University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Adam Starak
Student no. 361021

Application of parameterized techniques to Spanning Star Forest and its variants

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

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

Tytuł po polsku

Contents

Chapter 1

Introduction

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.*

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

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

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 $\epsilon > 0, k \geq 3$ 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.*

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

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.*

Theorem 1.9. Spanning Star Forest Extension *parameterized by the number of forced edges does not admit a polynomial kernel.*

Theorem 1.10. Spanning Star Forest Extension *parameterized by the number of free vertices admits a linear kernel.*

Theorem 1.11. Spanning Star Forest Extension *parameterized by the number of free vertices can be solved in ??.*

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 SETH fails, there is no algorithm for* Spanning Star Forest Extension *parameterized by treewidth that achieves running time $\mathcal{O}^*((2-\epsilon)^{tw})$ for any $\epsilon > 0$, where* tw *is treewidth of the input graph.*

Chapter 2

Preliminaries

2.1. Structures

In a simple graph G we denote by $V(G)$ and $E(G)$ the set of vertices and edges respectively. Let $\deg_G(v)$ denote degree of the vertex v in the graph G which is the number of adjacent vertices. Let $G \setminus v$ be the abbreviation for $G' = (V(G) \setminus \{v\}, E(G) \setminus \{(u,v) : u \in V(G)\})$. 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 $X \subseteq V(G)$ we define by $G[X]$ the graph induced by vertices from X. By $G' \subseteq G$ we denote a relation between graphs such that $V(G') \subseteq V(G)$ 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 size at least 2 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. We say that $I \subseteq V(G)$ is an independent set in G if and only if it holds that for all $u, v \in I$ $uv \notin E(G)$.

2.2. Parameterized complexity

Parameterized complexity is a young branch of computational complexity theory. The main outcomes of researches can be found in publications such as: Parameterized Complexity [1], Parameterized Complexity Theory [2] or in the most recent book Parameterized Algorithms [3].

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

See the example problems below to understand the concept:

Definition 2.2. Independent Set: Given a graph G and a positive integer k, find 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, find a set D such that $|D| \leq k$ and every vertex from the graph 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 size of a dominating set or by the treewidth.

Now, we want to introduce different complexity classes. The first one is called FPT (fixed parameter tractable). We say that a 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 an XP class. 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 to polynomial reduction, we now introduce a *parameterized reduction*, that is, a notion of transforming instances of a certain parameterized problem to 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 $f(k) \cdot |x^{\mathcal{O}(1)}|$ for some computable function f.

Finally, we introduce the last family of complexity classes. *W-hierarchy* is an ascending chain of classes that follows: W[1] $\subseteq$ W[2]... . For the purpose of this paper, we define W[1] as a closure of Independent Set problem and W[2] as a closure of Dominating Set problem. In other words, Independent Set is a W[1]-complete and Dominating Set is a W[2]-complete when parameterized by the size of the independent or dominating set respectively. It is not known what is the correlation between FPT and W[1] classes. In this paper, we assume that W[1] $\neq$ FPT.

Last but not least, we introduce a *kernelization algorithm* - a way of reducing the size of input instances:

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 $(v, u) \in E(G)$ there exists a bag 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 T.

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

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

- $X_i = \emptyset$ if i is either root or 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 $X_t = X_{t_1} = X_{t_2}$.

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

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 an instance is a NO-instance.

3.1. Decision variant

In decision variant of Spanning Star Forest, all that we have to do is to answer whether a graph is a YES-instance or a NO-instance. As it turns out, every graph that does not contain any isolated vertex has a spanning star forest. See the lemma below:

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 inductive 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 correct spanning star forest. Now, suppose that there exists a vertex u such that $\deg_G(u) > 1$. Let $C \subseteq G$ be a 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 for the graph G.

□

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

Corollary 3.1. *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. □

3.2. Constructing a solution

In this section we focus on obtaining an arbitrary solution for a given instance of Spanning Star Forest. We propose an algorithm, that for a connected graph outputs a spanning star forest in linear time. 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 $S_1, S_2, ..., S_n$ its 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 $|V(G)| \geq 2$
Output: spanning star forest of G
spanned $\leftarrow$ Array[$|V(G)|$];
$T \leftarrow$ SpanningTree(G);
$S \leftarrow \emptyset$;
for *v: postorder(T) and v is not a root* **do**
  if !*spanned*[v] **then**
    $u \leftarrow$ parent(T, v);
    $S \leftarrow S \cup \{uv\}$;
    spanned[v] $= True$;
    spanned[u] $= True$;
  end
end
$v \leftarrow root(T)$;
if !*spanned*[v] **then**
  $S \leftarrow S \cup \{uv\}$ where $v =$ parent(T, u);
end
return S;

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

Firstly, the algorithm creates a spanning tree T. 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 connected to its parent. If the root has not been added to the solution during the for loop, we add an arbitrary edge, incident to the solution, which finishes the algorithm.

Now we need to check if 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 appears in 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 following 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 consist of 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, 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 must contain a vertex v, its parent u and its grandparent w. From the Claim 3.3 we infer that w is the root. Moreover, u is the only neighbor of w in S. 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 the Theorem 1.1.

Proof (of Theorem 1.1). Given a graph G we run Algorithm 1 on its every connected component. Then, we merge partial results 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. □

Obtaining a spanning star forest without any limitations is simple. Both decision and constructive variants 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 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 $x \in Dom(\mu)$.

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 solution of minimal 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. □

In addition, we show one reduction rule, to exclude 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. Then, (G, k) is a YES-instance if and only if $(G \setminus I, k - |I|)$ is a YES-instance.*

Observe that the reduction can be simply proven as 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 $(G,', k')$ 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 a 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 correct solution for the instance (G', k') of Minimal Spanning Star Forest. 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.

□

The problems look so similar that one could ask whether there exists a reverse parameterized reduction. Indeed, it 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. 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 a 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). 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. Obviously, 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 next theorem:

Proof (of Theorem 1.2). Recall, that Dominating Set is a W[2]-complete problem. Note that by Lemma 4.2 and Lemma 4.3 the problems are equivalent. 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 reductions. Hence, we get:

Proof (of Theorem 1.3). Dominating Set is an NP-complete problem. As observed in the previous proof, Dominating Set and Minimal Spanning Star Forest are equivalent. Therefore, we conclude that Minimal Spanning Star Forest is also NP-complete. □

Interreducibility is a useful tool. Especially, if only one of the problems has been deeply studied in the past. As an example, we show how to transfer a lower bound for run 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:

Proof (of Theorem 1.4). 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 $\mathcal{O}(N^{k+o_k(1)})$ that concludes the proof. □

Now, consider the following theorem proven in *On the Possibility of Faster SAT Algorithms* [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* $\epsilon > 0, k \geq 3$ *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:

Proof (of Theorem 1.5). Firstly, we show that the lower bound is optimal. 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 $k \geq 3$ is the parameter. Let (G, k) be an instance of Dominating Set. We show an algorithm that contradicts Theorem 4.1. At first, 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. 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 this chapter, we use three different parameters: the number of forced edges, the number of free vertices and the treewidth.

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.

5.1. Preliminaries

5.1.1. Instance normalization

Notice that this time we do not have any limit on the number of connected components. The hardness of the problem lies in choosing which of the forced candidates should become a forced center and which one should become a forced ray. Also, observe that a star is a primitive structure. The star's maximal radius is equal to 2. It means that we can look at the problem rather locally than globally. Thus, it is possible to normalize instances i.e. remove a part of free vertices and free edges.

We propose a set of three conditions that every normalized instance should satisfy. Note that an *induced matching* M in a graph G is a set of disjoint edges such that there are no additional edges between the vertices of M in G.

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. $\forall_{u \in U}\ N_G(u) \subseteq V(F)$.

Surprisingly, almost all of the instances of Spanning Star Forest Extension can be normalized. Otherwise, we conclude that an instance is a NO-instance. See the following lemma:

Lemma 5.1. *Let (G, F) be an instance of* Spanning Star Forest Extension. *There exists a set of reduction rules such that, after exhaustive application, concludes that (G, F)*

is a NO-instance or outputs in polynomial time an equivalent normalized instance (G', F') *satisfying* $G' \subseteq G$ *and* $F' \subseteq F$.

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 it is a NO-instance.

Obviously, an isolated vertex cannot be a star. 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 with NO-instance if F forms a forbidden subgraph.

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

Now, let us show three rules. After exhaustive application, they return a set of forced edges that is 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 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 is 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 of size greater than 2, 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'). Observe that $F = F' \cup \mathrm{Rays}$. By the definition, $N_G(\mathrm{V(Rays)}) = F_C$. Thus, $S \cup \mathrm{Rays}$ is a spanning star forest for (G, F). Conversely, let S be a solution for (G, F). Obviously, S restricted to the vertices of G' is a spanning star forest since every forced star of size greater than 2 was reduced to a forced star of size 2.

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

Claim 5.1. *Given an instance* (G, F)*, if Reduction 2 does 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.*

Proof. We prove the claim by contradiction. Firstly, suppose that F is not a matching. Hence, there exists a connected component of size greater than 2. 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. □

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 $V_P = \{u : uv \in E(G) \text{ and } u, v \in U\}$ 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. Every vertex has at least one neighbor that is also a free vertex. Hence, by Lemma 3.1 G_P has a spanning star forest. □

Observe that during partitioning 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 forms an edge in 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 $v \in V(G_{NP}) \setminus V(S')$. v is a free vertex because all the forced vertices are spanned by forced edges from F. However, v has no neighbors outside G_{NP}. Hence, $v \notin V(S)$ and S is not a spanning star forest for G. Contradiction. □

Recall that free vertices of G_{NP} do not have edges to another free vertices. Thus, the graph 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') that is a normalized instance and follows $G' \subseteq G$ and $F' \subseteq F$.

□

Ultimately, 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. *(G, F) has a spanning star forest 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 $C = N_S(U)$. 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 has edges to free vertices in S. Thus, C is an independent set.

For the backward implication, observe that F is an induced matching. Thus, every forced edge is a single star. If $U \subseteq N[C]$ it means that 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. Now, 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. Observe that in S, for every $uv \in F$, the degree of at most one vertex is greater than 1. Thus, S is spanning star forest for (G, F). □

5.2. NP-completeness

Let (G, F) be a normalized instance of Spanning Star Forest Extension after normalization. Such a representation substantially simplifies further study. Indeed, we can show a reduction from CNF-SAT.

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

Proof. Firstly, we show a reduction. Suppose ϕ is an arbitrary instance of CNF-SAT. ϕ is a CNF formula, so let Clauses $= \{C_1, ..., C_m\}$ be the set of clauses and let Variables $= \{x_1, ..., x_n\}$ be the set of variables from 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 $v[x_i]v[\neg x_i]$. Observe that the graph consists of $2n + m$ vertices and n forced edges. 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 if and only if there exists a satisfiable evaluation of the formula ϕ.

Before we begin the proof, observe that (G, F) is a normalized instance. Firstly, there are no isolated vertices because every clause has at least one literal and variables corresponds to a forced edge. Secondly, F is an induced matching because 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).

For the forward implication, let S be a solution for (G, F). We create an 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 $v[l_j]v[C_i] \in E(S)$. Then, $\deg_S(l_j) > 1$. By the definition of σ, $\sigma(l_j) =$ True, and therefore $\sigma(C_i) =$ True. We conclude that $\sigma(\phi) =$ 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 $\sigma(\phi) =$ True, then for every $C_i \in$ Clauses, $\sigma(C_i) =$ True. Moreover, for every $C_i \in$ 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 $\sigma(x_i) = 1$ or $\sigma(\neg x_i) = 1$. 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, it 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 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 $F' \subseteq F$. Now, we proceed to a formula construction. 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. Moreover, every clause C_w consists 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. □

Finally, we can prove the main theorem of the section:

Proof (of Theorem 1.6). We apply the reduction described in Lemma 5.2 from an NP-complete problem, that is, CNF-SAT. □

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 based on SETH

Previously in this chapter, we were proving various properties of Spanning Star Forest Extension problem. 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:

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:

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.

Proof (of Theorem 1.7). 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:

Proof (of Theorem 1.8). 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. Cross-composition

A *cross-composition* is a framework for proving kernelization lower bounds. A technique, firstly introduced in 2008 by Bodleander et al. (ICALP 2008), has significantly increased the interest in kernelization. As a result, Bodleander, Jansen and Kratsch have published a straightforward schema to show a nonexistence of a kernel.

The following definitions and corollary are taken from *Parameterized Complexity* 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. $k \leq p(\max_{1 \leq i \leq t} |x_i| + log(t))$ 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.*

5.3.3. 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 NP $\subseteq$ coNP/poly. To achieve this, we show two different approaches. Firstly, we show a proof based on nonexistence of a polynomial kernel for CNF-SAT parameterized by the number of variables. Then, we prove it by a cross-composition from Spanning Star Forest Extension into itself. The second proof is not feasible. 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.

We begin with showing interreducability of the following problems:

Lemma 5.5. Spanning Star Forest Extension *parameterized by the number of forced edges and* CNF-SAT *parameterized by the number of variables are interreducible.*

Proof. The reductions shown in Lemma 5.2 and Lemma 5.2 are the proof for interreducibility. In both cases, the number of variables or forced edges does not grow. □

Additionally,

Lemma 5.6. *There exists a cross-composition from* Spanning Star Forest Extension *into itself, parameterized by the number of forced edges. Therefore,* Spanning Star Forest Extension *parameterized by $|F|$ does not admit a polynomial size kernel unless* NP $\subseteq$ coNP/poly.

5.4. Parametrization by the number of free vertices

In this section, we parameterize the Spanning Star Forest Extension by the number of free edges. Unlike, the first variant, there exists a kernelization algorithm. We show, that the algorithm either outputs the answer or an instance of at most $3k$ vertices, where k is the parameter.

At first we introduce a definition of a crown decomposition proposed by Chor, Fellows and Juedes [5]. 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 [6], Hall [7] and an algorithm invented by Hopcroft and Karp [8] that are used further in this subsection.

Theorem 5.1 (**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 5.2 (**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 5.3 (**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 $|N(X)| < |X|$.*

5.5. Parametrization by treewidth

5.5.1. Preliminaries

For Spanning Star Forest Extension, we extend the notation of tree decomposition. Namely, for introduce vertex node, we distinguish *1*introduce free vertex node and *introduce forced vertex node*. We also extend the definition for introduce edge and forget vertex node consequently. Every cell 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 is 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 notations. For an assignment function f of X and $v \in X$, we use $f_{|v}$ to denote the restriction of f to $X \setminus \{v\}$. For a subset $X \subseteq V(G)$ consider an assignment function $f : X \to \{\text{True}, \text{False}\}$. For a vertex $v \in V(G)$ and a logic value $p \in \{\text{True}, \text{False}\}$ we define a new assignment $f_{v \to p} : X \cup \{v\} \to \{\text{True}, \text{False}\}$ 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 a cell from a dynamic table we provide three arguments. The first one is a node t from a tree decomposition. The second one is a forced vertices assignment $f : (X_t \cap V(F)) \to \{\text{True}, \text{False}\}$. The last one is a free vertices assignment $g : (X_t \cap U) \to \{\text{True}, \text{False}\}$.

Leaf node For a leaf node t we have that $X_t = \emptyset$. An empty graph is a correct 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 $v \in V(F)$. We simply assign an arbitrary value to the new vertex. Observe that we allow the case where for $vu \in F$, $f(v) = f(u) = \mathrm{True}$. However, we set the value for such a function to False during the introduce forced edge node. Thus, we get:

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

Introduce free vertex node Let t be an introduce free vertex node with a child t' such that $X_t = X_{t'} \cup \{v\}$. Notice that the vertex v is isolated in G_t. Therefore, we have the following formulas:

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

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. 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 be a forced vertices assignment function and g be a free vertices assignment function. Suppose that $g(v) = \mathrm{False}$. Then, we simply pass the value from the child's node. Otherwise, if $g(v) = \mathrm{True}$, we distinguish two cases. Firstly, let $f(u) = \mathrm{False}$. It means that v was added to a star by another free edge. If $f(u) = \mathrm{True}$, then v could be also added to a star by the edge vu. Thus, we obtain the following equation:

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

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

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

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 forced assignment function f, that satisfies $f(v) = f(u) = \mathrm{True}$, $\mathrm{dp}[t', f] = \mathrm{False}$ because we have already changed their values during introduce forced edge node. Thus, the formula looks as follows:

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

Forget free vertex node Let t be an 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 free vertices assignment function g, $g(v) = \text{True}$. Consequently, we obtain:

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

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. $f = f_1 = f_2$, for forced vertices assignment functions f, f_1, f_2.
2. $g(v) = g_1(v) \vee g_2(u)$, for every free vertex $v \in X_t \cap U$.

Intuitively, we make sure that the centers remains at the same position and at least one of the children's assignments added every free vertex to a star. Therefore, we get the following equations:

$$\text{dp}[t, f, g] = \bigvee_{g_1, g_2 \text{ match } g} dp[t_1, f_1, g_1] \wedge dp[t_2, f_2, g_2]$$

5.5.3. Complexity analysis

Observe that, except for a join node, every operation can be done in constant time. A naive approach to solve the disjunction would result in time $\mathcal{O}^*(2^{\text{tw}(G)})$. Thus, we could conclude with an algorithm solving Spanning Star Forest Extension parameterized by treewidth in time $\mathcal{O}^*(4^{\text{tw}(G)})$ as there are $O(|G| \cdot 2^{\text{tw}(G)})$ array entries. However, we can improve the run time. Thus, we want to introduce a smarter way of solving equations like this. Consider the following definition:

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, there is a theorem, stated in the *Parameterized Algorithms*, that says:

Theorem 5.4. *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.*

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

Lemma 5.7. *Given a join node t, one can calculate all its values in time $O(2^{tw(G)})$.*

Proof. Fix a forced vertices assignment f. 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 calculation that we do during a join node. Thus, $\mathrm{dp}[t, f, g] = (c_{t_1,f} *_c c_{t_2,f})(g^{-1}(1)) > 0$.

There are $2^{|X_t \cap V(F)|}$ different forced vertices assignments. For a given forced vertices assignment f, by Theorem 5.4, we can calculate values for f and every possible free vertices assignment g in time $2^{|X_t \cap U|}$. Thus, for a join node t we can fill its dynamic table cells in time $2^{|X_t \cap V(F)|} \cdot 2^{|X_t \cap U|} = 2^{|X_t|} \leq 2^{\mathrm{tw}(G)}$. □

Proof (of Theorem 1.12). Consider the algorithm described in the previous subsection. To calculate a single entry for introduce and forget nodes, we need constant time. By Lemma 5.7, 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 values 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. □

Proof (of Theorem 1.13). Let ϕ be an arbitrary instance of CNF-SAT problem with n variables. We apply a reduction from Lemma 5.2 and obtain an equivalent instance (G, F). Now, we need to prove that $tw(G) \leq n$. Thus we propose the following path decomposition. Let $B_1 = \{v[x_i] : 1 \leq i \leq n\}$. Then, we iteratively introduce and forget every vertex $v \in N[B_1] \cap U$. Next, for every vertex $v[x_i]$ we introduce vertex $v[\neg x_i]$ and forget $v[x_i]$. Finally, we repeat the second step, that is, we iteratively introduce and forget every vertex $v \in N[\{v[\neg x_i] : 1 \leq i \leq n\}] \cap U$. Observe that every bag of the decomposition consists of at most $n + 1$ vertices. Thus, $pw(G) = n$. Since $tw(G) \leq pw(G)$ we conclude that $tw(G) \leq n$.

To conclude, observe that if there was an algorithm solving Spanning Star Forest Extension parameterized by treewidth in $2^{o(\mathrm{tw}(G))}$, then it would contradict SETH. □

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] 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)

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

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

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