University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Adam Starak
Student no. 361021

Title in English

Master's thesis
in COMPUTER SCIENCE

Supervisor:
dr Michał Pilipczuk
Institute of Informatics

October, 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.*

Theorem 1.2. Minimal Spanning Star Forest *is* NP*-complete.*

Theorem 1.3. Minimal Spanning Star Forest *is* W[2]*-complete.*

Theorem 1.4. *Unless SETH fails, there is no algorithm for* Minimal Spanning Star Forest *that achieves running time $\mathcal{O}^*((2-\epsilon)^n)$ for any $\epsilon > 0$, where n is the number of stars.*

Theorem 1.5. Spanning Star Forest Extension *is* NP*-complete.*

Theorem 1.6. Spanning Star Forest Extension *parameterized by the number of forced edges does not admit a polynomial kernel.*

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. *Unless SETH fails, there is no algorithm for* Spanning Star Forest Extension *parameterized by the number of forced edges that achieves running time $\mathcal{O}^*((2-\epsilon)^{|F|})$ for any $\epsilon > 0$, where $|F|$ is the number of forced edges.*

Theorem 1.9. Spanning Star Forest Extension *parameterized by the number of free vertices admits a linear kernel.*

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

Theorem 1.11. Spanning Star Forest Extension *parameterized by treewidth can be solved in time 2^{tw}.*

Theorem 1.12. *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 $E(G)$ a set of vertices and edges respectively. Let $\deg_G(v)$ denote a degree of vertex v in 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. A graph P_k is a path of length k. A graph C_k is a cycle of length k. 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 *Spanning forest* T of a graph G is a union of trees that spans G. 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. A *star* of size 2 has two *candidates.*

2.2. Parameterized complexity

In computer science, *parameterized complexity* is a young branch of computational complexity theory. The first systematic paper was released in 1990 by Downey and Fellows. The further investigations of researchers led to numerous theorems and hypotheses. To begin with, let us formally define a parameterized problem. For the sake of clarity, all the definitions are taken from the book (parameterized complexity).

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 below to understand the concept:

Definition 2.2. 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 might be multiple different parameters for a single problem. For example, in this paper, we will investigate thoroughly a problem with respect to three different parameters. More interestingly, each parametrization yields different results.

Also, parameterized complexity helped to distinguish hardness of NP-complete problems. The first class is called FPT (fixed parameter tractable). A problem is in FPT if and only if it has an FPT algorithm defined below:

Definition 2.3. *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 called an XP class. Similarly, a problem is in XP if and only if it has an XP algorithm defined below:

Definition 2.4. *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 $n^{f(k)} \cdot n^c$ where c is a constant, independent of n, k, and f is a computable function.*

W-hierarchy is an alternative way of classifying hard problems. Since it is not the main topic of the paper, we just want to mention that W-hierarchy is an ascending chain of classes that follows: $W[1] \subseteq W[2] \ldots$. As an example, Independent Set is a W[1]-complete problem whereas Dominating Set is a W[2]-complete problem. It is not known what is the correlation between FPT and W[1] classes. In this paper, we assume that W[1] $\neq$ FPT.

Having discussed various complexity classes, let us introduce a tool for reducing one parameterized problem to another. The mechanism described below is widely used in the paper.

Definition 2.5. *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 $f(k) \times |x^{\mathcal{O}(1)}|$ for some computable function f.*

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

Definition 2.6. *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.*

Kernelization was not a highly explored area. Not until the year 2008, when researchers developed a technique to prove lower bounds on kernels. We discuss the framework as well as show an example further in this paper.

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 node**: a node t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$ for some vertex $v \notin X_{t'}$.
  - **Forget node**: a node t with exactly one child t' such that $X_t = X_{t'} \setminus \{w\}$ for some vertex $w \in X_{t'}$
  - **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. For a given graph G, we say that S is a spanning star forest if S spans G and every connected component of S is a star. We propose an algorithm working in linear time that outputs a spanning star forest or concludes with a NO-instance.

3.1. Decision Spanning Star Forest problem

It turns out that the problem formulated in such a way is relatively simple. Although, various variants described in this paper make it more complex. The following lemma easily clarifies all the concerns about its hardness.

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 trivially 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 maximal, connected component satisfying $u \in V(C)$. Based on the degree of u, we infer that $|V(C)| > 2$. Thus, we conclude that in a connected graph of 3 vertices there exists a vertex v such that $C \setminus v$ has no isolated vertices and neither has $G \setminus v$. 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 take any $w \in N_S[u]$. 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 DECISION SPANNING STAR FOREST.

Corollary 3.1. DECISION 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. Spanning Star Forest Problem

In this section we focus on obtaining an arbitrary solution for a given instance of SPANNING STAR FOREST. Firstly, let us introduce two claims which help to normalize an instance and make the algorithm more clear.

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 of 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, all that is left to do, is to design an algorithm which solves the problem for trees.

```
Input: connected graph $G$
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 $spanned[v] \neq 1$ then
        $u \leftarrow parent(v)$;
        $S \leftarrow S \cup \{(u, v)\}$;
        $spanned[v] = 1$;
        $spanned[u] = 1$;
    end
end
$v \leftarrow root(T)$;
if $spanned[v] \neq 1$ then
    $S \leftarrow S \cup \{(u, v)\}$ where $u$ is a child of $v$;
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. If the input graph is not connected, we run the algorithm separately on each component and then merge results based on Claim 3.1.

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.

Claim 3.3. *If $(u_1, u_2), (u_2, u_3) \in E(S)$, $u_2 = parent(u_1)$ and $u_3 = parent(u_2)$, then u_3 is the root.*

Proof. Suppose contrary, that u_3 is not the root. Then, both (u_1, u_2) and (u_2, u_3) have been added during the for loop. Contradiction because u_2 has been added to the solution during u_1's iteration and the algorithm should not have added the edge (u_2, u_3). Thus, u_3 is the root. □

From the Proposition 3.3 we infer that no two consecutive parents are added to the solution unless the last node is the root. It means that every root's child is either a center or a candidate. Thus, adding an arbitrary root's edge is a correct step in the algorithm. Moreover, there are no paths of length greater than 2. If there was one, it would mean that the root is connected to two different vertices which is false.

Lemma 3.2. *Algorithm 1 is correct.*

Proof. Clearly, S is a forest. So, it suffices to check that there are no isolated vertices and the solution does not induce a path of length 3. The first claim is true because we enumerate over all vertices and pair them up with its parent if a vertex has not been added to the solution in previous iteration. So is the second claim based on the information inferred from Proposition 3.3. □

Having proven the correctness of Algorithm 1 we proceed to the complexity analysis.

Theorem 3.1. *A solution for* Spanning Star Forest *can be found in linear time.*

Proof. An arbitrary spanning tree of any graph can be found in linear time. The main loop has $n-1$ iterations (every vertex is processed once), each of which takes constant time. Thus, the total runtime is linear. □

The problem stated without any constraints is simple. Both decision and normal version of the problem can be solved in linear time. The next variations investigated in this paper yield more complex results.

3.3. Maximal 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 looks slightly different than the previous one. From the other hand, the problem resembles Dominating Set, which is defined as follows:

Definition 3.1. 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 adjacent to one of the vertices from D.*

At first glance, one can say that a center is related to a dominating vertex whereas a ray is related to a dominated vertex. Candidates might be represented by either a dominating or dominated vertex. However, we cannot finish our research at this point because in Dominating Set isolated dominating vertices are allowed and some vertices are 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 3.2. *Given an instance (G, k) of Dominating Set Problem that does not contain any isolated vertices and a solution D, a* domination mapping *is a function $m : V(G) \backslash D \to D$ such that satisfies $(x, m(x)) \in E(G)$ for all $x \in Dom(m)$.*

Lemma 3.3. *Let G be a graph without isolated vertices and let D be a dominating set of minimal size. Then, there exists a domination mapping m such that m is surjective.*

Proof. Let m be a dominating mapping that maximizes $|im(m)|$. If m is surjective, then the proof is finished. Otherwise, there exists a vertex v such that $v \notin im(m)$. Consider the following cases::

1. Suppose $N_G(v) = \emptyset$. Contradiction, G has no isolated vertices.
2. Suppose $u \in N_G(v) \cap D$. Contradiction, D was said to be a solution of minimal size whereas $D \setminus \{u\}$ is a valid, smaller solution.
3. Suppose $u \in N_G(v) \setminus D$ and $w = m(u)$. If $|m^{-1}(w)| = 1$, then $((D \setminus \{v, w\}) \cup u)$ is a valid, smaller solution for a graph G. Contradiction.
4. Suppose $u \in N_G(v) \setminus D$ and $w = m(u)$. If $|m^{-1}(w)| > 1$ then a mapping:

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

   is a valid mapping that satisfies $im(m) \subsetneq im(m')$. Thus, one can create a new mapping m'' inductively such that m'' is surjective. Contradiction, we assumed that no such mapping exists.

Since all the first three cases led to a contradiction and the last one shows how to inductively expand the image of a domination mapping, we may conclude that there exists a domination mapping m such that m is surjective. □

Armed with the lemma, we are ready to prove the main theorem of the section.

Theorem 3.2. Minimal Spanning Star Forest *is NP-Complete.*

Proof. Membership in NP: The witness is a spanning star forest S. The verifier checks if all the vertices from input graph are included in S, there are no isolated vertices and S does not induce a P_3 nor $C_{>2}$.

Hardness: We show hardness by a reduction from Dominating Set that completes the proof. Given graph G and integer k, in PTIME, for every isolated vertex $v \in V(G)$ introduce a vertex v' and an edge (v, v'). Now, 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.

The backward implication is simple. Suppose S is a solution for (G', k). We create the dominating set D as follows: for every star of size 2 pick an arbitrary candidate that is present in graph G and for every star of size greater than 2 pick a center. Obviously $|D| \leq k$ because there are at most k connected components in S. Every vertex from G' is adjacent to one of the centers or candidates that we chose during construction of a set D.

To prove the forward implication, let D be a solution of minimal size for (G, k). Obviously, D is also a minimal dominating set for the graph G' as D contains every isolated vertex. Thus, by Lemma 3.3. there exists a mapping m that is surjective. Now, we claim that the graph $S = (V(G'), \{(x, m(x)) : x \in Dom(m)\})$ is a correct solution for the instance (G', k)

of MINIMAL SPANNING STAR FOREST. By surjectivity, there are no isolated vertices in S. Moreover, there is no path nor cycle of length 3 because m maps vertices from $V(G') \setminus D$ to D and for all $v \in V(G') \setminus D$, $deg_S(v) = 1$.

□

The problems look so similar that one could ask whether the reverse reduction is true. Indeed, with a small twist in the previous idea, one can prove the reverse reduction instantly. In the proof we only point out the construction because the concept of proving the instance equivalence remains the same.

Theorem 3.3. *There exists a parameterized reduction from* MINIMAL SPANNING STAR FOREST *to* DOMINATING SET *parameterized by the size of dominating set.*

Proof (sketch). Let (G, k) be an instance of MINIMAL SPANNING STAR FOREST. We create the instance (G', k') for dominating set as follows: let $G' = G$. If G contains an isolated vertex, then $k' = 0$. Otherwise, the value remains the same. Now, we claim that (G, k) is a YES-instance for MINIMAL SPANNING STAR FOREST if and only if (G', k') is a YES-instance for dominating set.

To prove the following reduction one can use the method which was described in Theorem 3.3 with a little remark: if an instance (G, k) contains an isolated vertex, then obviously it is a NO-instance for MINIMAL SPANNING STAR FOREST and so is (G', k') for dominating set because G' is not an empty graph. □

The theorem implies that MINIMAL SPANNING STAR FOREST is as hard as DOMINATING SET and the problems are *interreducible.* Thus, we can immediately obtain the following corollary.

Corollary 3.2. MINIMAL SPANNING STAR FOREST *is* W[2]-complete.

Chapter 4

Spanning Star Forest Extension

In this chapter, we significantly change the problem. Let G be a graph and $F \subseteq E(G)$ be a set of *forced edges*. In the SPANNING STAR FOREST EXTENSION the question, that we want to answer now, is whether there exists a spanning star forest S such that $F \subseteq E(S)$. We used three different parameters: number of forced edges, number of free edges and treewidth.

4.1. Preliminaries

4.1.1. Notation

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 3 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 denoted by U.

4.1.2. Instance normalization

Notice that this time we do not have any limit on the number of connected components. The hardness of the problem lays 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. try to remove vertices that are "far enough" from forced vertices.

Let (G, F) be an arbitrary instance of SPANNING STAR FOREST EXTENSION. Firstly, consider trivial cases.

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

Reduction 2 If F induces in G a path or a cycle of size at least 3, then it is a NO-instance.

For the sake of simplicity, if a vertex has one neighbor only, we can add the edge to the set of forced edges. There is no other way to create a spanning star forest without taking that particular edge.

Reduction 3 If a free vertex u satisfies $N[u] = \{v\}$, then add (u, v) to F.

Suppose that a subset of forced edges forms a star of size at least 3. Then, the forced center is already determined. Thus, we can remove from the instance all the edges that has exactly one end in forced ray and the other in a free vertex.

Reduction 4 Suppose F_R is a set of forced rays. Remove the set of edges $\{(u,v) : u \in F_R,\ v \in V(G) \setminus V(F)\}$

Moreover, notice that after exhaustive application of previous reductions, for all $v \in F_R$, it is true that $deg(v) = 1$. That is, forced rays are only connected to its forced centers. So, for every forced star, contracting the set of forced rays into a forced candidate, does not have an impact on the solution.

Reduction 5 Suppose that (G, F) is the output graph after exhaustive application of previous reductions. For every forced star, contract the set of forced rays into a forced candidate.

Let us summarize our work and infer how an input graph looks like. Currently, no two forced edges have a common vertex. So, we immediately obtain the following:

Corollary 4.1. *After exhaustive application of Reduction 5, F is a matching.*

We can distinguish three types of different forced edges. Ones that have outgoing edges from both ends, one end and ones that have no outgoing edges. For the sake of clarity, we can remove isolated forced edges.

Reduction 6 Remove isolated forced edges.

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

Claim 4.1. *G_P always has a solution.*

To prove the claim, we can simply apply lemma 3.1. G_P does not have any forced edges nor isolated vertices. All that is left to do is to prove that edges between G_P and G_{NP}, that were lost during partitioning, does not have any impact on the solution. The following theorem proves the intuition.

Theorem 4.1. *An instance (G, F) has a solution if and only if (G_{NP}, F) has one.*

Proof. The backward implication is trivial. Suppose S is a solution for (G_{NP}, F). We can partition G into G_P and G_{NP} and find a solution, say S', for a graph G_P. Then, $S \cup S'$ is a correct solution for G.

Now, consider the forward implication. Let S be a solution for (G, F). Assume contrary that there exists a vertex $u \in V(G_{NP})$ that does not belong to any star. Trivially, vertices from $V(F)$ are covered. Thus, $u \in V(G_{NP}) \setminus V(F)$. If $u \in V(G_{NP}) \setminus V(F)$ it follows that for all $(u,v) \in E(G)$, $(u,v) \in E(G_{NP})$. If it was not true, the vertex u would have been placed in the graph G_P during partitioning. Thus, there exists an edge $(u,v) \in E(S)$ such that $(u,v) \in E(G_{NP})$. Contradiction because u belongs to a star. □

Theorem 4.1 provides us the next rule:

Reduction 7 Update $G = G_{NP}$.

Now, we want to present the last set of reduction rules that we can infer after the application of previous ones:

Reduction 8 If $u \notin V(F)$, $(u, v), (u, w) \in E(G)$ and $(v, w) \in F$, then remove u.

Reduction 9 If $(u, v) \in F$ and $deg(u) = 1$, then apply $G = G \setminus N[v]$.

To sum up, after exhaustive application of reduction rules, we will either conclude that it is a NO-instance or output a very structured graph. There exists a matching F consisting of all the forced candidates. Additionally, all the free vertices are connected to at least two different ends of forced edges.

4.2. NP-completeness

We are going to show NP-completeness of Spanning Star Forest Extension by a reduction from CNF-SAT. Let us begin with a recall:

Definition 4.1. *CNF-SAT*

Let (G, F) be an instance of Spanning Star Forest Extension after normalization. Then, G consists of vertices of two types: ones that have only edges to vertices from $V(F)$ and ones that have exactly one forced edge (and potentially many free ones). Such a representation substantially simplifies further investigations. Indeed, we can prove that the problem is NP-complete.

Lemma 4.1. *There exists a polynomial time reduction from* CNF-SAT *to* Spanning Star Forest Extension.

Proof. Suppose ϕ is an arbitrary instance of CNF-SAT. ϕ is a formula written in CNF, so let $C = \{C_1, ..., C_m\}$ be a set of clauses and let $Vars = \{x_1, ..., x_n\}$ be a set of variables occuring 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 x_{l_i} in a clause C_j we introduce an edge (C_j, x_{l_i}). Now, we say that (G, F), that has been just described, has a spanning star forest if and only if there exists a satisfiable evaluation of variables.

Forward implication: Let S be a solution for (G, F). Then, a set of centers is a correct evaluation that satisfies the formula ϕ because every clause S has a witness.

To prove the backward implication, assume there exists an evaluation σ of variables that satisfies the formula. Let $tt = \{l_i : \sigma(l_i) = 1\}$. Note that C contains either x_i or $\neg x_i$. Now, let us construct a solution S. Firstly, include all the isolated edges. Then, for every vertex representing a clause C_i take an arbitrary l_j such that $l_j \in N(C_i) \cap tt$ and include edge (C_i, l_j) into the solution. The operation is safe. The set $N(C_i) \cap C$ is empty only if the clause C_i is empty. But, if the clause C_i is empty, then it is always satisfied. If the intersection is nonempty, then there exists a witness l_k that satisfies the clause. Finally, there exists an edge between the literal and the clause that joins the clause vertex to a star.

□

As we can see, clauses and variables in CNF-SAT are encoded as free vertices and forced edges in Spanning Star Forest Extension respectively. Thus, we get the immediate corollary:

Corollary 4.2. *A CNF-formula ϕ of n variables and m clauses can be reduced in polynomial time to a* Spanning Star Forest Extension *instance of n forced edges, m free vertices and $\mathcal{O}(n \cdot m)$ free edges.*

There is one more observation that we want to point out in this paper. 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 are going to present the construction only.

Lemma 4.2. *There exists a polynomial time reduction from* Spanning Star Forest Extension *to* CNF-SAT.

Proof (sketch). Suppose (G, F) is an arbitrary instance of Spanning Star Forest Extension. Every free vertex represents a separate clause. Every two forced candidates of a forced edge are represent literals x_i and $\neg x_i$. An edge between a forced candidate and a free vertex indicates an occurrence of a literal in a clause. □

Immediately, we obtain a symmetric corollary:

Corollary 4.3. *An instance (G, F) of* Spanning Star Forest Extension *can be reduced in polynomial time to a $CNF-SAT$ instance of $|V(G) \setminus V(F)|$ clauses, $|F|$ variables. The length of the formula is equal to $|E(G) \setminus F|$.*

Lemmas and corollaries described in this section are the key observations that we use in the next sections to prove lower bounds or reducibility.

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

Theorem 4.2. Spanning Star Forest Extension *is NP-complete.*

Proof. Membership in NP: The witness is a spanning star forest S. The verifier checks if every connected component is a star and if every forced edge is included in the solution.

Hardness: It has been already proven in Lemma 4.1 by a reduction from Spanning Star Forest Extension. □

4.3. 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 a number of forced edges. We show two major results: Spanning Star Forest Extension parameterized by the number of forced edges does not invoke a kernel of polynomial size and a lower bound under strong exponential hypothesis.

4.3.1. 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.

Definition 4.2. *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 4.3. *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 4.4. *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.

Let us give a simple example to illustrate how to use the tools. Recall that in HAMILTONIAN PATH problem, given a graph G, the question is whether there exists a path consisting of all the vertices from G. In LONGEST PATH problem, given a graph G and an integer k, we ask to find a path of size k..

Theorem 4.3. LONGEST PATH *does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly.*

Proof. We show that HAMILTONIAN PATH cross-composes into LONGEST PATH. Firstly, we need to find a polynomial equivalence relation $\mathcal{R}$. So, let $\mathcal{R}$ be a relation such that two graphs are in the same class if and only if they have the same number of vertices. Also, we introduce a separate class for a malformed input. The conditions from Definition 4.2 are trivially satisfied.

Thus, we proceed to the second step. We show an algorithm $\mathcal{A}$ as an evidence that HAMILTONIAN PATH cross-composes into LONGEST PATH. Let $G_1, ..., G_t$ be an arbitrary instance such that $G_1, .., G_t$ is equivalent with respect to $\mathcal{R}$. The algorithm outputs a disjoint union of graphs, further called G, and $k = |V(G_1)|$. Now, let us show that the two conditions stated in the Definition 4.3 hold. Obviously $k = |V(G_1)| \leq \max\limits_{1 \leq i \leq t} |G_i|$. Now, suppose that (G, k) has a solution, that is, it consists of a path of length k. G is a disjoint union of graphs $G_1, ..., G_t$. Moreover, based on relation $\mathcal{R}$, k is equal to the number of vertices of every graph G_i. Thus, there exists an index i for which G_i has a Hamiltonian Path. For the backward implication, we say that there exists an index i for which G_i has a Hamiltonian Path. (G, k) is a YES-Instance too because it is a union of disjoint graphs that consists of G_i.

Ultimately, having proven that an NP-hard problem HAMILTONIAN PATH cross-composes into LONGEST PATH, based on Corollary 4.4, we conclude the proof. □

4.3.2. 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 NP $\subseteq$ coNP/poly. To achieve this, we show a cross-composition from SPANNING STAR FOREST EXTENSION into itself. The proof shown below is not as feasible as the one that we discussed in the previous subsection. 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.

Theorem 4.4. *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.

Proof. Let $\mathcal{R}$ be a polynomial equivalence relation such that two instances, $(G_1, F_1), (G_2, F_2)$, are in the same class if and only if $|F_1| = |F_2|$. Additionally, $\mathcal{R}$ considers equivalently all the malformed graphs. Hence, we assume that all the input instances has the same number of forced edges. It is a natural choice that simplifies further investigations.

Now, we proceed to the construction process. Let $(G_1, F_1), ..., (G_t, F_t)$ be the family of t input instances of Spanning Star Forest Extension. For the sake of clarity, we duplicate some instances so that $t = 2^i$ for some integer i. Note that pessimistically, we double the input only.

We are now ready to formally define the output graph and a parameter. Firstly, we create a pair (G, F) by merging forced edges of every input instance. Observe, that we reduced the size of the parameter and now it fits the limit stated in Definition 4.3. Furthermore, we create an instance selector that encodes the OR behavior. We introduce $log(t)$ forced edges $(l_1, r_1), ..., (l_{log(t)}, r_{log(t)})$. Then, we proceed recursively to add edges. We start with a set $S = \{\{G_1, ..., G_t\}\}$. At step k, we iterate over every set of S and do the following: split the chosen set in the "middle", take the "left" subset and introduce edges from l_k to every free vertex included in the subset, take the "right" subset and introduce edges from r_k to every free vertex included in the subset, add splitted sets to S.

Of course, we did not exceed the stated limit for the parameter. After the addition of the instance selector, the output instance consists of $|F_1| + log(t)$ forced edges. Before we proceed to the second condition, observe that the instance selector guarantees that free vertices from any $t-1$ graphs will be spanned by stars that have a center in instance selector . Therefore, all that remains is to check if at least one graph has a spanning star forest. Hence, the backward implication is simple. Suppose that graph G_i has a spanning star forest. We select edges going from instance selectors in such a way that all graphs but G_i are spanned by a star forest. Thus, we can use forced edges from F to provide a spanning star forest for graph G_i. For the forward implication, suppose that the output instance has a spanning star forest. Indeed, instance selector covered at most $t-1$ graphs. Thus, there exists at least one index i for which (G_i, F_i) has a spanning star forest. □

4.3.3. Lower bound based on SETH

Previously in this chapter, we were proving various properties of Spanning Star Forest Extension problem. We showed the problem is NP-complete, it does not admit a polynomial kernel and we stated 9 reduction rules to simplify instances. In this subsection, we show a simple routine that solves Spanning Star Forest Extension. Furthermore, we prove that there does not exist a faster algorithm unless SETH fails.

Consider the following Algorithm 2. It simply iterates over all maximal sets of forced candidates that do not form any edge. 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 is capable of it, then the input is a NO-instance. Thus, we may conclude this part with the following theorem:

Theorem 4.5. Spanning Star Forest Extension *parameterized by the number of forced edges can be solved in time $\mathcal{O}^*(2^{|F|})$.*

Proof. There are $2^{|F|}$ possible sets. Every iteration takes linear time to process the set. □

```
Data: Pair $(G, F)$ after exhaustive application of reduction rules
Result: spanning star forest of $G$ extending $F$
$Centers \leftarrow \{C : |C| = |F| \text{ and } \forall u, v \in C,\ (u, v) \notin F\}$;
for $C \in Centers$ do
    if $G[C]$ spans $G$ then
        return YES-instance;
    end
end
return NO-instance;
```

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

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 the point.

Theorem 4.6. *Unless SETH fails, there is no algorithm for* Spanning Star Forest Extension *parameterized by the number of forced edges that achieves running time $\mathcal{O}^*((2-\epsilon)^n)$ for any $\epsilon > 0$, where n is the size of the set of forced edges.*

Proof. We have already presented a sufficient reduction in Lemma 4.1. Hence, we want to verify that the existence of an algorithm, running in time stated in the theorem, would contradict SETH. Therefore, suppose that Spanning Star Forest Extension can be solved in time $\mathcal{O}^*(2^{\delta n})$, for some $\delta < 1$. Assuming SETH, there exists a constant $q \geq 3$ such that q-SAT cannot be solved in time $\mathcal{O}^*(2^{\delta n})$, where n is the number of variables. Consider now the following algorithm for q-SAT: apply the polynomial reduction from Lemma 4.1 and then solve the resulting instance of Spanning Star Forest Extension using the assumed faster algorithm. Based on Corollary 4.3 we can transform an arbitrary instance of CNF-SAT of n variables into a Spanning Star Forest Extension instance with n forced edges. Thus, we could solve q-SAT in time that contradicts SETH. □

4.4. Parametrization by the number of non-isolated edges

4.5. Parametrization by treewidth

Bibliography

[1]