that only determines the existence of a solution. Now, we focus on constructing an arbitrary solution for a given instance. We propose an algorithm that for a graph outputs a spanning star forest in linear time if it exists. Firstly, let us introduce two claims:

Claim 3.1. *If $C_1, C_2, ...C_n$ are the connected components of a graph G and $S_1, S_2, ..., S_n$ are their spanning star forests respectively, then $\bigcup_{i=1}^{n} S_i$ is a spanning star forest for G.*

Claim 3.2. *A connected graph G has a spanning star forest if and only if its spanning tree T has.*

The first claim can be trivially proven by the definition of a spanning star forest while the second one follows directly from Lemma 3.1. Equipped with this information, we present an algorithm which solves the problem for connected graphs.

```
Input: connected graph G such that |V(G)| ≥ 2
Output: spanning star forest of G
spanned ← new Array[|V(G)|];
T ← SpanningTree(G);
S ← ∅;
for v: postorder(T) and v is not the root do
    if not spanned[v] then
        u ← parent(T, v);
        S ← S ∪ {uv};
        spanned[v] = True;
        spanned[u] = True;
    end
end
v ← root(T);
if not spanned[v] then
    u ← arbitrary node vertex such that v = parent(T, u);
    S ← S ∪ {uv};
end
return S;
```

Algorithm 1: Obtaining a spanning star forest from a connected graph.

Firstly, the algorithm creates a spanning tree T, say rooted. Then, it does a simple bottom-up traversal. If the current node v has not been added to the solution yet, the algorithm adds the edge connecting it with parent. If the root has not been added to the solution during the for loop, we add an arbitrary edge incident to it, which finishes the algorithm.

Now we need to check that the obtained graph is a spanning star forest. There is one non-trivial operation that the algorithm does. Specifically, if the root has not been added during the for loop, we connect the root to any existing star without checking whether it remains a correct star. Before we proceed to the lemma about the correctness of Algorithm 1, let us prove the following claim:

Claim 3.3. *Suppose that a connected graph G is the input for Algorithm 1. Let T be a spanning tree obtained during SpanningTree(G) procedure and S be the output graph. If $u_1u_2, u_2u_3 \in E(S)$, $u_2 = parent(T, u_1)$ and $u_3 = parent(T, u_2)$, then u_3 is the root and u_3 has exactly one neighbor in S.*

Proof. Observe that no two consecutive parents can be added during the for loop. Thus, edge u_2u_3 must have been added in the if statement. Since $u_3 = \text{parent}(T, u_2)$, u_3 must be the root. Moreover, having known that the root becomes spanned by S for the first time during the if statement, we conclude that u_2 is the only neighbor of u_3 in S. □

Lemma 3.2. *Algorithm 1 ran on a connected graph G satisfying $|V(G)| \geq 2$ outputs a spanning star forest S for G.*

Proof. To prove the lemma, we need to show that all of the four following conditions hold after a successful execution of the algorithm:

1. S does not consist of any cycle.
2. S spans G, i.e. $V(S) = V(G)$.
3. S does not have any isolated vertices.
4. S does not contain a path of length 3.

Let $T \subseteq G$ be a tree created during SpanningTree(G) procedure. Trivially, S does not contain a cycle because S is a subgraph of T, which is a tree. Now, observe that the algorithm iterates over all vertices and, except for the root, pairs every vertex with its parent. The last pair, the root and its child, is added either in the for loop or in the if statement. Thus, we conclude that S does not have any isolated vertices.

Finally, we prove that S does not contain a path of length 3. Note that such a path would need to contain a vertex v, its parent u and its grandparent w. From the Claim 3.3 we infer that w is the root and u is the only neighbour of w in S. Now, observe that if any other child of u existed in that star, the last condition would still hold. So, suppose that a child of v is in the same component. Contradiction, because u would be the root then. Therefore, there does not exist a path of length 3 and we conclude that S is a spanning star forest for G. □

Having proven the correctness of Algorithm 1, we proceed to the complexity analysis i.e. we prove Theorem 1.1.

Theorem 1.1. Spanning Star Forest *can be solved in linear time. Moreover, given a graph G, one can find a solution in linear time if it exists.*

Proof. Given a graph G we run Algorithm 1 on every connected component of G. Then, we merge the obtained spanning star forest in linear time. Notice that an arbitrary spanning tree of any connected component can be found in linear time. The main loop of the algorithm has $n - 1$ iterations, where n is the number of vertices of the component, because every vertex is processed once. Moreover, it takes constant time to finish one iteration. Thus, the total run time is linear. □

Thus, obtaining a spanning star forest without any limitations is easy. Both the decision and the constructive variant of the problem can be solved in linear time.

Chapter 4

Minimal Spanning Star Forest problem

In Minimal Spanning Star Forest, given a graph G and a natural number k, the objective is to determine whether there exists a spanning star forest S such that the number of connected components of S is at most k.

It is natural to ask whether one can find a solution that minimizes the number of connected components. The problem formulated in that way resembles Dominating Set. At first glance, one can say that a center corresponds to a dominating vertex whereas a ray corresponds to a dominated vertex. Candidates corresponds to either a dominating or a dominated vertex. However, in Dominating Set isolated dominating vertices are allowed and some vertices can be dominated by multiple neighbors.

To give a systematic parameterized reduction between these two problems, we need to get a better understanding of Dominating Set.

Definition 4.1. Given a graph G and a dominating set D, a domination mapping is a function $\mu : V(G) \setminus D \to D$ such that satisfies $(x, \mu(x)) \in E(G)$ for all $x \in V(G) \setminus D$.

Lemma 4.1. *Let G be a graph without isolated vertices and let D be a dominating set in G of minimum size. Then, there exists a domination mapping μ such that μ is surjective.*

Proof. Let μ be a dominating mapping that maximizes $|\operatorname{Im}\mu|$. If μ is surjective, then the proof is finished. Otherwise, there exists a vertex $v \in D$ such that $v \notin \operatorname{Im}\mu$. Consider the following cases:

1. Suppose $N_G(v) = \emptyset$. Contradiction, G has no isolated vertices. Let u be any neighbor of v.

2. Suppose $u \in D$. Contradiction, D was assumed to be a dominating set of minimum size whereas $D \setminus \{v\}$ is a smaller dominating set.

3. Suppose $u \notin D$ and let $w = \mu(u)$. If $|\mu^{-1}(w)| = 1$, then $((D \setminus \{v, w\}) \cup u)$ is a smaller dominating set for the graph G. Contradiction.

4. Finally, suppose $|\mu^{-1}(w)| > 1$. Then, the mapping:

$$\mu'(x) = \begin{cases} v, & \text{if } x = u \\ \mu(x), & \text{otherwise} \end{cases}$$

   is a domination mapping that satisfies $\operatorname{Im}\mu \subsetneq \operatorname{Im}\mu'$. Contradiction, we assumed that μ is a dominating mapping that maximizes $|\operatorname{Im}\mu|$.

Since all the cases led to a contradiction we conclude that there exists a domination mapping μ such that μ is surjective. $\square$

In addition, we show one reduction rule that removes unnecessary vertices:

Claim 4.1. *Let (G, k) be an instance of* Minimal Spanning Star Forest *and $I \subseteq V(G)$ be the set of isolated vertices in G. Then, (G, k) is a YES-instance if and only if $(G \setminus I, k - |I|)$ is a YES-instance.*

Claim 4.1 follows by observing that every isolated vertex must be included in the dominating set. Equipped with the above information, we are ready to show the parameterized reduction:

Lemma 4.2. *There exists a parameterized reduction that takes an instance (G, k) of* Dominating Set *and returns an instance (G', k') of* Minimal Spanning Star Forest *such that $G' \subseteq G$ and $k' \leq k$.*

Proof. Firstly, we modify the instance. By Claim 4.1, let $(G,', k') = (G \setminus I, k - |I|)$ be the instance without isolated vertices. If $k' < 0$ we conclude that (G, k) is a NO-instance. Otherwise, we claim that (G', k') is a YES-instance of Dominating Set if and only if (G', k') is a YES-instance of Minimal Spanning Star Forest.

Consider the backward implication. Suppose S is a spanning star forest for (G', k'). We create the dominating set D as follows: for every star in S of size 2 pick an arbitrary candidate and for every star of size greater than 2 pick its center. Obviously, $|D| \leq k'$ because there are at most k' stars in S. Moreover, observe that every vertex $v \in V(G') \setminus D$ is either a ray or a candidate in S. Thus, there exists an edge $vu \in E(G')$ where $u \in D$.

To prove the forward implication, let D be a minimum size dominating set for (G', k'). By Lemma 4.1, there exists a domination mapping μ that is surjective. Now, we claim that the graph $S = (V(G'), \{x\mu(x) : x \in V(G') \setminus D\})$ is a solution for the instance (G', k') of Minimal Spanning Star Forest. Observe that S has at most k' components as $|D| \leq k'$ By surjectivity, there are no isolated vertices in S because dominated vertices are paired with dominating ones. Moreover, μ maps vertices from $V(G') \setminus D$ to D. Thus, we obtain that for all $v \in V(G') \setminus D$, $\deg_S(v) = 1$ and there does not exist an edge in S connecting two dominating vertices. Thus, S is a spanning star forest. $\square$

The problems look so similar that one could ask whether there exists a reverse parameterized reduction. Indeed, this is true and the following lemma formally proves it:

Lemma 4.3. *There exists a parameterized reduction that takes an instance (G, k) of* Minimal Spanning Star Forest *and returns an instance (G, k') of* Dominating Set *such that $k' \leq k$.*

Proof. Let (G, k) be an instance of Minimal Spanning Star Forest. If G contains an isolated vertex, then return $(G, 0)$. Otherwise, return (G, k). Now, we claim that (G, k) is a YES-instance of Minimal Spanning Star Forest if and only if (G, k') is a YES-instance of dominating set where $k' = 0$ or $k' = k$.

Consider the forward implication. Let S be a spanning star forest of at most k stars for the graph G. Then, by Lemma 3.1, G does not have any isolated vertices and $k' = k$. Observe that G does not change during the reduction. We create a dominating set D as follows: for every star of size 2 pick an arbitrary candidate and for every star of size at least 3 pick its center. Obviously, $|D| \leq k$ because S contains at most k stars. So, suppose that there exists $v \in V(G) \setminus D$ that is not dominated. However, S spans G which means that v is in one of the stars. Therefore, not only v is either a ray or a candidate in S, but also there exists an edge $vu \in E(S)$ such that u is either a center or a candidate. By the definition of D, $u \in D$. Contradiction because u dominates v.

For the backward implication, let D be a minimum size dominating set for (G, k'). Note that G has no isolated vertices and $k' = k$. By Lemma 4.1, there exists a domination mapping μ such that μ is surjective. Now, we claim that the graph $S = (V(G), \{x\mu(x) : x \in V(G) \setminus D\})$ is a spanning star forest. S spans G as it contains all the vertices from G. Moreover, there are no isolated vertices because for every vertex $v \in V(G) \setminus D$ there exists exactly one vertex $u \in D$ such that $vu \in E(S)$ and for every vertex $u \in D$ there exists at least one vertex $v \in V(G) \setminus D$ such that $vu \in E(S)$. From the previous sentence we also infer that the mapping forces every connected component to be a star, which concludes the proof. □

Provided that there exist reductions from Dominating Set to Minimal Spanning Star Forest and from Minimal Spanning Star Forest to Dominating Set we are ready to prove the main results.

Theorem 1.2. Minimal Spanning Star Forest *is* W[2]*-complete.*

Proof. Recall, that Dominating Set is a W[2]-complete problem. Note that by Lemma 4.2 and Lemma 4.3 the problems are equivalent with respect to parameterized reductions. Thus, Minimal Spanning Star Forest is W[2]-complete. □

Moreover, note that the parameterized reductions stated in the previous lemmas can be considered as polynomial-time reductions. Hence, we get:

Theorem 1.3. Minimal Spanning Star Forest *is* NP-complete.

Proof. Dominating Set is an NP-complete problem. As observed in the previous proof, Dominating Set and Minimal Spanning Star Forest are equivalent with respect to polynomial reductions. Therefore, we conclude that Minimal Spanning Star Forest is also NP-complete. □

Interreducibility is a useful tool. Especially, if one of the problems has been deeply studied in the past. As an example, we show how to transfer a lower bound for the running time from Dominating Set to Minimal Spanning Star Forest. Firstly, we prove the existence of an algorithm that works in time stated in Theorem 1.4:

Theorem 1.4. Minimal Spanning Star Forest *can be solved in* $\mathcal{O}^*(N^{k+o_k(1)})$*, where* N *is the number of vertices.*

Proof. Let (G, k) be an instance of Minimal Spanning Star Forest. We apply the reduction from Lemma 4.3 and obtain an instance (G, k') of Dominating Set. It is known that there exists an algorithm solving Dominating Set in time $\mathcal{O}(N^{k+o_k(1)})$ [5]. This concludes the proof. □

Now, consider the following theorem proven by Pătraşcu and Williams [4]:

Theorem 4.1. *Unless* CNF-SAT *cannot be solved in time* $\mathcal{O}^*((2-\epsilon')^n)$ *for some* $\epsilon' > 0$*, there does not exist constants* $\epsilon > 0, k \geq 7$ *and an algorithm solving* Dominating Set *on instance with parameter equal to* k *that run in time* $\mathcal{O}(N^{k-\epsilon})$*, where* N *is the number of vertices of the input graph.*

Armed with the theorem and reductions, we are ready to prove the last result for Minimal Spanning Star Forest:

Theorem 1.5. *Unless* CNF-SAT *cannot be solved in time* $\mathcal{O}^*((2-\epsilon')^n)$ *for some* $\epsilon' > 0$*, there does not exist constants* $\epsilon > 0, k \geq 7$ *and an algorithm solving* Minimal Spanning Star Forest *on instance with parameter equal to* k *in time* $\mathcal{O}(N^{k-\epsilon})$*, where* N *is the number of vertices of the graph.*

Proof. Assume CNF-SAT cannot be solved in the stated time. However, suppose there exist a constant $\epsilon > 0$ and an algorithm $\mathcal{A}$ that solves Minimal Spanning Star Forest instance in time $\mathcal{O}(N^{k-\epsilon})$, where $k \geq 7$ and $\epsilon > 0$ are fixed. Let (G, k) be an instance of Dominating Set. We show an algorithm that contradicts Theorem 4.1. Firstly, we apply the reduction stated in Lemma 4.2. We obtain an instance (G', k') of Minimal Spanning Star Forest such that $G' \subseteq G$ and $k' \leq k$. Then, we can apply algorithm $\mathcal{A}$ to obtain the answer in time $\mathcal{O}(N^{k'-\epsilon})$, where $N = |V(G')| \leq |V(G)|$ and $k' \leq k$. Contradiction. □

Chapter 5

Spanning Star Forest Extension

In this chapter, we study a significantly different variant of the SPANNING STAR FOREST problem. Let G be a graph and $F \subseteq E(G)$ be a set of *forced edges*. In the SPANNING STAR FOREST EXTENSION we ask whether there exists a spanning star forest S such that $F \subseteq E(S)$.

In further, we denote by F a set of *forced edges*. Vertices that have exactly one forced edge are called *forced candidates*. Similarly, if a subset of F forms a *forced star* of size greater than 2, then we call its particles a *forced center* and *forced rays* consequently. We denote by F_R a set of all forced rays and by F_C a set of all forced centers. Vertices that does not belong to $V(F)$ are called *free vertices* and they are denoted by U.

We consider three different parameters for this problem: the number of forced edges, the number of free vertices and the treewidth.

5.1. Instance normalization

Notice that this time we do not have any restriction on the number of connected components. The hardness of the problem lies in choosing which of the forced candidates should become centers and which should become forced rays.

In this section we propose a definition of a *normalized instance* — an instance which satisfies a set of conditions described below. Note that an *induced matching* M in a graph G is a set of disjoint edges such that there are no edges outside of M with both endpoints in $V(M)$.

Definition 5.1. A pair (G, F) is normalized if the following conditions hold:

1. G does not have isolated vertices.
2. F is an induced matching.
3. For every $u \in U$, all neighbors of u are in $V(F)$.

Surprisingly, every instance of SPANNING STAR FOREST EXTENSION can be either normalized or discarded as a NO-instance. This is explained in the following lemma:

Lemma 5.1. *There is an algorithm working in polynomial time that takes an instance (G, F) of* SPANNING STAR FOREST EXTENSION *and either conclude that the instance is a NO-instance or outputs an equivalent normalized instance (G', F') satisfying $G' \subseteq G$ and $F' \subseteq F$ and $U' \subseteq U$, where U' is the set of free vertices in G'.*

Proof. We begin by showing cases for which we conclude that (G, F) is a NO-instance:

Reduction 1 If graph G contains an isolated vertex, then (G, F) is a NO-instance.

Obviously, an isolated vertex cannot be a star by Lemma 3.1. Now, observe, that in the extension variant, there exists a set of edges that must be added to the solution. Therefore, we can instantly conclude that an instance is a NO-instance if F forms a forbidden subgraph.

Reduction 2 If F contains a path or a cycle of length at least 3, then (G, F) is a NO-instance.

Now, let us show three reduction rules. After their exhaustive application, the set of forced edges becomes an induced matching. Firstly, we remove free edges between forced vertices:

Reduction 3 Remove the set of edges $\{vu : v, u \in V(F),\ vu \in E(G) \setminus F\}$.

Clearly, if such an edge was included in a solution S, then the solution would contain a path or a cycle of length 3. Thus, the operation is safe.

Now suppose that a subset of forced edges forms a star of size at least 3. Then, the forced center is already determined. Hence, we can remove from the instance all the free edges that have at least one end in a forced ray.

Reduction 4 Remove the set of edges $\{uv : v \in F_R,\ uv \in E(G) \setminus F\}$

We claim that the operation is safe. To prove it, suppose contrary. Let $u \in V(G)$, $v \in F_R$ and $uv \in E(G) \setminus F$. Now, suppose that there exists a solution S such that $uv \in E(S)$. However, v is a forced ray. So there exists a vertex $c \in F_C$ and $v' \in F_R$, such that $v \neq v'$ and $vc, cv' \in F$. Moreover, $vc, cv' \in E(S)$. If $u = v'$, then S contains a cycle of length 3. Otherwise, uv, vc, cv form a path of length 3. Thus, S is not a spanning star forest.

Observe that after exhaustive application of the above rules, every $v \in F_R$ satisfies $\deg_G(v) = 1$. Every forced ray is connected to its forced center only. Hence, for every forced star of size greater than 2 we can remove all forced rays except for one.

Reduction 5 Suppose that (G, F) is the output graph after exhaustive application of the previous reductions. For every forced star with more than 2 vertices, remove all the forced rays except for one.

We prove the safeness now. Let (G', F') be the output instance after application of Reduction 5. We claim that (G', F') has a spanning star forest if and only if (G, F) has a spanning star forest.

For the forward implication, let S be a solution for (G', F'). Let S_C be a set of centers in S. Recall that by F_C we denote the set of forced centers in G and by F_R we denote the set of forced rays. We claim that $S_C \subseteq F_C$. Indeed, observe that Reduction 4 removes all the free edges that has at least one end in a forced ray. So it follows that for all $v \in V(G') \cap F_R$, $\deg_{G'}(v) = 1$ and $\deg_{G'}(v) \geq \deg_S(v) \geq 1$. Therefore, we conclude that $S \cup (F \setminus F')$ is a spanning star forest for G as we always connect a removed vertex to a center or a candidate.

Conversely, let S be a spanning star forest for (G, F). Now, let $S' \subseteq S$ be a subgraph restricted to vertices of G'. Observe that every forced star in G remains a forced star in G' as F' is a matching. Moreover, removed vertices do not have any edges to free vertices in G by Reduction 4. Thus, S' is a spanning star forest for the graph G'.

Let us summarize the work and describe how the instance looks like after exhaustive application of Reductions 1-5:

Claim 5.1. *Given an instance (G, F), if Reductions 1-2 do not yield that a graph is a NO-instance, then exhaustive application of Reductions 3-5 outputs (G', F') such that F' is an induced matching in G'.*

Proof. We prove the claim by contradiction. Firstly, suppose that F is not a matching. Hence, there exists a connected component with more than 2 vertices. It must be a star because Reduction 2 does not yield that (G, F) is a NO-instance. Contradiction, we did not apply exhaustively Reduction 5 to decrease the size of each forced star. Now, suppose that F is not an induced matching. Hence, there exist vertices $v, u \in V(F)$ such that $vu \in E(G) \setminus F$. Contradiction, Reduction 3 has not been applied exhaustively. $\square$

Now, let us focus on the second part of the graph i.e. free vertices. As we have already seen, by Lemma 3.1, there exists a spanning star forest if and only if there are no isolated vertices. Let $V_P = \{u : \text{there exists } v \in U \text{ such that } uv \in E(G)\}$ and $V_{NP} = V(G) \setminus V_P$. Finally, $G_{NP} = G[V_{NP}]$ and $G_P = G[V_P]$. Now, we claim that:

Claim 5.2. *G_P has a spanning star forest.*

Proof. By the definition of G_P, Every vertex has at least one neighbor in G_P. Hence, by Lemma 3.1 G_P has a spanning star forest. $\square$

Observe that during partitioning G into G_P and G_{NP} we lose the information about some edges. Specifically, let $L = \{vu : vu \in E(G), v \in V(G_{NP}),\ u \in V(G_P)\}$ be the set. Note that vertices from G_{NP} that are incident to L are the forced vertices. Additionally, both forced vertices and vertices from G_P are already satisfied i.e. we can always span them by a star forest. Therefore, we can formally state the following claim:

Claim 5.3. *Let (G, F) be an instance of* Spanning Star Forest Extension *after exhaustive application of Reductions 1-5. Then (G, F) has a solution if and only if (G_{NP}, F) has one.*

Proof. For the backward implication, suppose S is a solution for (G_{NP}, F). We partition G into G_P and G_{NP}. By Lemma 5.2, let S' be a solution for G_P. Then, $S \cup S'$ is a spanning star forest for G.

Now, consider the forward implication. Let S be a solution for (G, F) and let $S' = S \cap E(G_{NP})$, be a subgraph restricted to the edges of G_{NP} only. Observe that S' is a star forest. If S' is a spanning star forest for G_{NP} then we conclude. Otherwise, there exists a vertex v such that $v \in V(G_{NP}) \setminus V(S')$. Note that v is a free vertex because all the forced vertices are spanned by forced edges from F. However, by definition of G_P, v has no neighbors outside G_{NP}. Hence, $v \notin V(S)$ and S is not a spanning star forest for G. Contradiction. $\square$

Recall that free vertices of G_{NP} are not adjacent to other free vertices. Thus, the instance (G_{NP}, F) satisfies the last condition of Definition 5.1.

To conclude, let (G, F) be an input graph. If Reduction 1 or Reduction 2 applies to (G, F) then it is a NO-instance. Otherwise, we apply Reductions 3-5 exhaustively, in order, and obtain (G', F') . Finally, we partition G' into G'_{NP} and G'. As an output we return an instance (G'_{NP}, F') that follows $G'_{NP} \subseteq G$, $F' \subseteq F$ and $U' \subseteq U$, where U' is the set of free vertices in G'_{NP}. $\square$

Finally, we want to point out an advantage of a normalized instance of Spanning Star Forest Extension over an arbitrary one. Consider the following claim:

Claim 5.4. *Let (G, F) be a normalized instance of* Spanning Star Forest Extension. *Then, (G, F) is a YES-instance if and only if there exists an independent set $C \subseteq V(F)$ such that $U \subseteq N(C)$.*

Proof. For the forward implication, let S be a spanning star forest for (G, F). We claim that $C = N_S(U)$ satisfies the required condition. Clearly, $C \subseteq V(F)$ as free vertices have edges to forced vertices only. Additionally, for every forced edge $vu \in F$ either v or u does not have edge to free vertices in S. Thus, C is an independent set.

For the backward implication, observe that F is an induced matching. If $U \subseteq N[C]$, then for every $v \in U$ there exists a vertex $u \in C$ such that $vu \in E(G)$. Thus, we can add every free vertex to one of the existing stars. Let $S \subseteq G$ be a subgraph that takes all the forced edges and, for every free vertex, takes exactly one edge to a forced vertex from C. Such edges exist because $U \subseteq N(C)$. Observe that in S, for every $uv \in F$, the degree of at most one vertex is greater than 1 because C is an independent set. Thus, S is spanning star forest for (G, F). □

5.2. NP-completeness

In this section we present a parameterized reductions from Spanning Star Forest Extension to CNF-SAT. As it turns out, the number of forced edges corresponds to the number of variables and the number of free vertices corresponds to the number of clauses.

Lemma 5.2. *There exists a polynomial time reduction that takes an instance ϕ of* CNF-SAT, *say with n variables and m clauses, and returns an equivalent normalized instance (G, F) of* Spanning Star Forest Extension *such that $|V(G)| = 2n + m$, $|F| = n$ and $|U| = m$.*

Proof. Firstly, we present the reduction. Suppose ϕ is the input instance of CNF-SAT. As ϕ is a CNF formula, let Clauses $= \{C_1, ..., C_m\}$ be the set of clauses and let Variables $= \{x_1, ..., x_n\}$ be the set of variables in ϕ. For every clause C_i we introduce a vertex $v[C_i]$ and for every variable x_i we introduce two vertices $v[x_i], v[\neg x_i]$ and a forced edge $v[x_i]v[\neg x_i]$. Now, for every occurrence of a literal l_i in a clause C_j we introduce a free edge $v[C_j]v[l_i]$. Finally, we say that (G, F), the graph that we described, has a spanning star forest that extends F if and only if there exists an assignment satisfying the formula ϕ.

Firstly, we prove that (G, F) is a normalized instance that satisfies the constrains on the number of vertices. Since for every variable from ϕ we introduced one edge we have that $|F| = n$. Furthermore, for every clause we introduced exactly one vertex so $|U| = m$. Thus, we get that $V(G) = 2n + m$. Now we claim that (G, F) is normalized. There are no isolated vertices because every clause has at least one literal and variables corresponds to forced edges. Secondly, F is an induced matching because, apart from F, vertices introduced for literals (forced vertices) are connected to vertices introduced for clauses (free vertices) only. And finally, no edges were introduced between vertices introduced for clauses (free vertices).

We now check equivalence of the instances. For the forward implication, let S be a solution for (G, F). We create the evaluation σ as follows:

$$\sigma(x_i) = \begin{cases} \text{True, if } \deg_S(v[x_i]) > 1 \\ \text{False, otherwise} \end{cases}$$

We claim that the evaluation satisfies ϕ. Fix an arbitrary clause C_i. Now, observe that there exists a literal $l_j \in C_i$ such that $v[l_j]v[C_i] \in S$. Then, $\deg_S(l_j) > 1$. By the definition of σ,

$\sigma(l_j) = \text{True}$, and therefore $\sigma(C_i) = \text{True}$. We conclude that $\sigma(\phi) = \text{True}$ as we proved that an arbitrary clause in the formula is satisfied.

To prove the backward implication, assume there exists an evaluation σ of variables that satisfies the formula. If $\sigma(\phi) = \text{True}$, then for every $C_i \in \text{Clauses}$ there exists a literal $l_i \in C_i$ such that $\sigma(l_i) = 1$. Now, let $L = \{v[l] : \sigma(l) = 1\}$. Clearly, $\{v[C_i] : C_i \in \text{Clauses}\} \subseteq N_G(L)$ because σ satisfies the formula ϕ. Moreover L is an independent set in G because for every forced edge $v[x_i]v[\neg x_i] \in F$ either $\sigma(x_i) = \text{False}$ or $\sigma(\neg x_i) = \text{False}$. Hence, by Lemma 5.4, there exists a spanning star forest for (G, F). □

There is one more observation that we want to point out in this section. Since a CNF-formula is trivially encoded as a spanning star forest extension instance, one can ask if the problems are interreducible. Indeed, this is true and we present the backward reduction.

Lemma 5.3. *There exists a polynomial time reduction that takes an instance (G, F), such that (G, F) has n forced edges and m free vertices, and returns a formula ϕ of* CNF-SAT *such that ϕ has at most n variables and at most m clauses.*

Proof. Firstly, we apply Lemma 5.1 to normalize the instance. If it yields that an instance is a NO-instance, we return a formula $(x \wedge \neg x)$. Otherwise, let (G', F') be the output of the exhaustive application of the reduction rules. Observe that $G' \subseteq G$ and $F' \subseteq F$ and $U' \subseteq U$. Now, we proceed to a the construction of a formula. For every forced edge vu we introduce a variable x_{vu} and we arbitrarily label its ends as x_{vu} and $\neg x_{vu}$. Now, for every free vertex w we introduce a clause C_w consisting of a disjunction of literals labels($N_G(w)$). Finally, we claim that (G, F) has a spanning star forest if and only if the formula ϕ, described above, is satisfiable.

Observe that the instances are equivalent to the instances described in Lemma 5.2. One can follow the reasoning as in the previous reduction. □

Lemma 5.2 and Lemma 5.3 directly imply the following:

Theorem 1.6. Spanning Star Forest Extension *is* NP-*complete.*

5.3. Parametrization by the number of forced edges

In this section, in addition to an instance (G, F) we receive a parameter k which is equal to the number of forced edges. We show two major results: Spanning Star Forest Extension parameterized by the number of forced edges does not admit a kernel of polynomial size and a lower bound under Strong Exponential Hypothesis.

5.3.1. Lower bound for a running time

Previously in this chapter, we proved that Spanning Star Forest Extension is NP-complete. We showed that the problem is NP-complete, it does not admit a polynomial kernel and we stated reduction rules to simplify instances. In this subsection, we show a simple routine that solves Spanning Star Forest Extension parameterized by the number of forced edges. Furthermore, we prove that there does not exist a faster algorithm unless CNF-SAT cannot be solved in time $\mathcal{O}^*((2-\epsilon)^n)$, for $\epsilon > 0$.

Consider the following Algorithm 2. It simply iterates over all maximal independent sets of forced candidates. If a set spans all the vertices, then it means that the set of forced edges can be extended to a spanning star forest. Otherwise, if none of the sets satisfies the condition, then the input is a NO-instance. Now, see the following lemma:

Data: normalized instance (G, F)
Result: spanning star forest of G extending F
$Centers \leftarrow \{C : C \subseteq V(F),\ |C| = |F| \text{ and } \forall u, v \in C,\ vu \notin F\}$;
for $C \in Centers$ **do**
  if $U \subseteq G(C)$ **then**
    return YES-instance;
  end
end
return NO-instance;

Algorithm 2: Extending a spanning star forest from a normalized graph.

Lemma 5.4. *Given a normalized instance (G, F) parameterized by $|F|$, Algorithm 2 outputs the answer whether (G, F) has a spanning star forest.*

Proof. Lemma 5.4 proves the correctness of the algorithm. □

With Lemma 5.4, we can proceed to the next theorem stated in the introduction:

Theorem 1.7. Spanning Star Forest Extension *parameterized by the number of forced edges can be solved in time $\mathcal{O}^*(2^{|F|})$ where $|F|$is the number of forced edges.*

Proof. Firstly, we normalize the input instance by Lemma 5.1 which takes a polynomial time. We either conclude that an instance is a NO-instance or obtain (G, F). Then, we apply Algorithm 2. It does at most $2^{|F|}$ iterations because this is the number of different maximal independent sets in an induced matching. Every iteration takes polynomial time to process the set. Hence, we conclude that the algorithm works in time $\mathcal{O}^*(2^{|F|})$. □

Note that the described algorithm is a simple brute force. We do not optimize the search. Moreover, there is no need to fight for a better complexity unless SETH fails. The following theorem proves it:

Theorem 1.8. *There exists an algorithm solving* CNF-SAT *in time $2^{o(n)}$, where n is the number of variables, if and only if there exists an algorithm solving* Spanning Star Forest Extension *parameterized by the number of forced edges in time $2^{o(n)}$, where n is the number of forced edges.*

Proof. Suppose $\mathcal{A}$ is an algorithm that solves CNF-SAT in time $2^{o(n)}$, where n is the number of variables. Now, we show an algorithm solving Spanning Star Forest Extension parameterized by the number of forced edges. Let (G, F) be an arbitrary instance of Spanning Star Forest Extension with a parameter equal to $|F|$. We apply the reduction from Lemma 5.2 and obtain a formula ϕ, that has exactly $|F|$ variables. Then, we apply algorithm $\mathcal{A}$ to obtain the result. Observe that the reduction works in polynomial time. Thus, the above algorithm works in time $2^{o(n)}$.

For the converse implication, assume $\mathcal{A}$ is an algorithm that solves Spanning Star Forest Extension parameterized by the number of forced edges in time $2^{o(n)}$, where n is the number of forced edges. We show an algorithm for CNF-SAT problem now. Let ϕ be an arbitrary CNF-formula with n variables. We apply the reduction from Lemma 5.3 and obtain a normalized instance (G, F) of Spanning Star Forest Extension, such that $|F| = n$. Now, we run the algorithm $\mathcal{A}$ on (G, F), and hence we get the answer. The described algorithm works in time $2^{o(n)}$. □

5.3.2. Lower bound for a kernel

In this section, we prove that Spanning Star Forest Extension parameterized by the number of forced edges does not admit a polynomial kernel unless some classes collapse. To achieve this, we show two different approaches. Firstly, we show a proof based on parameterized reductions stated in previous subsection. Consider the following lemma:

Lemma 5.5. CNF-SAT *parameterized by the number of variables has a polynomial kernel if and only if* Spanning Star Forest Extension *parameterized by the number of vertices has one.*

Proof. For the forward implication, let $\mathcal{A}$ be such kernelization algorithm. Now, we show a kernelization algorithm for Spanning Star Forest Extension problem. Let (G, F) be an arbitrary instance. We apply the reduction from Lemma 5.3 and obtain a formula ϕ of $|U|$ clauses and $|F|$ variables. Now, we apply the algorithm $\mathcal{A}$ and obtain a formula ϕ' such that $|\phi'| \leq \text{poly}(|F|)$. Finally, we create an instance (G', F') by the reduction from 5.2. Note that now, $|G'| + |F'| \leq \text{poly}(|F|)$ as the second reduction does not change the size.

Observe that for the converse implication, one can use the same reasoning. □

Now, we state the following result of Fortnow and Santhanam [11]:

Theorem 5.1. CNF-SAT *is not polynomially kernelizable unless NP $\subseteq$ coNP/poly.*

Immediately, we get:

Theorem 1.9. Spanning Star Forest Extension *parameterized by the number of forced edges does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly.*

Proof. The theorem follows directly from Lemma 5.5 and Theorem 5.1. □

The second approach that we show is completely different. We show a notion of *cross-composition.* It is a framework for proving kernelization lower bounds. A technique, firstly introduced in 2008 by Bodleander et al. [10] has significantly increased the interest in kernelization. We know introduce the schema. The following definitions and corollary are taken from Parameterized Complexity [3] book.

Definition 5.2. An equivalence relation $\mathcal{R}$ on Σ^* is called a *polynomial equivalence relation* if the following conditions hold:

1. There exists an algorithm $\mathcal{A}$ such that given given $x, y \in \Sigma^*$ decides whether $x \equiv_{\mathcal{R}} y$ in time $p(|x| + |y|)$ for a polynomial p..
2. Relation $\mathcal{R}$ restricted to the set $\Sigma^{\leq n}$ has at most polynomially many equivalence classes.

Definition 5.3. Let $L \subseteq \Sigma^*$ be a language, $\mathcal{R}$ be an equivalence relation $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. A *cross-composition* of a language L into Q is an algorithm $\mathcal{A}$ that given an input $x_1, ..., x_t \in L$, equivalent with respect to $\mathcal{R}$, outputs an instance $(x, k') \in \Sigma^* \times \mathbb{N}$ such that:

1. $k \leq p(\max\limits_{1 \leq i \leq t} |x_i| + log(t))$ for a polynomial p.
2. $(x, k') \in Q$ if and only if there exists an index i such that $x_i \in L$.

Corollary 5.1. *If an NP-hard language L cross-composes into the parameterized language Q, then Q does not admit a polynomial kernel unless NP$\subseteq$ coNP/poly.*

We prove nonexistence of a polynomial kernel by a cross-composition from Spanning Star Forest Extension into itself. Observe that Spanning Star Forest Extension is NP-complete by Theorem 1.6. In the proof, we use an *instance selector*, a pattern commonly applied to solve a composition. Intuitively, we need to come up with a gadget that satisfies all instances but one. Therefore, we require that at least one of the packed instances has a solution.

Lemma 5.6. *There exists a cross-composition from* Spanning Star Forest Extension *into itself, parameterized by the number of forced edges.*

Proof. Firstly, we define a relation $\mathcal{R}$. Assume that all malformed graphs are considered as equivalent. Moreover, we say that $(G_1, F_1) \equiv_{\mathcal{R}} (G_2, F_2)$ if and only if graphs induces by forced vertices are isomorphic. Observe that the relation implies that the graphs have the same amount of forced edges.

Now, let $(G_1, F_1), \cdots, (G_t, F_t)$ be the input instances. We normalize them by applying Lemma 5.1. If all of the instances are NO-instances, we return an isolated vertex. Otherwise, let $(H_0, F_0', \cdot, H_{p-1}, F_{p-1}')$ be the normalized instances. Because we operate on binary representation of indices, we duplicated some instances so that $p = 2^s$ for some integer $s > 0$. Observe that the step at most doubles the number of instances.

We now proceed to the construction of the output instance. Firstly, for every input instance we label forced edges with $f_1, ..., f_n$. Then, let (G, F) be a sum of graphs $H_0, ..., H_{p-1}$ such that the forced edges with the same labels are unified. Thus, we have that $|F| = n$. Now, we introduce $2s$ forced vertices v_α^β where $\beta \in \{0, 1\}$ and $0 \leq \alpha \leq s-1$, and s forced edges $v_\alpha^0 v_\alpha^1$. In addition, for every forced edge $v_\alpha^0 v_\alpha^1$ we introduce a free vertex v_α and edges $v_\alpha^0 v_\alpha, v_\alpha^1 v_\alpha$. Observe that now $|F| \leq n + \log(t)$. Recall that by U_i we denote a set of free vertices of the graph G_i. For every index i, where $0 \leq i \leq p-1$, we do the following: let $i = b_0 b_1 ... b_{s-1}$ be a bit representation. If necessary, we add leading zeros so that every value is represented by s bits. For every vertex $v \in U_i$ we introduce a set of edges $\{vv_\alpha^{1-b_\alpha} : 0 \leq \alpha \leq s-1\}$.

We output the modified instance (G, F). Observe that (G, F) is also a normalized instance. The first condition of the cross-composition is satisfied as $|F| \leq n + \log(p)$. So, to finish the proof, we need to show that one of the instances (G_i, F_i) is a YES-instance if and only if (G, F) is a YES-instance.

For the forward implication, let S be a spanning star forest for (G_i, F_i). Let $C_S = N_S(U_i)$ be a set of centers in S. We construct a solution for (G, F) as follows. Let $i = b_0 ... b_{s-1}$ be a bit representation with leading zeros if necessary. Let $C_G = \{v_\alpha^{b_\alpha} : 0 \leq \alpha \leq s-1\}$. By the definition of G, we get that $N(C_S \cup C_G) = U$. Thus, by Lemma 5.4, (G, F) has a spanning star forest.

Conversely, let S be a solution for (G, F). We define $C_G = N(\{v_\alpha : 0 \leq \alpha \leq s-1\})$. Superscripts of vertices $v_0^{\alpha_0}, ..., v_{s-1}^{\alpha_{s-1}}$ create a value i. Now, observe that $N(U_i) \subseteq V(F)$ because the superscripts of vertices $v_j^{\alpha_j}$, for $0 \leq j \leq s-1$, match bit representation of i. Thus, (G_i, F_i) is a yes instance. □

Theorem 1.9. Spanning Star Forest Extension *parameterized by the number of forced edges does not admit a polynomial kernel unless NP ⊆ coNP/poly.*

Proof. By Corollary 5.1 and Lemma 5.6 we obtain that Spanning Star Forest Extension parameterized by $|F|$ does not admit a polynomial size kernel unless NP ⊆ coNP/poly. □

5.4. Parametrization by the number of free vertices

In this section, we parameterize Spanning Star Forest Extension by the number of free edges. We begin with the algorithm that solves the problem in time ??.Then, unlike in the first variant, we present a kernelization algorithm. We show, that the reduced instance has at most $3k$ vertices where k is the parameter.

5.4.1. Algorithm

We start with stating a theorem for the CNF-SAT problem parameterized by the number of clauses proven by AUTHORS.

Theorem 5.2. *TODO*

Now, observe that CNF-SAT parameterized by the number of clauses and Spanning Star Forest Extension parameterized by the number of free vertices are interreducible with respect to polynomial reductions. Thus, we are ready to show an algorithm.

Theorem 1.7. Spanning Star Forest Extension *parameterized by the number of forced edges can be solved in time $\mathcal{O}^*(2^{|F|})$ where $|F|$is the number of forced edges.*

Proof. Let (G, F) be an arbitrary instance of Spanning Star Forest Extension parameterized by the number of free vertices. Firstly, we apply Lemma 5.1. If the lemma yields that the instance is a NO-instance, then we conclude. Otherwise, let (G', F') be the output graph of normalization lemma. We apply Lemma 5.3 and obtain a formula ϕ. Observe that ϕ has at most $|F|$ variables and at most $|U|$ clauses. Now, by Theorem 5.2 there exists an algorithm that decides whether ϕ is satisfiable or not. So, we apply it and obtain the answer for the instance (G, F). □

5.4.2. Kernelization

At first we introduce a definition of a crown decomposition proposed by Chor, et al. [6]. Recall that for a disjoint sets $X, Y \subseteq V(G)$, a *matching of X into Y* is a matching M such that every edge has one endpoint in X and one endpoint in Y. In addition, for every $x \in X$ there exists exactly one edge $xy \in M$ such that $y \in Y$.

Definition 5.4. A *crown decomposition* of a graph G is a partitioning of $V(G)$ into three parts C, H and R, such that:

- C is nonempty.
- C is an independent set.
- There are no edges between C and H. That is, H separates C from R.
- Let E' be the set of edges between C and H. Then, E' contains a matching of size $|H|$. In other words, G contains a matching H into C.

Without digging into details, we show the results of Kőnig [7], Hall [8] and an algorithm invented by Hopcroft and Karp [9] that are used further in this subsection.

Theorem 5.3 (**Kőnig's theorem**). *In every undirected bipartite graph the size of a maximum matching is equal to the size of a minimum vertex cover.*

Theorem 5.4 (**Hall's theorem**)**.** *Let G be an undirected bipartite graph with bipartition (V_1, V_2). The graph G has a matching of V_1 into V_2 if and only if for all $X \subseteq V_1$, we have $|N(X)| \geq |X|$.*

Theorem 5.5 (**Hopcroft-Karp algorithm**)**.** *Let G be an undirected Let G be an undirected bipartite graph with bipartition (V_1, V_2), on n vertices and m edges. Then, we can find a maximum matching as well as a minimum vertex cover of G in time $\mathcal{O}(m\sqrt{n})$. Furthermore, in time $\mathcal{O}(m\sqrt{n})$ either we can find a matching of V_1 into V_2 or an inclusion-wise minimal set $X \subseteq V_1$ such that $|N(X)| < |X|$.*

Now, we present the lemma proposed by Cygan, et al. [3] that is a fundamental concept for designing kernelization algorithms.

Lemma 5.7. *Let G be a graph without isolated vertices with at least $3k + 1$ vertices. There is a polynomial-time algorithm that either*

- *finds a matching of size $k + 1$ in G; or*
- *finds a crown decomposition of G.*

Finally, equipped with the above information, we are ready to show a linear kernel for Spanning Star Forest Extension parameterized by the number of free vertices.

Lemma 5.8. *There exists a polynomial-time algorithm, that, given an instance (G, F) with a parameter $|U|$, either*

- *answers whether (G, F) has a spanning star forest or*
- *outputs a subgraph of (G, F) such that it has at most $|U|$ forced edges.*

Proof. Firstly, observe a trivial case. If $|F| \leq |U|$, then (G, F) is the reduced instance. Otherwise, we apply Lemma 5.1. We either conclude that (G, F) is a NO-instance, which satisfies the first bullet of the stated lemma, or obtain a normalized graph (G', F'). Observe that an isolated edge is a star and it cannot be expanded. Thus, we can remove them as well. In further, we consider only the case where $|F'| \geq |U'|$. If $|F'| < |U'|$, then (G', F') is the reduced instance.

Let $G_{F',U'}$ be a bipartite graph formed from (G', F') by contracting forced edges. We apply Hopcroft-Karp's algorithm to $G_{F',U'}$. We either obtain a matching M of F' into U' or an inclusion-wise minimal set $X \subseteq F'$ such that $|N(X)| < |X|$.

Consider the first case in which Hoprocft-Karp's algorithm returns a matching. Then, we conclude that (G, F) is a YES-instance due to the following claim:

Claim 5.5. *Assume (G, F) is a normalized instance of* Spanning Star Forest Extension *where $|F| \leq |U|$. If Hoprocft-Karp's algorithm ran on $G_{F,U}$ returns a matching M of F into U, then (G, F) is a YES-instance.*

Proof. Since M is a matching of F into U, then we infer that $|F| \geq |U|$. Furthermore, we assumed that $|F| \leq |U|$. Thus, $|F| = |U|$ and M is also a matching of U into F. Now, we claim that there exists a spanning star forest for (G, F). Indeed, let $M' \subseteq E(G')$ be a set of edges of minimum size such that M' corresponds to M. Then, $M' \cup F$ is a spanning star forest for G. □

Observe that the normalized instance (G', F') satisfies the assumptions from the above claim.

Now, consider the second case. Let $X \subseteq$ be an inclusion-wise minimal set such that $|N(X)| < |X|$. Let $C = X$, $H = N(X)$ and $R = V(G_{F',U'}) \setminus (H \cup C)$. Now, we claim that:

Claim 5.6. *(C, H, R) is a crown decomposition of $G_{F',U'}$.*

Proof. C is nonempty and C is an independent set as F' is an induced matching in G'. Moreover, H separates C from R because H is a set of all the neighbors of C. Now, we prove that there exists a matching of H into C. Select an arbitrary $v \in C$. There is a matching of $C \setminus \{v\}$ into H since $|N(C')| \geq |C|$ for every $C' \subseteq C$. Since $|C| > |H|$, we have that the matching of $C \setminus \{v\}$ into H is actually a matching of H into C. Therefore, (C, H, R) is a crown decomposition of $G_{F',U'}$. □

□

5.5. Parametrization by treewidth

In this section, we parameterize the problem by the treewidth of the input graph. As an input, we receive a tuple $(G, F, \mathcal{T})$ where $\mathcal{T}$ is a tree decomposition of G. We present a dynamic programming algorithm that works on a given tree decomposition. Later, we prove a lower bound of a running time based on a reduction from the CNF-SAT problem. Note that we only consider G as a normalized instance.

5.5.1. Preliminaries

We extend the notation of tree decomposition. Namely, for introduce vertex node, we distinguish *introduce free vertex node* and *introduce forced vertex node*. Similarly, We extend the definition for introduce edge and forget vertex node consequently. Every entry of a dynamic table dp has three parameters: a tree decomposition node t and two assignment functions f, g. An *assignment function for forced vertices* $f : (X_t \cap V(F)) \to \{\text{True}, \text{False}\}$ is a mapping that distinguishes two states. If $f(v) = 1$, then we say that v is a center whereas, if $f(v) = 0$, then v is either a candidate or a ray. An *assignment function for free vertices* $g : (X_t \cap U) \to \{\text{True}, \text{False}\}$ is a mapping that indicates whether a free vertex has been0 added to a star or not. Hence, for a free vertex v we say that v is in a star if $f(v) = 1$ and, if it is not, then $f(v) = 0$.

For the sake of clarity, we introduce the following notation. Suppose $(G, F, \mathcal{T})$ is an input instance. Let $(t, X_t) \in \mathcal{T}$ be a node and the corresponding set of vertices of a tree decomposition. By G_t we define a subgraph of G such that $V(G_t) = X_t$ and G_t has edges that have been introduced in a subtree rooted in t. Let $\mathcal{F}_t = \{(X_t \cap V(F)) \to \{\text{False}, \text{True}\}\}$ be a family of all assignment functions for forced vertices from X_t and let $\mathcal{G}_t = \{(X_t \setminus V(F)) \to \{\text{False}, \text{True}\}\}$ be a family of all assignment functions for free vertices from X_t. For any assignment function h and $X' \subseteq X$, where X is the domain of h, we use $h_{|X'}$ to denote the restriction of h to X'. For a subset $X \subseteq V(G)$ consider an assignment function h. For a vertex $v \in V(G)$ and a logic value $p \in \{\text{True}, \text{False}\}$ we define a new assignment $h_{v \to p} : X \cup \{v\} \to \{\text{True}, \text{False}\}$ as follows:

$$f_{v \to p} = \begin{cases} f(u), & \text{if } u \neq v \\ p, & \text{if } u = v \end{cases}$$

5.5.2. Algorithm

We provide formulas for every type of a node. To call an entry from a dynamic table we provide three arguments1. The first one is a node t from a tree decomposition. The second one is a forced vertices assignment $f \in \mathcal{F}_t$. The last one is a free vertices assignment $g \in \mathcal{G}_t$. By $\mathrm{dp}[t, f, g]$ the answer whether in G_t there exists an independent set $C \subseteq V(F)$ such that

- $C \cap X_t = f^{-1}(\mathrm{True})$, and
- $g^{-1}(\mathrm{True}) \subseteq N_{G_t}(C)$

Observe that the existence of an independent set implies the existence of a spanning star forest based on Lemma 5.4.

Leaf node For a leaf node t we have that $X_t = \emptyset$. An empty graph is a spanning star forest. Hence:

$$\mathrm{dp}[t, \emptyset, \emptyset] = \mathrm{True}$$

Introduce forced vertex node Let t be an introduce node with a child t' such that $X_t = X_{t'} \cup \{v\}$ and $v \in V(F)$. Observe that v is isolated in G_t. Thus, we pass the value from the child node:

$$\mathrm{dp}[t, f, g] = \mathrm{dp}[t', f_{|X_{t'}}, g]$$

Introduce free vertex node Let t be an introduce free vertex node with a child t' such that $X_t = X_{t'} \cup \{v\}$. Note that $\mathrm{dp}[t, f, g] = \mathrm{False}$ if $g(v) = \mathrm{True}$ as v is isolated in G_t. Thus, we obtain the following:

$$\mathrm{dp}[t, f, g] = \begin{cases} \mathrm{dp}[t', f, g_{|X_{t'}}], & \text{if } g(v) = \mathrm{False} \\ \mathrm{False}, & \text{otherwise} \end{cases}$$

Introduce free edge node Let t be an introduce free edge node labeled with an edge $vu \in E(G) \setminus F$ and let t' be the child of t. Recall that G is a normalized instance. Without loss of generality, assume that $v \in U$ and $u \in V(F)$ as every free edge has one end in a free vertex and the other one in a forced vertex. Let $f \in \mathcal{F}_t$ and $g \in \mathcal{G}_t$. Suppose that $g(v) = \mathrm{False}$. Then, we simply pass the value from the child's node. Otherwise, $g(v) = \mathrm{True}$. We need to find a vertex $u' \in V(F) \cap X_t$ such that $f(u') = \mathrm{True}$ and v is a neighbor of u' in G_t. It can be u if $f(u) = \mathrm{True}$ or some other forced vertex for which we have already introduced the edge vu'. Thus, we obtain the following equations:

$$\mathrm{dp}[t, f, g_{v \to \mathrm{True}}] = \begin{cases} \mathrm{dp}[t', f, g_{v \to \mathrm{True}}] \vee \mathrm{dp}[t', f, g_{v \to \mathrm{False}}], & \text{if } f(u) \\ \mathrm{dp}[t', f, g_{v \to \mathrm{True}}], & \text{otherwise} \end{cases}$$
$$\mathrm{dp}[t, f, g_{v \to \mathrm{False}}] = \mathrm{dp}[t', f, g_{v \to \mathrm{False}}]$$

Introduce forced edge node Let t be an introduce free edge node labeled with an edge $vu \in F$, t' be the child of t and let $f \in \mathcal{F}_t$. Calculations for t are simple. We set to False all the elements of dynamic table for which $f(v) = f(u) = \text{True}$:

$$\mathrm{dp}[t, f, g] = \begin{cases} \mathrm{dp}[t', f, g], & \text{if } f(v) = \text{False or } f(u) = \text{False} \\ \text{False}, & \text{otherwise} \end{cases}$$

Forget forced vertex node Let t be an forget forced vertex node with a child t', such that $X_t = X_{t'} \setminus \{v\}$ and let $u \in V(F)$ such that $vu \in F$. Observe that for any $f \in \mathcal{F}_t$ that satisfies $f(v) = f(u) = \text{True}$, $\mathrm{dp}[t', f] = \text{False}$ because we have already changed their values during introduce forced edge node. Thus, the formula looks as follows:

$$\mathrm{dp}[t, f, g] = \mathrm{dp}[t', f_{v \to \text{True}}, g] \vee \mathrm{dp}[t', f_{v \to \text{False}}, g]$$

Forget free vertex node Let t be a forget free vertex node with a child t' such that $X_t = X_{t'} \setminus \{v\}$ where $v \in U$. We can pass the value from a child node if and only if v was added to a star, that is, for a $g \in \mathcal{G}_{\sqcup}$, $g(v) = \text{True}$. Consequently, we obtain:

$$\mathrm{dp}[t, f, g] = \mathrm{dp}[t', f, g_{v \to \text{True}}]$$

Join node Let t be a join node with children t_1 and t_2. Recall that $X_t = X_{t_1} = X_{t_2}$. We say that assignment functions f_1, g_1 of X_{t_1} and f_2, g_2 of X_{t_2} *match* with assignments f, g of X_t if the following conditions hold:

1. For every forced vertex $v \in X_t \cap V(F)$, $f(v) = f_1(v) = f_2(v)$.
2. For every free vertex $v \in X_t \cap U$, $g(v) = g_1(v) \vee g_2(u)$,

Hence, we get:

$$\mathrm{dp}[t, f, g] = \bigvee_{g_1, g_2 \text{ match } g} dp[t_1, f, g_1] \wedge dp[t_2, f, g_2]$$

5.5.3. Complexity analysis

Now we proceed to the complexity analysis. Observe that introduction and forget nodes requires constant number of operations. We either copy a specific value from a child's node or perform a logical OR operation of two boolean values.

Computing a join node is the bottleneck in this algorithm. A naive approach to obtain the value for a join node cell would result in time $\mathcal{O}^*(4^{\mathrm{tw}(G)})$. Thus, we could conclude with an algorithm solving Spanning Star Forest Extension parameterized by treewidth in time $\mathcal{O}^*(8^{\mathrm{tw}(G)})$ as there are $O(|G| \cdot 2^{\mathrm{tw}(G)})$ array entries. However, we can improve the running time, using fast computation of the *cover product*.

Definition 5.5. The *cover product* of two functions $f, g : 2^V \to \mathbb{Z}$ is a function $(f *_c g) : 2^V \to \mathbb{Z}$ such that for every $Y \subseteq V$:

$$(f *_c g)(Y) = \sum_{A \cup B = Y} f(A)g(B)$$

Now, we state the theorem proved by Björklund, et al. [12]:

Theorem 5.6. *For two functions $f, g : 2^V \to \mathbb{Z}$, given all 2^n values of f and g in the input, all the 2^n values of the cover product $f *_c g$ can be computed in $\mathcal{O}(2^n \cdot n)$ arithmetic operations.*

Note that the disjunction in every join node is nothing else than a cover product for a fixed forced vertices assignment. Thus, we formulate the lemma:

Lemma 5.9. *Given a join node t, one can calculate all values of $dp[t, f, g]$, for $f \in \mathcal{F}_t$ and $g \in \mathcal{G}_t$, in time $O(2^{tw(G)})$.*

Proof. Fix $f \in \mathcal{F}_t$. We define a function $c_{t,f} : 2^{X_t \cap U} \to \mathbb{Z}$ as follows:

$$c_{t,f}(X) = dp[t, f, g], \text{ such that } g^{-1}(1) = X.$$

Now, for $X \subseteq X_t \cap U$, observe that:

$$\begin{aligned}(c_{t_1,f} *_c c_{t_2,f})(X) &= \sum_{A \cup B = X} c_{t_1,f}(A) c_{t_2,f}(B) \\ &= \sum_{g_1^{-1}(1) \,\cup\, g_2^{-1}(1) = X} dp[t_1, f, g_1] dp[t_2, f, g_2]\end{aligned}$$

which exactly reflects the computation that we perform during a join node. Thus, $\mathrm{dp}[t, f, g] = (c_{t_1,f} *_c c_{t_2,f})(g^{-1}(1)) > 0$.

Observe that $|\mathcal{F}_t| = 2^{|X_t \cap V(F)|}$. By Theorem 5.6, for a fixed $f \in \mathcal{F}_t$ and every $g \in \mathcal{G}_t$ we can calculate values of $\mathrm{dp}[t, f, g]$ in time $2^{|X_t \cap U|}$. Thus, we can fill all the entries corresponding to the node t in time $\mathcal{O}^*(2^{|X_t \cap V(F)|} \cdot 2^{|X_t \cap U|}) = \mathcal{O}^*(2^{|X_t|}) \leq \mathcal{O}^*(2^{\mathrm{tw}(G)})$. □

Theorem 1.12. Spanning Star Forest Extension *parameterized by treewidth can be solved in time $2^t \cdot poly(t) \cdot n$.*

Proof. Consider the algorithm described in the previous subsection. To calculate a single entry for introduce and forget nodes, we need constant time. By Lemma 5.9, we showed that that the values for a join node can be calculated in $\mathcal{O}^*(2^{\mathrm{tw}(G)})$. There are polynomially many nodes in a tree decomposition. Thus, we can fill the entries of a dynamic table in time $\mathcal{O}^*(2^{\mathrm{tw}(G)})$ and provide the answer whether the input graph has a spanning star forest. □

We have just proven the complexity of the algorithm stated in the previous subsection. Now, we show a lower bound for the problem based on the CNF-SAT problem.

Theorem 1.13. *Unless* CNF-SAT*cannot be solved in time $\mathcal{O}^*((2-\epsilon')^n)$ for some $\epsilon' > 0$, there is no algorithm for* Spanning Star Forest Extension *parameterized by treewidth that achieves running time $\mathcal{O}^*((2-\epsilon)^t)$ for any $\epsilon > 0$, where* t *is the treewidth of the input graph.*

Proof. We provide a polynomial-time reduction that takes a CNF-SAT instance ϕ of n variables and m clauses, and constructs an instance (G, F) of width n. If there existed an algorithm solving Spanning Star Forest Extension parameterized by the treewidth of the input graph in $\mathcal{O}^*((2-\epsilon)^n)$, then we could compose the reduction with this algorithm and solve CNF-SAT in time $\mathcal{O}^*((2-\epsilon')^n)$.

We first give a brief overview of how the output graph looks like and provide a notation for referencing to the specific vertex. The output instance (G, F) consists of $n+1$ copies of the *formula graph*, denoted as $G_1, G_2, ..., G_{n+1}$ and n *formula gadgets* $H_G^1, H_G^2, ..., H_G^n$. Every formula graph G_i consists of m *clause graphs* $C_1^i, C_2^i, ..., C_m^i$. Every formula gadget H_G^i consists of m *clause gadgets* $H_1^i, H_2^i, ..., H_m^i$. By v_j^i we reference to vertices in j-th clause graphs from i-th formula graph and by g_j^i we reference to vertices in j-th clause gadget from i-th formula gadget.

After the brief introduction, we present a construction of a clause graph C_j^i. We introduce a free vertex $v_j^i[c]$ and n forced edges $v_j^i[x_k]v_j^i[\neg x_k]$. For every literal l occurring in the clause c_i, we introduce an edge $v_j^i[l]v_j^i[c]$. Observe that every clause graph consists of $2n+1$ vertices and n forced edges.

Now, we present a construction of a clause gadget H_j^i. Firstly, we introduce n free vertices $g_j^i[1], g_j^i[2], ..., g_j^i[n]$. If $j < m$, then for every $g_j^i[k]$ we introduce edges $v_j^i[\neg x_k]g_j^i[k]$ and $g_j^i[k]v_{j+1}^i[x_k]$. Otherwise, if $j = m$, then we introduce edges $v_m^i[\neg x_k]g_m^i[k]$ and $g_m^i[k]v_1^{i+1}[x_k]$. Observe that for every clause gadget we introduced n additional free vertices.

Intuitively, the graph consists of n paths for each variable. Every two consecutive forced edges are separated by two free edges. Then, we have $n+1$ copies of vertices corresponding to clauses. Each clause has its own set of n forced edges.

Let us discuss the specifications of the output graph. If every clause graph has $2n+1$ vertices and n forced edges while every clause gadget has n free vertices, we conclude that $|V(G)| = (n+1) \cdot m \cdot (2n+1) + n \cdot m \cdot n = \mathcal{O}(mn^2)$ and $|F| = (n+1) \cdot m \cdot n = \mathcal{O}(mn^2)$. Observe that (G, F) is normalized as there are no isolated vertices, every clause vertex and every gadget vertex is adjacent to forced vertices only and F is an induced matching. Note the following claim:

Claim 5.7. *The reduction outputs an instance (G, F) such that $tw(G) \leq n$.*

Proof. Observe that by simply sweeping the graph from C_1^1 to C_m^{n+1} we can show that $\mathrm{pw}(G) \leq n$. Hence, we conclude that $\mathrm{pw}(G) \leq \mathrm{tw}(G) \leq n$ because path decomposition is a specific case of a tree decomposition. □

Finally, we show that ϕ is satisfiable if and only if (G, F) has a spanning star forest. For the forward implication, let σ be an evaluation such that $\sigma(\phi) = \text{True}$. If $\sigma(\phi) = \text{True}$, then for every $C_i \in \text{Clauses}$ there exists a literal $l_i \in C_i$ such that $\sigma(l_i) = 1$. Now, let $L = \{v_j^i[l] : \sigma(l) = 1, 1 \leq i \leq n+1 \text{ and } 1 \leq j \leq m\}$. Clearly, $\{v_j^i[C_k] : C_k \in \text{Clauses}, 1 \leq i \leq n+1 \text{ and } 1 \leq j \leq m\} \subseteq N_G(L)$ because σ satisfies the formula ϕ. Moreover, for every $g_j^i[k]$, $g_j^i[k] \in N_G(L)$ as it is adjacent to vertices corresponding to both k-th literal and its negation. Moreover, observe that L is an independent set in G because for every forced edge $v_j^i[x_k]v_j^i[\neg x_k] \in F$ either $\sigma(x_i) = \text{False}$ or $\sigma(\neg x_i) = \text{False}$. Hence, by Lemma 5.4, there exists a spanning star forest for (G, F).

For the converse, let S be a spanning star forest. We denote by C a set of centers in S. Clearly, $C \subseteq V(F)$. Fix a constant k such that $1 \leq k \leq n$. By P_k we denote a path corresponding to k-th variable. Now, consider the set $K = P_k \cap C$. A *switch* is a vertex $g_j^i[k]$ such that $|N_G(k)| = 2$. In other words, the position of a center changes from a vertex corresponding to $\neg x_k$ to a vertex corresponding to x_k. Note the following:

Claim 5.8. *If S is the solution for (G, k), then for a path P_k there exists at most one switch in S.*

Proof. Suppose contrary that there are two switches. So, there is a switch from x_k to $\neg x_k$, say g_j^i. Then, $\deg_S(g_j^i) = 0$. Contradiction, S is not a spanning star forest. □

Since there are n paths, S can have at most n switches in total. Hence, there exists a copy of a graph formula G_i such that it does not have any switches between the clauses. We create the evaluation σ as follows:

$$\sigma(x_k) = \begin{cases} \text{True, if there exists } j,\ 1 \leq j \leq m \text{ such that } \deg_S(v_j^i[x_k]) > 1 \\ \text{False, otherwise} \end{cases}$$

We claim that the evaluation satisfies ϕ. Firstly, note that in G_i there are no switches. Fix an arbitrary clause C_j. Now, observe that there exists a literal $l_k \in C_j$ such that $v_j^i[l_k]v_j^i[c] \in S$. Then, $\deg_S(l_k) > 1$. By the definition of σ, $\sigma(l_k) = \text{True}$, and therefore $\sigma(C_j) = \text{True}$. We conclude that $\sigma(\phi) = \text{True}$ as we proved that an arbitrary clause in the formula is satisfied. $\square$

Bibliography

[1] Downey, R., Fellows, M.: *Parameterized Complexity.* Springer-Verlag, New York (1999).

[2] Flum, J., Grohe, M.: *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin (2006).

[3] Cygan, M., Fomin, F., Kowalik L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: *Parameterized Algorithms.* Springer, (2015).

[4] Pătraşcu, M., Williams, R.: *On the Possibility of Faster SAT Algorithms.* Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1065-1075. SIAM (2010).

[5] Eisenbrand F., Grandoni F.: *On the Complexity of Fixed Parameter Clique and Dominating Set.* Theor. Comput. Sci. 326, pp. 57–67, (2004).

[6] Chor, B., Fellows, M, Juedes, D.: *Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps.* WG'04 Proceedings of the 30th international conference on Graph-Theoretic Concepts in Computer Science, pp. 257-269, (2004).

[7] König, D.: *Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre.* Math. Ann. 77(4), pp. 453-465, (1916)

[8] Hall, P.: *On representatives of subsets*, J. London Math. Soc. 10, pp. 26-30, (1935).

[9] Hopcroft, J.E., Karp, R.M.: *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs.* SIAM J. Computing 2, pp. 225-231, (1973).

[10] Bodlaender, H., Downey, R., Fellows, M., Hermelin, D.: *On problems without polynomial kernels.* Proceedings of 35th International Colloquium on Automata, Languages and Programming, pp. 563–574, (2008).

[11] Fortnow, L., Santhanam, R.: *Infeasibility of instance compression and succinct pcps for NP.* Journal of Computer and System Sciences, 77(1), pp. 91–106, (2011).

[12] Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: *Fourier meets Möbius: fast subset convolution.* Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC), pp. 67-74. ACM, New York (2007).