FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Induced star partition of graphs with respect to structural parameters
Student:	Bc. Xuan Thang Nguyen
Supervisor:	RNDr. Ondřej Suchý, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	until the end of summer semester 2023/2024

Instructions

Get familiar with the Induced Star Partition problem and with the basic notions of Parameterized Complexity.
Get familiar with known results for the problem.

Investigate whether some of the famous meta-theorems (such as Courcelle's theorem) can be applied to the problem.
Develop an explicit parameterized algorithm for the problem with respect to the size of the minimum vertex cover, the treewidth of the input graph, or the cliquewidth of the input graph or find major obstacles in developing such algorithms.

After consulting with the supervisor, select one of the above mentioned algorithms and implement it in a suitable language.

Test the resulting program on a suitable data, evaluate its performance.

Electronically approved by doc. Ing. Jan Janoušek, Ph.D. on 17 February 2023 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Induced star partition of graphs with respect to structural parameters

Bc. Nguyen Xuan Thang

Department of Theoretical Computer Science
Supervisor: RNDr. Ondřej Suchý, Ph.D.

May 4, 2023

Acknowledgements

I would like to thank my supervisor RNDr. Ondřej Suchý, Ph.D. for his support, guidance and advices throughout the whole time of creating this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 4, 2023 . . . .

Citation of this thesis

Nguyen, Xuan Thang. *Induced star partition of graphs with respect to structural parameters*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Abstract

An induced star partition of an undirected graph $G = (V, E)$ is a partition $S = (S_1, \ldots, S_q)$ of $V(G)$ such that each set S_i induces a star (graph isomorphic to $K_{1,r}$ for some $r \geq 0$). The INDUCED STAR PARTITION problem asks whether G admits an induced star partition of size q. This problem was proven to be NP-complete for each fixed $q \geq 3$ [1] and has an exact $3^n n^{O(1)}$ time polynomial space algorithm [1, 2]. To the best of our knowledge, there are no known algorithms based on structural parameters for the problem. We present the following results: (1) The problem is FPT when parameterized by the vertex cover number of the graph, and there is an exact $O(k^{2k+1}n^2)$ time algorithm, where k is the vertex cover number of the input graph. (2) The problem is FPT when parameterized by the treewidth of the graph and there is an exact $O(tw(G)^{2tw(G)} \cdot n)$ time algorithm, where $tw(G)$ is the treewidth of the input graph. (3) For a fixed q, the problem can be solved linear time on graphs with bounded cliquewidth. We also provide a simple implementation of our algorithm parameterized by the vertex cover number in C++ and evaluate its performance.

Keywords Induced star partition, Exact algorithms, FPT, Vertex Cover, Treewidth, Cliquewidth

Abstrakt

Zabýváme se problémem ROZDĚLENÍ NA INDUKOVANÉ HVĚZDY na neorientovaných grafech. Cílem je rozdělit graf na q množin $S_1, \ldots, S_q$ tak, že každá množina S_i indukuje hvězdu (graf izomorfní grafu $K_{1,r}$ pro nějaké $r \geq 0$). Je známo, že pro každé pevné $q \geq 3$ je tento problém NP-úplný [1]. Existuje ale exaktní algoritmus, který dokáže rozdělit graf na q indukovaných hvězd v čase $3^n n^{O(1)}$ a použije polynomiálně mnoho paměti [1, 2]. Není nám známo, že by existoval exaktní parametrizovaný algoritmus pro tento problém. V této práci předvedeme následující výsledky: (1) Problém patří do třídy FPT pokud budeme parametrizovat vrcholovým pokrytím grafu a existuje exaktní algoritmus běžící v čase $O(k^{2k+1}n^2)$, kde k je velikost minimálního vrcholového pokrytí grafu. (2) Problém patří do třídy FPT pokud budeme parametrizovat stromovou šířkou grafu a a existuje exaktní algoritmus běžící v čase $O(tw(G)^{2tw(G)} \cdot n)$, kde $tw(G)$ je stromová šířka grafu. (3) Pro každé pevné q platí, že problém lze vyřešit v lineárním čase na grafech s omezenou klikovou šířkou. Také poskytujeme jednoduchou implementaci algoritmu parametrizovaného vrcholovým pokrytím grafu v jazyce C++ a vyhodnotili jsme výkonnost implementace.

Klíčová slova Rozdělení na indukované hvězdy, Exaktní algoritmus, FPT, Vrcholové pokrytí, Stromová šířka, Kliková šířka

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

Graph partitioning is a widely studied topic in the field of computer science. To give some examples, partitioning a graph into k independent sets can be seen as finding a k-coloring of the graph and partitioning a graph into k stars (not necessarily induced) can be tied to the well known DOMINATING SET problem.

Consider the following team formation problem that was introduced in [3]: Assume that we have a number of agents. Our goal is to form at most q teams, such that each team contains at least one agent sharing information with every other team member. This problem can be modeled as partitioning G into q stars.

Another reason why one would want to study the problem of partitioning a graph into stars is that the problem can solve optimal shift scheduling of pharmacies [4].

In this work, we consider a variation on the partitioning problem called Induced Star Partition, where we want to partition the graph into q induced star. The main result that sparked our interest is that the problem is NP-complete for all fixed $q \geq 3$ [1]. Other known results will be discussed in Chapter 3 but our main concern is that there is no efficient exact algorithm for the problem.

In this thesis, we apply techniques and known results from the parameterized complexity theory on the problem and present the following results: (1) The problem is FPT when parameterized by the vertex cover number of the graph and there is an exact $O(k^{2k+1}n^2)$ time algorithm, where k is the vertex cover number of the input graph. (2) The problem is FPT when parameterized by the treewidth of the graph and there is an exact $O(tw(G)^{2tw(G)} \cdot n)$ time algorithm, where $tw(G)$ is the treewidth of the input graph. (3) For a fixed q, the problem can be solved linear time on graphs with bounded cliquewidth.

The structure of the thesis is as follows: In Chapter 2 we give a brief overview of the notation, definitions and techniques that will be used in this work. In Chapter 4 and Chapter 5 we present the algorithm parameterized by vertex cover and treewidth, respectively, and give proof of correctness for the algorithms.

The implementation of the algorithm parameterized by vertex cover will be discussed in Chapter 6. We not only describe the choices made during the implementation but also present experimental results.

Chapter **2**

Preliminaries

In this chapter we give an overview of the notations, definitions, and techniques that will be used in this thesis. We first start with basic definitions and notation of graphs. Afterwards, we formally define the problem that we will be working on in this thesis. Finally, we describe the parameterized complexity tools that were used in our algorithms.

2.1 Graph notation

We define a graph G as an ordered pair (V, E) where V is a set of vertices and E is a set of edges. We define $V(G)$ as the vertices of a graph G and often, if it is clear which graph we refer to, we simply use V. Similarly, we use E instead of $E(G)$ when the context is clear. For an *undirected* graph $G = (V, E)$ we define all edges $e = \{u, v\} \in E$ to be an unordered pair of vertices $u, v \in V$. For a *directed* graph $G = (V, E)$ we define all edges $e = (u, v) \in E$ to be an ordered pair of vertices $u, v \in V$. All graphs considered in this thesis are simple and finite, which means a graph does not have more than one edge between two vertices (multiedges), there are no edges that start and end at the same vertex (self loops) and V is a nonempty finite set. Often we will be working with an undirected graph G, thus we will simply refer to it as graph G. We use the standard notation of n denoting the size of $V(G)$ and m is the number of edges in $E(G)$.

Let $A \subseteq V(G)$ be a set of vertices of a graph G, then a graph $G[A] = (A, E')$ denotes an induced subgraph of G such that for each $u, v \in A$ it holds that $\{u, v\} \in E(G)$ if and only if $\{u, v\} \in E(G[A])$.

Let $B \subseteq V(G)$ be a set of vertices of a graph G, then $G - B = G[V \setminus B]$. If $B = \{v\}$, then we also use $G - v$ instead of $G - \{v\}$.

We denote $N_G(v)$ as the neighborhood of vertex $v \in V$ in an undirected graph G, which means $N_G(v) = \{u \in V(G) \mid \{u, v\} \in E(G)\}$. We also define neighborhood for a set: Let $S \subseteq V(G)$, then $N_G(S) = \cup_{v \in S} N_G(v)$. We define the degree of a vertex v as the number of adjacent vertices to v and we denote this value as $\deg_G(v) = |N_G(v)|$. If vertex v has $\deg_G(v) = 1$, then we call v a leaf vertex. If $\deg_G(v) = 0$ then v is an *isolated* vertex.

Let k be a positive integer, then $[k]$ denotes a set of integers $\{1, \ldots, k\}$. We call a family of sets $P = (P_1, \ldots, P_k)$ a partitioning of a set V if and only if the following conditions hold:

1. $\forall i, j \in [k] : i \neq j \implies P_i \cap P_j = \emptyset$,
2. $\bigcup_{i=1}^{k} P_i = V$,
3. $\emptyset \notin P$.

We call a set $I \subseteq V$ an independent set if for all pairs $u, v \in I$ it holds that $\{u, v\} \notin E$. For simplicity sake we call a set I *independent* if I is an independent set. A graph G is defined to be *bipartite* if V can be partitioned into two sets (A, B), such that both A and B are independent and for all edges $e = \{u, v\} \in E(G)$ it holds $u \in A, v \in B$, or $u \in B, v \in A$. Let $K_{m,n}$ denote a complete bipartite graph with partitions (A, B) such that both A and B are independent sets of sizes m and n, respectively, and for every pair $u \in A, v \in B$ it holds that $\{u, v\} \in E(K_{m,n})$.

We define a path P_i in a graph G as a sequence of vertices $(v_1, \ldots, v_{i+1})$ such that for each $j \in [i]$ it holds that $\{v_j, v_{j+1}\} \in E(G)$ and no edges repeat on the path. A path P_i has i vertices and the length of the path is $i - 1$ (the number of edges). Often, we will use the symbol P_i as a set of a partitioning, thus to denote a path in graph, we will explicitly refer to it as a path to avoid ambiguity. The distance between two vertices $u, v \in V(G)$ is defined as the length of the shortest path between u, v.

We call a graph G a *star* if G is isomorphic to $K_{1,r}$ for $r \geq 0$.

2.2 Problem Definition

The main problem that we will be solving in this thesis is the Induced Star Partition problem. In this section, we only define the problem. Other known results will be discussed in Chapter 3. We refer to [1] for the problem definition.

Definition 2.1. Let $G = (V, E)$ be an undirected graph, an *induced star partition* is a partition $S = (S_1, \ldots, S_q)$ of $V(G)$ such that for each $i \in [q]$ the graph $G[S_i]$ is isomorphic to a star.

We say that G admits an induced star partition S of size q if and only if $V(G)$ can be partitioned into q sets $(S_1, \ldots, S_q)$ and each set S_i induces a star.

Definition 2.2. The minimum q for which a graph G admits an induced star partition of size q is called the *induced star partition number*.

In [1], the authors presented three associated computational problems:

	INDUCED q-STAR PARTITION
INSTANCE:	A graph G.
GOAL:	Decide whether G admit an induced star partition of size q?

	INDUCED STAR PARTITION
INSTANCE:	A graph G and a positive integer q.
GOAL:	Decide whether G admit an induced star partition of size q?

	MIN INDUCED STAR PARTITION
INSTANCE:	A graph G.
GOAL:	Find the minimum q for which G admits an induced q-star partition.

Often not only do we want to find the induced star partition number but we also want to find the partition S. The pair (S_i^c, S_i^ℓ) partitions the i-th star S_i in S with a set of centers S_i^c and a set of leaves S_i^ℓ. We require the following conditions:

1. S_i^ℓ is independent,
2. $|S_i^c| = 1$,
3. $S_i^\ell \subseteq N_G(S_i^c)$.

2.3 Parameterized Complexity

One of the main results presented in [1] that sparked our interest is that the INDUCED STAR PARTITION problem is NP-complete[1] for each $q \geq 3$. One of our main concerns is that there is no known exact polynomial-time algorithm for NP-complete problems. For INDUCED STAR PARTITION, there is a $2^n n^{O(1)}$ time and exponential space algorithm and an exact $3^n n^{O(1)}$ time and polynomial space algorithm [1, 2].

As we can see, the running time of the previously known exact algorithms grow exponentially with the number of vertices n. Parameterized approach to solving problems allows us to extract more information from the problem using parameters. Usually, such an approach allows us to be more precise with the analysis or design more efficient algorithms and therefore limit the exponential factor not with n but by some function related to the parameter.

[1] Refer to [5] for the definition of NP-complete problems and the implication of computational complexity of such problems.

Let us give the precise definitions and a brief overview of techniques used to analyze problems when a parameter is present. All definitions presented in this section can be found in [6, 7]. Refer to these books for exact details and more.

Definition 2.3. A *parameterized* problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is fixed, finite alphabet. For an instance $(x, k) \in \Sigma^*$, k is called the *parameter.*

Definition 2.4. A parameterized problem L is called *fixed-parameter tractable* (FPT) if there exists an algorithm $\mathcal{A}$, a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant c such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ correctly decides whether $(x, k) \in L$ in time bounded by $f(k) \cdot |(x, k)|^c$. The complexity class containing all fixed-parameter tractable problems is called FPT.

One of most widely used techniques for parameterized complexity is preprocessing graphs using reduction rules. The goal is to a set of design algorithms to remove, in some way, uninteresting parts of the graph that can be solved very quickly. Reduction rules allow us to simplify the graph and shrink the size of the instance.

Definition 2.5. We say that two instances of Q are *equivalent* if $(I, k) \in Q$ if and only if $(I', k') \in Q$. A *reduction rule* for a parameterized problem Q is a function φ that maps an instance $(I, k) \in Q$ to an equivalent instance $(I', k') \in Q$ such that φ is computable in time polynomial in $|I|$ and $|k|$.

The whole framework heavily relies on the used parameter and the techniques and known algorithms rely on the choice of parameter. There are many kind of parameters, such as the size of the solution, structural parameters (vertex cover, treewidth, maximum degree, ...) or a combination of them. In this thesis, we will be mainly working with two very important structural parameters: vertex cover and treewidth.

2.3.1 Vertex cover

Definition 2.6. Let $G = (V, E)$ be a an undirected graph. We call a set $C \subseteq V(G)$ a *vertex cover* of G if and only if for each edge $e = \{u, v\} \in E(G)$ it holds that $u \in C$ or $v \in C$.

We call a C a *minimum vertex cover* if there is no vertex cover C' such that $|C'| < |C|$. For graph G we define k the vertex cover number if the size of minimum vertex cover equals k.

Vertex cover is an essential parameter of graphs and many results are known for finding the vertex cover of graphs. The problem of finding a minimum vertex cover is NP-complete [8] but FPT when parameterized by the solution size [6]. To the best of our knowledge, the problem has an exact $O(1.2738^k + kn)$ time algorithm [9].

The graph $G \setminus C$ has no edges and we will be exploiting this property to design our algorithm in Section 4.4.

2.3.2 Treewidth

Treewidth is another structural parameter that is often used in parameterized algorithms. In order to define treewidth, we need to first introduce the concept of tree decomposition as defined in [6].

Definition 2.7. Let G be an undirected graph. A tree decomposition of G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where 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:

1. $\bigcup_{t \in V(T)} X_t = V(G)$.
2. For every $\{u, v\} \in E(G)$, there exists a node t of T such that bag X_t contains both u and v.
3. For every $u \in V(G)$, the set $T_u = \{t \in V(T) \mid u \in X_t\}$ induces a connected subtree of T.

Definition 2.8. The *width* of tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ equals $\max_{t \in V(T)} |X_t| - 1$.

Definition 2.9. The *treewidth* of a graph G is the minimum possible width of a tree decomposition of G.

Algorithms based on using tree decomposition usually operate on a nice tree decomposition, thus we also define it the same way as in [6].

Definition 2.10. A rooted tree decomposition $(T, \{X_t\}_{t \in V(T)})$ with root r is called *nice* if the following conditions are satisfied:

1. $X_r = \emptyset$ and $X_\ell = \emptyset$ for every leaf ℓ of T.
2. Every non-leaf node of T is of one of the following three types:

   Introduce node: a node t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$ for some vertex $v \notin X_{t'}$; we say that v is *introduced* at t.

   Forget node: a node t with exactly one child t' such that $X_t = X_{t'} \setminus \{v\}$ for some vertex $v \in X_{t'}$; we say that v is *forgotten* at t.

   Join node: a node t with two children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$.

We use V_t to denote the union of all bags $X_{t'}$ such that t' is a node in the subtree rooted at t. Notice that $V_r = V(G)$. We also use G_t to denote the graph $G[V_t]$.

The following lemma allows us to compute a nice tree decomposition when only a tree decomposition is given.

Lemma 2.1. If a graph G admits a tree decomposition of width at most k, then it also admits a nice tree decomposition of width at most k. Moreover, given a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G of width at most k, one can in time $O(k^2 \cdot \max(|V(T)|, |V(G)|))$ compute a nice tree decomposition of G of width at most k that has at most $O(k|V(G)|)$ nodes.

Now we formulate the following lemma that will be pivotal for the proof of correctness of the algorithm presented in Section 5.2.

Lemma 2.2. Let T be a tree decomposition of graph G and $\{a, b\}$ be an edge of T. The forest $T - ab$ obtained from T by deleting edge $\{a, b\}$ consists of two connected components T_a (containing a) and T_b (containing b). Let $A = \bigcup_{t \in V(T_a)} X_t$ and $B = \bigcup_{t \in V(T_b)} X_t$. Then $A \cap B \subseteq X_a \cap X_b$ and there is no edge between $(A \setminus B)$ and $(B \setminus A)$ in G.

We call (A, B) a separation of G with separator $X_a \cap X_b$.

Let us also formulate a lemma that will be useful for the time complexity analysis.

Lemma 2.3. Let T be a tree decomposition of a graph G with width at most k. It is possible to construct a data structure in time $k^{O(1)}n$ that allows performing adjacency queries in time $O(k)$.

2.3.3 Cliquewidth

Another parameter that will be used in this thesis is called *cliquewidth*.

Definition 2.11 (Downey, Fellows [7])**.** Let G be an undirected graph. The smallest number of colors needed to construct G using the following operations is called *cliquewidth* of G.

1. $\emptyset_i$: create a vertex with color i;

2. $join(i, j)$: add edge between all vertices of color i and j;

3. $(i \to j)$: recolor all vertices of color i to color j;

4. $\bigsqcup$: take a disjoint union of G_1 and G_2.

This parameter can capture the structural complexity of the graph and generalizes the treewidth parameter. It has been proven that graphs with bounded treewidth also have bounded cliquewidth [10].

2.3.4 Courcelle's theorem

Courcelle's theorem is a frequently used tool to prove that a problem is FPT when parameterized by treewidth. The precise syntax and semantics of the formula used in Courcelle's theorem is fully described in [6, 7]. We only provide a high level intuition of the theorem and the language used in the theorem and a brief syntax description.

Definition 2.12 (Downey, Fellows [7]). MSO_2—Monadic second-order logic is a fragment of second-order logic used in order to describe graphs using logic formulae. The syntax uses

1. logical connectives $\wedge, \vee, \neg$,
2. variables for vertices, edges, sets of vertices, and sets of edges,
3. quantifiers $\forall, \exists$ that can be applied to variables,
4. binary relations:
    a) $u \in V$ where u is a vertex variable and V is a vertex-set variable,
    b) $e \in E$ where e is an edge variable and E is an edge-set variable,
    c) $inc(e, u)$, interpreted as edge e is incident on vertex u,
    d) $adj(u, v)$ interpreted as vertices u, v are adjacent,
    e) equality for variables of same type.

Theorem 2.1 (Courcelle [11]). Assume that φ is a formula of MSO_2 and G is an n-vertex graph equipped with evaluation of all free variables of φ. Suppose, moreover, that a tree decomposition of G of width t is provided. Then there exists an algorithm that verifies whether φ is satisfied in G in time $f(||\varphi||, t) \cdot n$, for some computable function f.

This meta-theorem will play an essential role proving that our problem is FPT when parameterized by treewidth. If the problem can be formulated using MSO_2, then the problem can be solved in linear time on graphs with bounded treewidth.

Furthermore, if the problem does not use quantifiers over sets of edges, then the formulae is MSO_1 and Courcelle et al. [12] proved that such problems can be solved in linear time on graphs with bounded cliquewidth if a construction sequence is given.

2.4 Flow network

In this section, we define a flow network in a similar way as in [13]. The maximum flow algorithm will play an essential role in our algorithm for the Induced Star Partition problem which will be described in Section 4.4.

Definition 2.13. Let $\mathcal{N} = (V, E)$ be a directed graph in which each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$. Let $s \in V$ be a source vertex and $t \in V$ be a target vertex. Then we call an ordered set $(\mathcal{N}, s, t, c)$ a *flow network*.

Definition 2.14. Let $(\mathcal{N}, s, t, c)$ be a flow network. A *flow* function (or simply *flow*) is any function $f : E \to \mathbb{R}_0^+$ satisfying the following conditions:

- $\forall e \in E : 0 \leq f(e) \leq c(e)$,
- $\forall v \in V \setminus \{s, t\}$:
$$\sum_{(x,v)\in E} f(x, v) = \sum_{(v,x)\in E} f(v, x).$$

We call f an *integer flow* if f is a flow and for each edge $e \in E$ it holds that $f(e) \in \mathbb{N}_0$.

Definition 2.15. Let f be a flow function, then $|f|$ denotes the *value* (also can be called *weight*) of the flow and

$$|f| = \sum_{(s,x)\in E} f(s, x) - \sum_{(x,s)\in E} f(x, s).$$

For a given network, we define the following problem.

	MAX FLOW
INSTANCE:	A network $(\mathcal{N}, s, t, c)$.
GOAL:	A flow f such that $|f|$ is maximal.

The famous Ford-Fulkerson's algorithm [14] proves that this problem can be solved in polynomial time when all edges of the network have rational capacity. Edmons-Karp [15] extended these results and showed that the MAX FLOW problem can be solved in $O(nm^2)$ time.

Another theorem that will be used in the thesis is about integer flows.

Theorem 2.2. If all edges in a flow network have integer capacity, then there is a maximum flow f such that for all edges e it holds that $f(e)$ is also an integer.

Proof. The famous min-cut max-flow theorem [13] shows that the weight of maximum flow $|f|$ is an integer. Edmons-Karp's algorithm [15] on such network will improve in each iteration the flow on edges of an augmenting path by an integer amount, thus the final flow f will also have integer flow on each edge. □

Chapter 3

Known Results

In this chapter we give a brief overview of known results for Induced Star Partition and its variants. We first start with hardness theorems for the problem on different classes of graphs, then move on to approximation schemes for the problem and then explore parameterized results. Afterwards we investigate other relaxed versions of the problem.

In [1] the authors studied a variant of a partitioning problem called Induced q-Star Partition which asked if G admits an induced star partition of size q. The main result that sparked our interest in the parameterized complexity analysis is that the problem is NP-complete when the graph is K_4-free and the number of stars is at least 3.

Theorem 3.1 (Shalu et al. [1]). For each fixed $q \geq 3$, the Induced q-Star Partition problem is NP-complete for K_4-free graphs.

Proof. Idea: The authors showed that Induced 3-Star Partition is NP-complete using reduction from 3-Coloring of triangle-free graphs to Induced 3-Star Partition. Then, for $q \geq 4$ a polynomial time reduction from Induced q-Star Partition to Induced 3-Star Partition can be constructed by adding $q - 3$ isolated vertices. □

The theorem shows hardness of the problem for $q \geq 3$, but when the number of partitions is at most 2, the problem can be solved in polynomial time [1].

Theorem 3.2 (Shalu et al. [1]). There is a polynomial time algorithm to decide whether a graph can be partitioned into at most 2 induced stars.

Proof. Idea: We can check if G is a star in polynomial time and verify if $q = 1$. To check if G can be partitioned into 2 stars, we exhaustively consider all pairs of vertices $x, y \in V(G)$ and set them as centers of each star respectively. Then, we try to partition $V(G) \setminus \{x, y\}$ into two sets (A, B) such that $G[A \cup B]$ is bipartite and $A \subseteq N_G(x)$ and $B \subseteq N_G(y)$. If we succeed to partition the vertices, then $(S_1^c, S_1^\ell) = (\{x\}, A)$ and $(S_2^c, S_2^\ell) = (\{y\}, B)$ induce a star. $\square$

The technical condition of G being K_4-free stems from the fact that if G is K_3-free (G does not have a triangle as subgraph), then the problem of partitioning the graph into q induced stars is equivalent to finding a dominating set of size q in a triangle free graph [1].

Lemma 3.1. Let G be K_3-free, then G admits an induced star partition of size q if and only if the G has a dominating set of size q.

Proof. A dominating set $D \subseteq V$ is such a set that for every $v \in V$ it holds that either v is in D or one of neighbours of v is in D. The graph G is triangle-free, thus for all $v \in V$ it holds that $N_G(v)$ is an independent set.

Suppose we have a dominating set $D = \{d_1, \dots, d_q\}$ of size q, then we set centers $S_i^c = d_i$ for $i \in [q]$. Each vertex $v \in V \setminus D$ has at least one of its neighbours in D, therefore we can assign v to at least one star.

If we have an induced star partition S of G and $|S| = q$, then the set $D = \bigcup_{i=1}^q S_i^c$ is trivially a dominating set, as every vertex is either a center, or a leaf vertex that is adjacent to a center vertex. $\square$

As we can see, the problem is closely related to the DOMINATING SET problem and known results for DOMINATING SET can be applied to show some of the results that will be mentioned in this chapter.

We first return to the NP-hardness of the problem and summarize all classes of graphs for which such results are known. The decision version of the Induced Star Partition problem is NP-complete for the following classes of graphs:

- chordal bipartite graphs [16],
- $(C_4, \dots, C_{2t})$-free bipartite graphs for every fixed $t \geq 2$ [17],
- subcubic bipartite planar graphs [1],
- line graphs [1],
- $K_{1,5}$-free split graphs [1],
- co-tripartite graphs [1].

The problem is NP-complete, thus we do not expect a polynomial-time deterministic algorithm to exist unless P=NP. There are some known exponential-time exact algorithms for the problem using standard set partitioning techniques: there is an exact $3^n n^{O(1)}$ time and polynomial space algorithm and an exact $2^n n^{O(1)}$ time and exponential space algorithm [1, 2].

For the following classes of graphs, a polynomial algorithm is known:

- trees [18],
- convex bipartite graphs [19],
- cluster graphs [1],
- $K_{1,2}$-free graphs [1].

In practice, computing the exact optimum of an NP-complete problem can be very time consuming, thus we often use an approximation of the actual optimal solution instead. In [1] the authors also studied a variant of the problem called Min Induced Star Partition and gave the following results.

Theorem 3.3 (Shalu et al. [1])**.** It is NP-hard to approximate Min Induced Star Partition to within $n^{\frac{1}{2}-\varepsilon}$ for all $\varepsilon > 0$.

Theorem 3.4 (Shalu et al. [1])**.** The Min Induced Star Partition problem has a polynomial time $\frac{r}{2}$-approximation algorithm for $K_{1,r}$-free graphs, where $r \geq 2$.

The Theorem 3.4 also implies that there is a 1.5-approximation algorithm for line graphs and co-bipartite graphs [1].

Theorem 3.5 (Shalu et al. [1])**.** The Min Induced Star Partition problem has a polynomial time 2-approximation algorithm for split graphs.

For triangle free graphs we can apply known results for dominating set and the following results for Min Induced Star Partition can be obtained:

- there is a greedy algorithm that can compute a $O(\log n)$-approximation [20],
- there exists a constant $c > 0$ such that the problem has no $c \log n$-approximation algorithm unless P=NP [21],
- let Δ be the maximum degree, then there is a $(\Delta + 1)$-approximation algorithm [22],

We now move on to the parameterized complexity analysis. From the parameterized complexity point of view, the problem is W[2]-complete for bipartite graphs and FPT for graphs of girth at least five when parameterized by the number of induced stars in the partition [23]. To the best of our knowledge, no other results are known for the problem from the parameterized complexity point of view and there seem to be no mention of exact parameterized algorithms using vertex cover or treewidth.

So far, we have only analyzed the problem when each set in partition S of G is an *induced* stars. In [24] the authors analyzed a similar problem called Constrained Star Partition Problem (CStarP)

	CStarP
Instance:	A graph G and a positive integer q.
Goal:	A star partition of cardinality q.

A star partition is a partition $A = (A_1, \dots, A_q)$ of G such that for all $i \in [q]$ it holds that $|A_i| \geq 2$ and $G[A_i]$ contains a spanning star (a star as subgraph that covers all vertices S_i). Compared to our problem, the set A_i cannot be of size one and furthermore, A_i does not have to induce a star—$G[A_i]$ just has to contain a star as a subgraph.

The main results for CStarP is that it is NP-complete, but a star partition of size q can be found in polynomial time on graphs with bounded treewidth and in $O(|V|^2)$ time on trees [24]. Unfortunately, there are no known results of applying these results on the Induced Star Partition problem and to the best of our knowledge, the proof cannot be easily modified for Induced Star Partition.

In [3], the authors studied the problem of partitioning the graph into k mutually disjoint sets of almost same size and such that each set contains a star (does not have to be induced) for subclasses of perfect graphs.

A more generalized problem called Partition Into H has also been studied in the past. For a fixed graph H, the question is whether a graph G can be partitioned into mutually disjoint sets of the same size and such that each set induces the graph H. This problem has been proven to be NP-complete for any fixed graph H on at least 3 vertices [25], but has a polynomial time algorithm for $H \approx K_2$ [26]. From the parameterized complexity point of view, it has been proven that for any fixed connected graph H, the Partition Into H problem can be expressed in an MSO_2 formula, therefore it is FPT when parameterized by treewidth [27].

Chapter **4**

Algorithm parameterized by vertex cover

In this chapter, we prove that Induced Star Partition is FPT when parameterized by the vertex cover number of the input graph. We first show a simple reduction rule for the problem and then show that the parameter q, the induced star partition number, can be bounded by the vertex cover number. Then, we show a negative result — it is not possible to design an algorithm that would first choose the centers of the stars from a vertex cover and then partition the leaf vertices. Finally, we present an $O(k^{2k+1}n^2)$ time algorithm for the problem and then provide proof of correctness of the algorithm.

4.1 Reduction rule and bounds

Let $G = (V, E)$ be an input graph with $|V(G)| = n$ vertices and q be the number of stars we want to partition the graph G into. Let $C \subseteq V(G)$ be a minimum vertex cover of G and $|C| = k$ be the vertex cover number. Also assume that C was given on the input together with G and q for simplicity. If the vertex cover is not given, there is an algorithm that can compute C in $O(1.2738^k + kn)$ time [9].

We first introduce a reduction rule that can deal with isolated vertices (vertex with $\deg_G(v) = 0$) in G.

Reduction rule 1. If G contains an isolated vertex v, delete v from G. The new instance is $I' = (G - v, q - 1)$.

Observation 4.1. Reduction rule 1 is correct. Vertex v has to be part of some star in a partitioning. Vertex v has no neighbours, thus it cannot be a leaf vertex and has to induce a center by itself.

Theorem 4.1. Let (G, q) be an input instance such that Reduction rule 1 is not applicable to (G, q) and k be the vertex cover number of G. If $q \geq k$, then (G, q) is a YES-instance.

Proof. Let $C = \{c_1, \ldots c_k\}$ be the minimum vertex cover of G and k be its size. We then construct a partition $S = (S_1, \ldots, S_s)$ of $V(G)$ such that each $S_i \in S$ induces a star. Let each S_i be a union of the set containing the center $S_i^c \neq \emptyset$ and the set of leaves S_i^ℓ.

We first select an arbitrary subset $A = \{a_1, \ldots, a_{q-k}\} \subseteq (V \setminus C)$ of size $q - k$. For each star S_{i+k} , where $i \leq q - k$, we set $(S_{i+k}^c, S_{i+k}^\ell) = (\{a_i\}, \emptyset)$. Each of the set created this way trivially induces a star with 1 vertex.

Now we construct the other k stars in the following way. For $i \leq k$, we set the center $S_i^c = \{c_i\}$. We know that $V \setminus C$ is an independent set and for each vertex $v \in (V \setminus C)$ there is an adjacent vertex from C because there are no isolated vertices. Thus for the rest of unused vertices in $V \setminus (A \cup C)$, we select any of its adjacent vertex $c_i \in C$ and add it to S_i^ℓ. We claim that each S_i then induces a star. Each S_i has a defined center S_i^c, therefore it is not empty. Furthermore, the leaves S_i^ℓ is a subset $N_G(S_i^c)$ as described in the construction. Finally S_i^ℓ indeed is an independent set because $S_i^\ell \subseteq (V \setminus C)$. □

Using Theorem 4.1, we can bound the value q by the vertex cover number in our algorithm. Whenever an instance where $q \geq k$ is given on the input, we construct a solution as described in the proof of Theorem 4.1. Let us assume from now on that $q < k$.

4.2 Vertex cover and center of stars

We further explore this technique of setting the center of the star S_i as one of the vertices from the given vertex cover. One might assume that there could exist an algorithm that would first find the centers of the stars in the vertex cover and then partition the rest of the vertices as leaf vertices. The following theorem proves that the idea does not work.

Theorem 4.2. There exists a family of graphs $\mathcal{G}$ such that for every $q \geq 3$ there exists a graph $G^q \in \mathcal{G}$ with star partition number q such that for every possible induced star partition S of size q and for every minimum vertex cover C of G^q, there exists a star $S_i \in S$ such that $S_i^c \cap C = \emptyset$.

We first describe a small gadget that we will use in the construction of G^q.

Definition 4.1. Let R^n be a graph on $n = |V(R^n)| \geq 4$ vertices. Let us label two special vertices $a, b \in V(R^n)$. The gadget R has the following edges:

1. $\{a, b\} \in E(R^n)$,
2. $\forall u \in (V(R^n) \setminus \{a, b\}) : \{a, u\} \in E(R^n) \wedge \{b, u\} \in E(R^n)$.

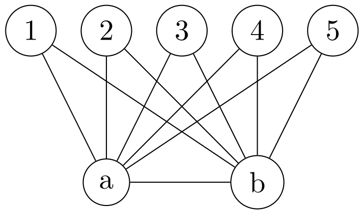

Figure 4.1: Gadget R^7 with 7 vertices.

An example of the gadget R^7 with 7 vertices is shown in Figure 4.1.

Observation 4.2. The set $V(R^n) \setminus \{a, b\}$ is an independent set.

Lemma 4.1. For every $n \geq 4$, the vertex cover number of R^n is 2 and the only minimum vertex cover of R^n is $\{a, b\}$.

Proof. We know that $a \in V(R^n)$ is adjacent to $b \in V(R^n)$, therefore a or b has to be part of the vertex cover. Without loss of generality assume that $a \in C$. The edges $\{b, u\}$ for $u \in V(R^n) \setminus \{a, b\}$ also have to be covered and only one vertex from $V(R^n) \setminus \{a, b\}$ cannot cover all $n - 2 \geq 4 - 2 = 2$ edges. Thus b is also part of the minimum vertex cover. $\square$

Lemma 4.2. For every $n \geq 4$, the induced star partition number of R^n is 2.

Proof. The graph R^n contains a triangle, therefore it is not isomorphic to a star and the induced star partition number is at least 2. We now show that R can be partitioned into 2 sets such that each set induces a star. Let $(S_1^c, S_1^\ell) = (\{a\}, V(R) \setminus \{a, b\})$ and $(S_2^c, S_2^\ell) = (\{b\}, \emptyset)$. The set S_2 induces a star with 1 vertex. The set $V(R) \setminus \{a, b\}$ is independent and each vertex $u \in V(R) \setminus \{a, b\}$ is adjacent to a as defined by the construction. Thus S_1 also induces a star. $\square$

We proceed to show the proof for Theorem 4.2.

Proof. To construct G^q, we take a disjoin union of $q - 1$ previously described gadgets R^n for $n \geq 2$ and add a new special vertex v. We also add edge $\{v, b_i\}$ for every gadget (meaning we add $q - 1$ edges) where b_i is the special vertex b from i-th gadget R_i^n. An example of graph G^q for $q = 4$ is shown in Figure 4.2.

We claim that the induced star partition number of G^q is exactly q and that the only possible way to partition G^q into q stars is in the following way. For each $i \in [q - 1]$ we have a star $(S_i^c, S_i^\ell) = (\{a_i\}, V(R_i^n) \setminus \{a_i, b_i\})$ as in the proof of Lemma 4.2. The last star is $(S_q^c, S_q^\ell) = (\{v\}, \{b_1, \ldots, b_{q-1}\})$. The set S_q^ℓ is trivially an independent set and each b_i is adjacent to v by the construction. This proves that the induced star partition number of G is at most q.

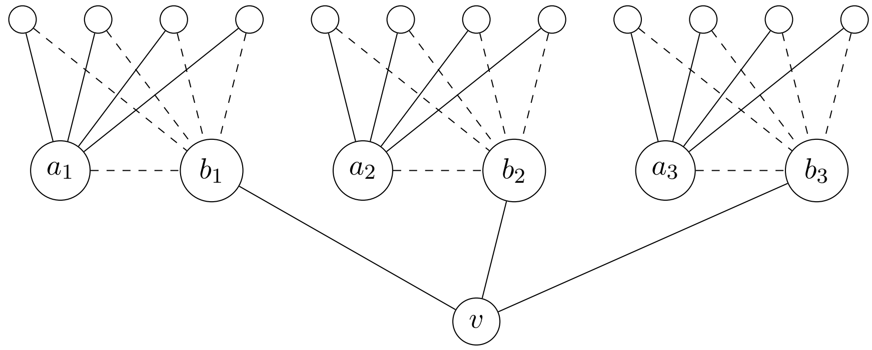

Figure 4.2: Graph G^4 consisting of 3 gadgets R^6 and a special vertex v. All edges of G^4 are represented either as dashed or solid line. Solid lines show induced edges of star S_i. Dashed lines represent edges between two different stars.

Assume towards a contradiction that there is another star partitioning $\widehat{S}$ of size $\widehat{s} \leq q$. The vertex v has to be part of some set $\widehat{S}_i \in \widehat{S}$ because $\widehat{S}$ is a partitioning. Without loss of generality assume that $i = 1$. We distinguish 2 cases, either $v \in \widehat{S}_1^\ell$ or $v \in \widehat{S}_1^c$.

First, assume that $v \in \widehat{S}_1^c$ is the center of star $\widehat{S}_1$. Vertex v is adjacent only to vertices b_i, thus $\widehat{S}_1^\ell \subseteq \{b_1, \dots, b_{q-1}\}$. The graph $G - \widehat{S}_1$ has $q - 1$ components, therefore at least $q - 1$ sets are needed to partition $G - \widehat{S}_1$ into induced stars and $|\widehat{S}| \geq 1 + (q - 1) = q$. Assume that there is an index $i \in [q - 1]$ such that $b_i \notin \widehat{S}_1$. Then the gadget R_i^n is unchanged in $G - \widehat{S}_1$ and two sets are needed to cover the gadget. This would imply that $|\widehat{S}| > q$ and $\widehat{S}$ is not minimal. We can conclude that $\widehat{S}_1^\ell = \{b_1, \dots, b_{q-1}\}$. The graph $G - \widehat{S}_1$ therefore is a union of $q - 1$ stars. We assumed that $n \geq 4$ in each gadget R_i^n, thus the center is unambiguously a_i and the set of leaves is $V(R_i^n) \setminus \{a_i, b_i\}$. This is exactly the construction of S.

Otherwise let us consider the case $v \in \widehat{S}_1^\ell$ is a leaf vertex of $\widehat{S}_1$. Then, one of its neighbours b_i is the center of $\widehat{S}_1$, let it be $b_1 \in V(R_1^n)$. The only vertices that are adjacent to b_1 are $(V(R_1^n) \setminus \{b_i\}) \cup \{v\}$ and we conclude that $\widehat{S}_1 \subseteq V(R_1^n) \cup \{v\}$. The subgraph $G[V(R_1^n) \cup \{v\}]$ is not a star, therefore $\widehat{S}_1$ could not have covered $G[V(R_1^n) \cup \{v\}]$. Then, the graph $G - \widehat{S}_1$ is a disjoint union of $q - 2$ gadgets R^n and part of the gadget R_1^n that was not covered by $\widehat{S}_1$. The induced star partition number of $G - \widehat{S}_1$ is $q' \geq 2(q-2)+1$: At least one set is needed to cover $R_1^n - (\widehat{S}_1 \setminus \{v\})$ and exactly $2(q-2)$ sets are needed to cover the $q-2$ gadgets (refer to Lemma 4.2). Therefore $|\widehat{S}| = q' + 1 \geq 2(q-1)$. We also assumed $q \geq 3$, which implies that $|\widehat{S}| \geq 2(q - 1) > q = |S|$.

We also claim that $C = \bigcup_{i=1}^{q-1} \{a_i, b_i\}$ is the only minimum vertex cover of G^q. Every edge $e \in E(R_i^n)$ for $i \in [q - 1]$ is covered because $\{a_i, b_i\} \subseteq C$. Then, the edges incident to the newly added vertex v is also covered because $b_i \in C$ for $i \in [q - 1]$.

Now consider towards a contradiction that there is another vertex cover C' of size $|C'| \leq |C|$. First, if $v \notin C'$, then all b_i are part of C' because b_i is adjacent to v. Then, the edges $\{a_i, u\}$ for $i \in R_i^n \setminus \{a_i, b_i\}$ also need to be covered. We want C' to be minimal, thus $a_i \in C'$, otherwise $|N_{G^q}(a_i) \setminus \{b_i\}| = |n-2| > 1$ vertices would have to be included instead. This is exactly the vertex cover C, thus let us now consider $v \in C'$. The graph $G - v$ is a disjoint union of $q-1$ gadgets and to cover each gadget, 2 vertices are needed as proven in Lemma 4.1. Thus $1 + 2(q-1) = |C'| > |C| = 2(q-1)$ and C' is a not a minimum vertex cover.

We showed that the described construction of S is the only way to partition the graph G^q in to q stars, where q is the induced star partition number of G^q. We also showed that G^q has only one minimum vertex cover C. We can observe that $v \notin C$ and $v \in S_i^c$ which concludes the proof that the center does not have to be in the vertex cover. □

The idea of first finding the centers of the stars (not necessarily from vertex cover) and then partitioning the rest of the vertices into leaves was used to show that the Induced Star Partition has a polynomial algorithm for each $q \leq 2$ [1]. For $q \geq 3$ this idea cannot be simply extended: Let $S^c = (\bigcup_{i=1}^{q} \{S_i^c\})$ be the union of all centers of each considered choice of q centers. The goal is to partition the set $V(G) \setminus S^c$ into at most q sets, each labeled as S_i^ℓ, such that each S_i^ℓ is an independent set and $S_i^\ell \subseteq N_G(S_i^c)$. This approach can be interpreted as finding a proper coloring of $V(G) \setminus C$, where each vertex $v \in V(G) \setminus C$ has a list of candidates (center S_i^c) that v can be part of. The list of candidates can be interpreted as a list of available colors for v. In other words, for each $v \in V(G) \setminus C$ we create a list of colors corresponding to $N_G(v) \cap S^c$. The standard list coloring problem when parameterized by the vertex cover number is $W[1]$-hard [28], thus it is not very prospective to first find the centers and then partition the leaves.

4.3 Partitioning the vertex cover

We propose an algorithm that can solve the Induced Star Partition problem in $O(k^{2k+1}n^2)$ time when parameterized by the vertex cover number k. We first give a high level idea, then describe the algorithm and finally give proof of correctness for the algorithm.

Our algorithm extends the idea of first working with the given vertex cover C and then match up the rest of the unused vertices $V(G) \setminus C$. We first try out all possible partitioning of C into q sets $P = (P_1, \ldots, P_q)$, then in polynomial time try to assign each vertex from $V \setminus C$ to a partial star $P_i \subseteq C$ as a leaf vertex or as a center. If we are able to assign every vertex from $V \setminus C$, then we have a solution.

Lemma 4.3. Let $G = (V, E)$ be an input graph without isolated vertices, C be a vertex cover of G of size k and q be the induced star partition number of G. Then there exists a induced star partition $S = (S_1, \dots, S_q)$ of $V(G)$, such $S_i \cap C \neq \emptyset$ for all $i \in [q]$.

Proof. Assume towards a contradiction that for every valid solution S of the Induced Star Partition problem on G, there is a star S_i such that $S_i \cap C = \emptyset$. Let S be an induced star partition with the least number of sets S_i that do not have a vertex from C.

Let's analyze an arbitrary $S_i \in S$ such that S_i only contains vertices from $V(G) \setminus C$. The set $V(G) \setminus C$ is an independent set, thus S_i can only contain one vertex from $(V(G) \setminus C)$, otherwise $G[S_i]$ would not be connected and could not induce a star. Let $(S_i^c, S_i^\ell) = (\{v\}, \emptyset)$. We assumed that the graph G has no isolated vertices, therefore v is adjacent to some vertices from C. Let us distinguish the following cases.

1. If there is a vertex $u \in N_G(v) \subseteq C$ adjacent to v, such that $u \in S_j^\ell$ for some $j \neq i$ and $|S_j| \geq 3$, then we remove u from the star S_j and add u to S_i. To be more precise, we create 2 new sets $(\widehat{S}_i^c, \widehat{S}_i^c) = (\{u\}, \{v\})$ and $(\widehat{S}_j^c, \widehat{S}_j^\ell) = (S_j^c, S_j^\ell \setminus \{v\})$. Then we construct $\widehat{S} = (S \setminus \{S_i, S_j\}) \cup \{\widehat{S}_i, \widehat{S}_j\}$. The set $\widehat{S}_i$ trivially induces a star $K_{1,1}$ and intersects C because $u \in C$. The set $\widehat{S}_j$ also induces a star as we just removed a vertex from $K_{1,r}$ and now we have $K_{1,(r-1)}$. Furthermore $G[S_i]$ was a tree on at least 3 vertices, thus it had at least 2 edges and u could have covered only 1 edge (recall u is a leaf vertex in $G[S_i]$). This all implies that $(\widehat{S}_j \cap C) \neq \emptyset$. The set $\widehat{S}$ is an induced star partition of G, but has less sets $\widehat{S}_i$ that do not intersect C than S. This is a contradiction with how we chose S.

2. Assume that v is adjacent to some $u \in C$ such that $u \in S_j^\ell$ for some $j \neq i$ and $|S_j| = 2$. Let $(S_j^c, S_j^\ell) = (\{w\}, \{u\})$. We distinguish the two following cases:

   $w \notin C$: We remove S_i and create $(\widehat{S}_j^c, \widehat{S}_j^\ell) = (\{u\}, \{w, v\})$. Both vertices w and v are not from the vertex cover C so they cannot be adjacent and $\widehat{S}_j^\ell$ is independent. Thus $\widehat{S} = (S \setminus \{S_i, S_j\}) \cup \{\widehat{S}_j\}$ is a solution of size $q - 1$ which is a contradiction with q being the induced star partition number.

   $w \in C$: We remove u from S_j and add it to S_i. To be more precise, we create 2 new sets $(\widehat{S}_i^c, \widehat{S}_i^\ell) = (\{u\}, \{v\})$ and $(\widehat{S}_j^c, \widehat{S}_j^\ell) = (\{w\}, \emptyset)$. Then we construct $\widehat{S} = (S \setminus \{S_i, S_j\}) \cup \{\widehat{S}_i, \widehat{S}_j\}$. Trivially both new sets induce a star and both sets intersect C. Thus $\widehat{S}$ has less sets $\widehat{S}_i$ that do not have a vertex from C. This is a contradiction with how we chose S.

3. Finally assume that for every vertex $u \in N_G(v) \subseteq C$ it holds that $S_j^c = \{u\}$ for some $j \neq i$. Then we any choose an arbitrary $u \in N_G(v)$ such that $\{u\} = S_j^c$, remove the set S_i from S, and move u into S_j^ℓ. Meaning that we construct $(\widehat{S}_j^c, \widehat{S}_j^\ell) = (\{u\}, S_j^\ell \cup \{v\})$. Then we set $\widehat{S} = (S \setminus \{S_i, S_j\}) \cup \{\widehat{S}_j\}$. The set $\widehat{S}_j$ still induces as a star: The set $\widehat{S}_j^\ell$ is an independent set because S_j^ℓ by assumption was an independent set and v is not adjacent to any leaf vertex. The set S_j^ℓ was by assumption a subset of $N_G(u)$ and v is adjacent also to the center u. We created a smaller induced star partition $\widehat{S}$ of size $q-1$ which is a contradiction with q being the induced star partition number of G.

In all three cases we were able to obtain either a smaller induced star partition or show that there is an induced star partition with less sets not intersecting vertex cover. □

We can conclude that if G can be partitioned into q stars, where q is the induced star partition number of G, then there exists an induced star partition S such that each set S_i contains some vertices from the vertex cover. Our algorithm will be looking exactly for this partition S. Notice that our previous lemma did not require C to be a minimum vertex cover and any (even not optimal) vertex cover can be used. This means that just an approximation of the minimum vertex is enough. On the other hand, our algorithm will have a multiplicative factor of k^{2k} therefore a small vertex cover is highly favorable.

We know now that the idea of first choosing the centers from the vertex cover does not work, but how is a vertex cover C partitioned within the induced star partition S? We now proceed with the analysis of $S \cap C$.

Let S be a induced star partition of $V(G)$ of size q that satisfies Lemma 4.3. Let us label $P = (P_1, \ldots, P_q)$, where for each $i \in [q]$ we set $P_i = P_i^c \cup P_i^\ell = S_i \cap C$ and $(P_i^c, P_i^\ell) = (S_i^c \cap C, S_i^\ell \cap C)$. We first analyze the properties each P_i.

Observation 4.3. Each set $P_i = S_i \cap C$ is either an independent set or $G[P_i]$ induces a star.

Lemma 4.4. Let $(P_i^c, P_i^\ell) = (S_i^c \cap C, S_i^\ell \cap C)$ and $P_i^c = \emptyset$, then $|S_i \setminus P_i| = 1$.

Proof. We know that P_i^c is empty, which means that the center S_i^c is not part of the vertex cover and $N_G(S_i^c) \subseteq C$. We also know that $S_i^\ell \subseteq N_G(S_i^c) \subseteq C$, thus $P_i^\ell = S_i^\ell \cap C = S_i^\ell$. Then, we have the following equality: $|S_i \setminus P_i| = |S_i^c| = 1$. □

Observation 4.4. Let $(P_i^c, P_i^\ell) = (S_i^c \cap C, S_i^\ell \cap C)$ and $P_i^c \neq \emptyset$, then $(S_i \setminus P_i) \subseteq S_i^\ell$.

Lemma 4.5. Let $P_i = S_i \cap C$. Then, the set $S_i \setminus P_i$ cannot contain both the center vertex and a leaf vertex of S_i.

Proof. We know that $(S_i \setminus P_i) \subseteq (V(G) \setminus C)$, thus $S_i \setminus P_i$ is an independent set. If $S_i \setminus P_i$ contained both the center and a leaf vertex, then there would be an edge that is not covered by C. □

As we can see, there are two main cases that can occur for $P_i = S_i \cap C$. If P_i does not contain the center S_i^c, then P_i contains all leaf vertices S_i^ℓ and we have some candidates $v \in V \setminus C$ that can be the center (v has to be adjacent to all leaf vertices). Another case that can occur is P_i contains the center vertex S_i^c and some leaf vertices from S_i^ℓ, then a vertex $v \in V \setminus C$ can be added to a partial star P_i only if v is not adjacent to any leaf vertex and v is adjacent to the center. With this, we are prepared formally to describe the algorithm.

4.4 The algorithm

Let $G = (V, E)$ be a graph without isolated vertices on n vertices, C be its vertex cover of size k and q be the star partition number of G. Let $C = \{v_{n-k+1}, \ldots, v_n\}$ be the last k vertices of $V(G)$, meaning $(V(G) \setminus C) = \{v_1, \ldots, v_{n-k}\}$.

If $q \geq k$, then construct and return a solution as described in the proof of Theorem 4.1.

Otherwise exhaustively try all sets $P = (P_1, \ldots, P_q)$ that satisfy the following conditions:

1. P is a partition of C, meaning:
   a) $\forall i, j \in [q] : i \neq j \implies P_i \cap P_j = \emptyset$,
   b) $\bigcup_{i=1}^{q} P_i = C$,
   c) $\emptyset \notin P$.
2. $\forall i \in [q] : P_i$ induces a star or is an independent set, meaning:
   a) $P_i = P_i^c \cup P_i^\ell$ and $P_i^c \cap P_i^\ell = \emptyset$,
   b) P_i^ℓ is an independent set,
   c) $|P_i^c| \leq 1$
   d) $P_i^c \neq \emptyset \implies P_i^\ell \subseteq N_G(P_i^c)$.

Without loss of generality, let the first h sets of P be sets that have $P_i^c = \emptyset$ and for all $i > h$ let $P_i^c \neq \emptyset$. Then, with the given P, construct a bipartite graph $\mathcal{B}(P) = (A \cup B, E')$ where

- $A = \{a_1, \ldots, a_{n-k}\} = V(G) \setminus C = \{v_1, \ldots, v_{n-k}\}$,
- $B = \{b_1, \ldots, b_{h+1}\}$,
- $\{a_i, b_j\} \in E(\mathcal{B}(P))$ if and only if

– $j \in [h]$ and $P_j \subseteq N_G(v_i)$, or

– $j = h + 1$ and there exists a set $P_{j'} \in P$ for $j' > h$ such that $(\widehat{P}^c_{j'}, \widehat{P}^\ell_{j'}) = (P^c_{j'}, P^\ell_{j'} \cup \{v_i\})$ induces a star, meaning $v_j \in N_G(P^c_j)$ and $P^\ell_j \cup \{v_i\}$ is and independent set.

If $|B| \geq |A|$ then repeat with another set P'. The current partitioning P of C cannot be extended to a solution.

For a given bipartite graph $\mathcal{B}(P) = (A \cup B, E')$ construct a flow network $(\mathcal{N}, s, t, c)$ the following way:

1. orient all edges from A to B,
2. add a source vertex s and add directed edges (s, a) for every $a \in A$,
3. add a target vertex t and add directed edges (b, t) for every $b \in B$.
4. set the capacity of $(x, y) \in E(\mathcal{N})$ as

$$c(x,y) = \begin{cases} |A| - |B| + 1 & x = b_{h+1}, y = t \\ 1 & \text{otherwise} \end{cases}$$

Then, find a integer maximum flow f in $\mathcal{N}$. If $|f| < n - k$, then repeat with another P'. Otherwise that a solution exists.

The solution $S = (S_1, \ldots, S_q)$ is constructed as follows:

1. for $j \in [h] : (S^c_j, S^\ell_j) = (\{v_i\}, P^\ell_j)$ where $f(a_i, b_j) = 1$,
2. for $j > h : (S^c_j, S^\ell_j) = (P^c_j, P^\ell_j \cup X_j)$, where $X_j = \{v_i \in V(G) \setminus C \mid f(a_i, b_{h+1}) = 1\}$.

Note that for a_i there can be more than one X_j that can contain a_i. In this case leave a_i in exactly one set X_j and it does not matter in which one.

4.4.1 Intuition for the algorithm

The intuition of the algorithm is as follows: Once we know how to partition the vertex cover, we try to assign all other vertices from $V(G) \setminus C$ to some partial star P_i. Each set P_i is either missing a center vertex or can accept new leaf vertices, but both types of vertices (center or leaf vertex) cannot be added to P_i at the same time (refer to Lemma 4.5). On one hand, each set P_i that still does not have a center must acquire a new center vertex from $V \setminus C$ to complete a star. On the other hand, if $P^c_i \neq \emptyset$, then P_i can accept new leaf vertices, but it also does not have to accept any. We want to capture these constraints and match up all vertices in $V \setminus C$ to some P_i to create a partitioning and each P_i either accepts 1 vertex (center vertex) or an unbounded amount (new leaf vertices).

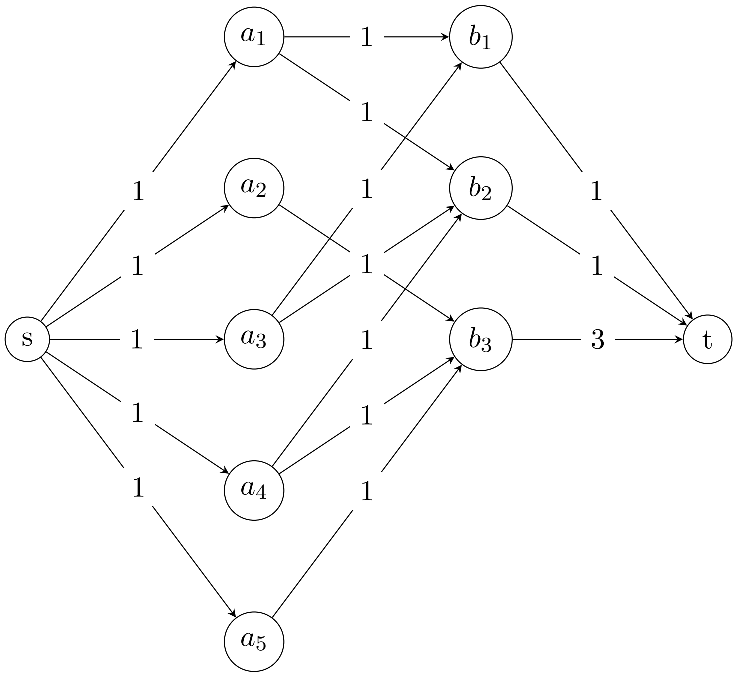

Figure 4.3: Example of a flow network $(\mathcal{N}, s, t, c)$ constructed using a bipartite graph $\mathcal{B}(P)$, where P is a partition of C and $|P| \geq 3$. Vertices $v_1, \ldots, v_5$ are not part of the given vertex cover and need to be assigned to a set P_j. Vertex b_3 is the dummy vertex that represents all sets P_j for $2 < j \leq q$ that have a defined center. Vertices b_1 and b_2 represent sets P_1 and P_2, respectively, and both P_1 and P_2 do not have a defined center. The numbers on edges denote the capacity $c(e)$.

We then model the constraints as an assignment problem. The capacity $c(s, a_i) = 1$ models that a_i can be assigned to at most one set P_j. For $j \in [h]$ we have the following edges: The edges (a_i, b_j) of the network model that v_i can be added to a partial star P'_j, the capacities on edges (b_j, t) set a bound on how many vertices P_j can accept. The special vertex b_{h+1} is a dummy vertex that encapsulates all the sets P_j that can accept new leaf vertices. The capacity $c(b_{h+1}, t)$ allows us to model that at most $|A| - h$ vertices can become new leaf vertices and h vertices must become centers.

An example of a network is shown in Figure 4.3.

4.5 Proof of correctness

In this section, we provide proof of correctness of the algorithm proposed in Section 4.4.

Theorem 4.3. Let G be an input graph without isolated vertices on n vertices, C be a vertex cover of G of size k and q be the induced star partition number of G. Let $P = (P_1, \ldots, P_q)$ be a partition of C. If the weight of a maximum integer flow f in the constructed network $(\mathcal{N}, s, t, c)$ equals $|f| = n - k$, then G admits an induced star partition of size q.

Proof. The construction of a solution S was described in Section 4.4. We first prove that S is a partitioning, then we prove that each set S_i induces a star.

To prove that S is a partition, we first show that $\emptyset \notin S$: This is true as for each $j \in [q]$ it holds that $P_j \neq \emptyset$ (P is a partition) and $P_j \subseteq S_j$, thus $S_j \neq \emptyset$. Now we show that each $v \in V(G)$ is included in exactly one set S_j.

For each $v \in C$, there indeed exists exactly one set P_j that contains v, because we assumed that P is a partitioning of C. Each S_j extends P_j with vertices $V(G) \setminus C$, thus v is exactly in one set S_j.

Let f be an integer flow with weight $|f| = n - k$. The source vertex s has $|A| = n - k$ outgoing edges, each with capacity $c(s, a_i) = 1$, thus $f(s, a_i) = 1$. Each vertex $a_i \in A$ has exactly one incomming edge with flow $f(s, a_i) = 1$, therefore there is exactly one outgoing edge with flow $f(a_i, b_j) = 1$ (we assumed f to be an integer flow). This implies that $v_i \in V \setminus C$ is included at least in one set S_j. If $j \leq h$, then only the set S_j contains $v_i \in V(G) \setminus C$. Else if $j = h + 1$, then we made sure that v_i is in exactly one set $S_{j'}$ such that $h < j' \leq q$. We can conclude that each vertex $v \in V(G) \setminus C$ is included in exactly one set S_j. Thus S is partition of $V(G)$.

Now we show that S_j induces a star for all $i \in [q]$. We know that that the sum of capacities of edges incomming to the target vertex t in the network is $h \cdot 1 + (|A| - |B| + 1) = h + ((n - k) - (h + 1) + 1) = n - k$. We assumed that $|f| = n - k$, which implies that $f(b_j, t) = c(b_j, t)$ for all $b_j \in B$.

First consider sets S_j for $j \in [h]$. Vertex $b_j \in B$ has exactly 1 outgoing edge with capacity $c(b_j, t) = 1 = f(b_j, t)$, therefore there can be only one incomming edge with flow $f(a_i, b_j) = 1$. We assumed for each $j \in [h]$ it holds that $P_j^c = \emptyset$ and the edge (a_i, b_j) is added only if $P_j \subseteq N_G(a_i)$. This implies that $P_j \cup \{a_i\}$ induces a star: vertex a_i is the center and P_j is the set of leaves of the star.

Otherwise assume that $h < j \leq q$. The set P_j by assumption induces a star and $P_j^c \neq \emptyset$. In the construction of S_j, we added X_j as new leaf vertices to P_j. For each pair $u, v \in S_j = P_j \cup X_j$ we prove the following two cases:

$u, v \in X_j$: The set $V(G) \setminus C$ is an independent set, therefore $X_j \subseteq (V(G) \setminus C)$ is also independent and u, v are not adjacent.

$u \in X_j, v \in P_j$: Let u be represented by a_i in the network. The edge (a_i, b_{h+1}) is present in the network only if $P_i \cup \{u\}$ still induced a star.

To sum it up, the constructed S is indeed a partitioning of $V(G)$ and each set $S_i \in S$ also induces a star. Therefore what we constructed in the algorithm is an induced star partition of size q. □

Theorem 4.4. Let G be an input graph without isolated vertices on n vertices, C be a vertex cover of G of size k and q be the induced star partitioning number of G. If G admits an induced star partition of size q, then there is a partition $P = (P_1, \ldots, P_q)$ of C, such that the value of the maximum integer flow f in the constructed network $(\mathcal{N}, s, t, c)$ equals $|f| = n - k$.

Proof. Assuming that G can be partitioned into q stars, then using Lemma 4.3, let us have a partition S' of V such that each $S'_i \in S$ induces a star and $S'_i \cap C \neq \emptyset$. Let S be a permutation of S' such that for the first h sets it holds that $S^c_i \cap C \neq \emptyset$ and for i such that $h < i \leq q$ it holds that $S^c_i \cap C = \emptyset$.

For each $i \in [q]$ we set $P_i = S_i \cap C$. Each vertex $v \in C$ is included in exactly one set S_i, thus $P = (P_1, \ldots, P_q)$ is a partitioning of C. Each $P_i \in P$ is trivially an independent set or induces a star as $P_i \subseteq S_i$. This implies that P is a valid partitioning of C and we can construct $\mathcal{B}(P)$ and flow network $(\mathcal{N}, s, t, c)$ as described in Section 4.4. The weight of the maximum flow through the network is at most $n - k$ because $\sum_{(s,x) \in E(\mathcal{N})} c(s, e) = |A| = n - k$. Therefore for all flows f it holds that $|f| \leq n - k$.

Now we show that an integer flow f with weight $|f| = n - k$ can be constructed. First, we set $f(e) = 0$ for all $e \in E(\mathcal{N})$. Then, for each $v_i \in V(G) \setminus C$ we use the relation $v_i \in S_j$ to increase the flow along the path s, a_i, y, t by 1 (meaning we set $f(e) = f(e) + 1$). If $j \leq h$, then we use $y = b_j$, else if $h < j \leq q$, then we use $y = b_{h+1}$. The set A is of size $|V(G) \setminus C| = n - k$, therefore $|f| = n - vc$.

We now verify that the described function f is a flow. First we prove that the path s, a_i, y, t exists. The edges (s, a_i) and (y, t) trivially exist. The set S_j that contains v_i induces a star, therefore we distinguish two cases, v_i is either the center or a leaf vertex.

$v_i \in S^c_j$: Vertex v_i is not part of the vertex cover, therefore $N_G(v_i) \subseteq C$. This implies that $P^c_i = S^c_i \cap C = \emptyset$ and $j \leq h$. Vertex v_i therefore is one of the considered candidates for P_j and the edge (a_i, b_j) indeed exists.

$v_i \in S^\ell_j$: We know that all leaf vertices in S^ℓ_j are adjacent to the center. Vertex v_i is not part of the vertex cover, therefore the center has to be part of the vertex cover and it holds that $h < j \leq p$. The set $P_j \cup \{v_i\}$ is just subset of S_j without some leaf vertices, therefore $P_j \cup \{v_i\}$ also induces a star. We can conclude that the edge (a_i, b_{h+1}) indeed exists.

Finally we show that $f(e) \leq c(e)$ for all edges in the network. The set S is a partitioning of $V(G)$, therefore for each vertex $v_i \in V(G) \setminus C$, there is exactly one set S_j such that $v_i \in S_j$. This implies that the flow going through $a_i \in A$ is exactly 1 and $f(e) \leq c(e)$ for all $e \in E(\mathcal{N})$ that have a_i as one of its endpoint. For $j \leq h$ we use Lemma 4.4 to deduce $|S_i \setminus P_i| = 1$, therefore the flow going through b_j is exactly 1. We know that h vertices from $V(G) \setminus C$ are center vertices, thus the other $|A| - h$ vertices are leaf vertices and the flow was increased along a path that contained b_{h+1}. This implies that $f(b_{h+1}, t) = |A| - h = |A| - (|B| - 1) = c(b_{h+1}, t)$.

We conclude that f is indeed a flow, all values $f(e)$ are integers and also $|f| = n - k$. $\square$

Theorem 4.5. Let G be an input graph on n vertices, C be a vertex cover of G of size k and q be the induced star partitioning number of G. Then the Induced Star Partition problem on G can be solved in $O(O(k^{2k+1}n^2))$ time.

Proof. We first reduce the instance (G, q) using Reduction rule 1 to remove isolated vertices in $O(n)$ time. Let us assume that G is without isolated vertices. If $q \geq k$ then we construct a solution in $O(n)$ time as described in the proof of Theorem 4.1 or else if $q < k$, then we try out all partitionings P. In the latter case, we try at most k^{2k} sets P, each vertex is either a center or a leaf vertex in one of $q < k$ sets. We construct the adjacency matrix to check if two vertices in G are adjacent in $O(1)$ time.

Not all partitions P can be used to construct a bipartite graph $\mathcal{B}(P)$, thus we have to check if each P_i is an independent set or induces a star. For each $P_j \in P$ we check if P_j^ℓ is an independent set in $O(|P_j|^2)$ time. Furthermore, if $P_j^c \neq \emptyset$ then we check in $O(|P_j|)$ time if the leaves are adjacent to the center. In total, checking each P_i takes $O(|P_j|^2)$ time and checking P takes $O(\sum_{j=1}^{q} |P_j|^2) = O(|P|^2) = O(k^2)$ time.

Now we analyze the construction of $\mathcal{B}(P)$. For each $j \in [h]$, finding all centers that can be added to P_j can be done in $f_j = O(|P_j| \cdot n)$ time: for each $u \in P_j^\ell$ we mark all of its neighbours, if there is a vertex $v_i \in V \setminus C$ that was marked $|P_j|$ times, then we add the edge (a_i, b_j) into the constructed bipartite graph. Across all P_j with $j \in [h]$, we get the running time

$$\sum_{j=1}^{h} f_j = O\left(n \cdot \sum_{j=1}^{h} |P_j|\right). \tag{4.1}$$

For each $h < j \leq q$, we try to insert $a_i \in V(G) \setminus C$ into P_j and check if $P_j \cup \{a_i\}$ induces a star in $f_j' = O(|P_j|)$ time: we check that v_i is not adjacent to any leaf vertex in $O(|P_j|)$ time and then check if v_i is adjacent to the center in $O(1)$ time. Across all P_j with $h < j \leq q$, we get the running time

$$\sum_{j=h+1}^{q} \left(|A| \cdot f_j'\right) = O\left(n \sum_{j=h+1}^{q} |P_j|\right). \tag{4.2}$$

The set P is a partitioning of C, therefore $\sum_{j=1}^{q} |P_j| = |C| = k$ and we can conclude that the edges of $\mathcal{B}(P)$ can be constructed in $O(kn)$ time.

The construction of the network can be done in linear time with regards to the size of $\mathcal{B}(P)$: we copy and orient edges in $\mathcal{B}(P)$, then add two new vertices and $O(|A| + |B|) = O(n)$ new edges. The number of edges in $\mathcal{B}(P)$ is at most $|A| \cdot |B| = O(kn)$.

Then, finding the maximum flow in the network can be done in $O(|A|^2 \cdot |B|)$ time. We use Edmons-Karp's algorithm [15] to construct a maximum integer flow. If $|E(\mathcal{B}(P))| < |A|$ then we know that a desired flow with weight $|f| = n - k$ cannot be constructed and we can safely assume that $|E(\mathcal{B}(P))| \geq |A|$. We know that for any flow f it holds that $|f| \leq |A|$, thus the number of iterations is at most $O(|A|)$ because the algorithm improves the flow at least by 1 in each iteration. Each iteration of Edmons-Karp's algorithm consists of finding a shortest augmenting path in $O(|V(\mathcal{B}(P))| + |E(\mathcal{B}(P))|)$ time and then modifying the flow on edges in $O(|E(\mathcal{B}(P))|)$ time, in total $O(|E(\mathcal{B}(P))|) = O(|A| \cdot |B|)$ time.

The construction of the solution S can be done in $O(kn)$ time: We copy P in $O(n)$ time, then for each $a_i \in A$ we find the edge with flow $f(a_i, b_j) = 1$ in $O(\deg_{\mathcal{B}(P)}(a_i)) = O(|B|) = O(k)$ time and add the vertex to the corresponding set.

The total running time then consists of trying all k^{2k} partitions, for each P we

1. check if P is valid in $O(k^2)$ time,
2. construct $\mathcal{B}(P)$ in $O(kn)$ time,
3. construct the network in $O(kn)$ time,
4. find max flow in the network in $O(|A|^2 \cdot |B|) = O(n^2 k)$ time.
5. construct the solution in $O(kn)$ time.

Thus the running time of the algorithm is $O(k^{2k+1} n^2)$ time. □

4.5.1 Branch-and-bound method

Part of the algorithm described in Section 4.4 has to generate all partitions P of C. In our implementation, we used a branch-and-bound method to gradually generate all possible sets P. We start with a set $P = (P_1, \ldots, P_q) = (\emptyset, \ldots, \emptyset)$ with q empty sets. Then, we iterate over the vertices in C an insert one vertex $c \in C$ into P at a time. Assume that the first j sets of P are not empty and for all $j < i \leq q$ it holds that $P_i = \emptyset$. We can try to insert the vertex $c \in C$ either (1) into some nonempty set P_i as center or as a leaf vertex or (2) into an empty set P_{j+1} as new center or as a leaf vertex. In total, the number of partitions is still $O(k^{2k})$, but the described recursive method of generating P allows us to implement some branch-cutting optimizations to remove some recursive branches that generate invalid partitionings P. The following optimizations were used in the implementation to cut of some recursive branches:

- If we insert c as a leaf vertex into P_i^ℓ, then we need to make sure that
  - $P_i^\ell \cup \{c\}$ is an independent set: meaning we check that c is not adjacent to any vertex $u \in P_i^\ell$;
  - the distance between c and each vertex $u \in P_i^\ell$ is exactly 2;
  - if $P_i^c \neq \emptyset$, then we check that c is adjacent to $v \in P_i^c$.
- If we insert c as a center vertex into P_i^c, then we check that c is adjacent to all vertices P_i^ℓ.

Chapter **5**

Algorithm on graphs with bounded treewidth

In this chapter, we prove that there exists a linear time algorithm for the INDUCED STAR PARTITION problem on graphs with bounded treewidth. We will first show that there exists a MSO_2 formulation for this problem and as such, we can use Courcelle's theorem [11] to show the existence of such an algorithm. Then we give an explicit dynamic programming algorithm on the tree decomposition of the graph, which can compute the induced star partition number in $O(k^{2k} \cdot n)$ time, where k is the treewidth of the graph and n is the number of vertices.

5.1 MSO2 formulation

Theorem 5.1. For every fixed q, there is an algorithm that decides whether the input graph G on n vertices, given with its tree decomposition, can be partitioned into q induced stars in $f(k) \cdot n$ time, where f is some computable function.

Proof. First, we formulate that a graph H is isomorphic to a star, meaning $H \approx K_{1,r}$ for some $r \geq 0$, using MSO_2. The intuition is as follows: we want to show, that in H there is a vertex u, such that u is adjacent to every other vertex $V(H) \setminus \{u\}$. This special vertex u is the center of the star. Following that, we want that every pair of vertices in $V(H) \setminus \{u\}$ to not be adjacent. This implies that $V(H) \setminus \{u\}$ is an independent set. These two properties are enough to fully characterize a star $K_{1,r}$.

$$\begin{aligned} \mathit{Star}(V) \equiv & (\exists u \in V)((\forall v \in V)(u \neq v \implies \mathit{adj}(u, v)) \\ & \wedge (\forall v, w \in V)(v \neq w \wedge v \neq u \wedge w \neq u \implies \neg \mathit{adj}(v, w))) \end{aligned} \tag{5.1}$$

We want to partition all vertices of the input graph G into sets of vertices so that sets are pairwise disjoint, the union of sets contains all vertices of the graph, and each set induces a star.

$$\begin{aligned}(\exists S_1, S_2, \dots S_q \subseteq V)(disjoint(S_1, S_2, \dots S_q)) \wedge (union(S_1, S_2, \dots, S_q)) \\ \implies S(S_1) \wedge S(S_2) \wedge \dots \wedge S(S_q)\end{aligned} \tag{5.2}$$

The disjoint function $disjoint(S_1, S_2, \dots S_q)$ expands to

$$(S_1 \cap S_2 = \emptyset) \wedge (S_1 \cap S_3 = \emptyset) \wedge \dots \wedge (S_{q-1} \cap S_q = \emptyset) \tag{5.3}$$

and defines sets to be pairwise disjoint for all pairs $i, j \in [q]$ such that $i < j$.

And if we want to be precise, then

$$S_i \cap S_j = \emptyset \equiv (\forall u \in V)(u \in S_i \iff u \notin S_j) \tag{5.4}$$

The function $union(S_1, S_2, \dots, S_q)$ expands to

$$(\forall v \in V)(v \in S_1 \vee v \in S_2 \vee \dots \vee v \in S_q) \tag{5.5}$$

The $S(S_i)$ function simply takes Equation 5.1 and replaces all occurrences of $x \in V$ with $x \in S_i$.

We showed that the Induced Star Partition name can be formulated using MSO_2 when q is fixed. Then, using Courcelle's theorem [11] we know that there exists a linear time algorithm for graphs with bounded treewidth if a tree decomposition is given together with the graph. □

5.1.1 Beyond tree width

In Theorem 5.1 we proved that there exists a fixed parameter tractable algorithm for the Induced Star Partition when parameterized by the treewidth. But from the MSO_2 formulation that we provided, we can notice that we did not use quantification over subsets of edges. Using the same formulation in the proof and with Courcelle's theorem [12], we also acquire that the problem is FPT when parameterized by the cliquewidth of the input graph when the sequence of operations to construct the graph is given.

5.2 Dynamic programming on tree decomposition

The algorithm that follows from Courcelle's theorem is FPT when parameterized by treewidth, but the hidden constants are prohibitive. In this section, we provide an explicit dynamic programming algorithm on graphs with bounded treewidth with reasonable computation time.

Let T be a nice tree decomposition of graph G with at most $O(k \cdot n)$ nodes, where $|V(G)| = n$ and k is the treewidth of G. In the algorithm, we will be filling a dynamic programming table C for each $t \in V(T)$ and valid partitioning P of X_t. We first define what a valid partitioning is.

Definition 5.1. For every $t \in V(T)$ we call $P = (P_1, P_2, \ldots, P_p)$ a partitioning of X_t if and only if the following conditions hold:

1. $\forall i, j \in [p] : i \neq j \implies P_i \cap P_j = \emptyset$,
2. $\bigcup_{i=1}^{p} P_i = X_t$,
3. $\emptyset \notin P$.

We will further partition each P_i into three sets: $P_i = P_i^1 \cup P_i^2 \cup P_i^3$ and $P_i^h \cap P_i^{h'} = \emptyset$ for $h \neq h'$.

Note that we require P_i to be nonempty, but we allow each individual P_i^h to be empty. Just all three subsets of P_i cannot be empty at the same time.

Definition 5.2. We call a partitioning P VALID with respect to X_t if and only if these following conditions hold:

a) Each set P_i is of one of the following types

T0 $|P_i^1| = 1 \wedge P_i^2 \neq \emptyset \wedge P_i^3 = \emptyset$,

T1 $|P_i^1| = 1 \wedge P_i^2 = \emptyset \wedge P_i^3 = \emptyset$,

T2 $|P_i^1| = 0 \wedge P_i^2 \neq \emptyset \wedge P_i^3 = \emptyset$,

T3 $|P_i^1| = 0 \wedge P_i^2 = \emptyset \wedge P_i^3 \neq \emptyset$.

b) Additionally, if P_i if of type

T0 then $P_i^2 \subseteq N_{G_t}(P_i^1)$ and P_i^2 is an independent set;

T2 then P_i^2 is an independent set;

T3 then P_i^3 is an independent set.

The partition P in some way will represent how an induced star partitioning has to intersect the bag X_t and subsequently the subgraph G_t. Some stars will have a nonempty intersection with X_t, some stars will intersect V_t but not X_t, and some stars are yet to be discovered—those that intersect $V \setminus V_t$. Each set P_i prescribes how the final partitioning should intersect the bag X_t and the dynamic programming table will help us store the stars that have been processed but no longer intersect X_t.

There are three cases that we distinguish. The intersection can contain: the center of the star and some leaf vertices vertex (T0), the center of the star only (T1), or the leaves of the star without the center (T2 and T3).

The set P_i^1 contains the center of the star and the sets P_i^2 and P_i^3 contain the leaves of the star. We explicitly define these two types of sets for leaves and give them a special meaning: The set P_i^2 contains vertices that have already „seen“ the center of the star in the graph G_t and conversely P_i^3 contains vertices that have yet to see the center of the star (meaning the center is in

$V \setminus V_t$). With this intention in mind, we want one of the sets P_i^2 or P_i^3 to be empty. We cannot have leaf vertices of the same star have seen the center and wait for the center to be discovered at the same time.

If $|P_i^1| \geq 2$, then such a partitioning is invalid. We have too many vertices as center of a star.

If $|P_i^1| = 1$, then the star has a center and all leaves can „see" the center, thus leaves vertices are in P_i^2. Additionally, we require that $P_i^3 = \emptyset$—we cannot have vertices that have yet to discover the center when the center is included in the bag.

If $|P_i^1| = 0$, then we need to have the information, whether the center has been forgotten or is yet to be discovered in the algorithm. This is why we introduced 2 types of sets for the leaves, P_i^2 and P_i^3. If the set P_i^2 is not empty, then we can be sure that the center is either in the P_i^1 or has been forgotten. Conversely if the set P_i^3 is not empty, then we can be sure that the center is not present in V_t and is yet to be discovered.

For a valid partitioning P of X_t, we want to construct the minimum compatible partial solution for t and P.

Definition 5.3. Let t be a node from the tree decomposition and P a valid partitioning of X_t of size p. We call $S = (S_1, \ldots, S_s)$ a partial solution compatible with P at t if and only if these following conditions hold:

a) S is a partitioning of V_t

   i $\forall i, j \in [s] : i \neq j \implies S_i \cap S_j = \emptyset$,

   ii $\bigcup_{i=1}^{s} S_i = V_t$,

   iii $\emptyset \notin S$;

b) $\forall i \in [s] : S_i = S_i^c \cup S_i^\ell$ and S_i is either isomorphic to a star or is an independent set:

   i S_i^ℓ is an independent set in G_t,

   ii $|S_i^c| \leq 1$ and if S_i^c is not empty, then also $S_i^\ell \subseteq N_{G_t}(S_i^c)$;

c) $\forall i \in [p] : S_i \cap X_t = P_i$ and furthermore if P_i is of type:

   T0 then $P_i^1 = S_i^c \wedge P_i^2 = S_i^\ell \cap X_t$,

   T1 then $P_i^1 = S_i^c \wedge S_i^\ell \cap X_t = \emptyset$,

   T2 then $S_i^c \neq \emptyset \wedge S_i^c \cap X_t = \emptyset \wedge P_i^2 = S_i^\ell \cap X_t$,

   T3 then $S_i^c = \emptyset \wedge P_i^3 = S_i^\ell$;

d) $\forall j > p$:

   - $S_j \subseteq V_t \setminus X_t$,

- $|S_j^c| = 1$.

The partial solution S compatible with P at t represents how the star partitioning should look like for the subgraph G_t, while having some of the stars (the ones intersecting X_t) as prescribed by P. We call the sets S_j that have an empty intersection with the bag X_t *forgotten* and they should induce a star because no other vertices can join them in the future due to the definition of tree decomposition. The center of the star is stored in S_i^c, while S_i^ℓ contains the leaves of the star.

An example of how a partitioning P of X_t could look like and the implication for S compatible with P is shown in Figure 5.1.

Now we can finally describe the algorithm. We will be filling a dynamic programming table $C[\cdot,\cdot]$ in a bottom up manner on T for each $t \in V(T)$ and each valid partitioning P of X_t. The value $C[t,P]$ is defined as the minimum number of forgotten stars in a partial solution S compatible for (t,P) on the graph G_t. The leaf nodes create the base case and for every non-leaf node, $C[t,P]$ will compute its values from its children. The result for the whole graph G is then stored in the root node at $C[r,\emptyset]$.

5.2.1 Leaf node

If t is a leaf node, then we set $C[t,\emptyset] = 0$.

5.2.2 Introduce node

Let t be an introduce node with child t' such that $X_t = X_{t'} \cup \{v\}$. Assume also that P is a valid partitioning of X_t. The set P is a partitioning, thus the new vertex v can be in exactly 1 set P_i. We compute $C[t,P]$ as follows:

$$C[t,P] = \begin{cases} +\infty & v \in P_i^2 \wedge |P_i^1| = 0 \\ C[t',\widehat{P}] & \text{otherwise} \end{cases} \tag{5.6}$$

The construction of $\widehat{P}$ depends on whether v is in P_i^1, P_i^2 or P_i^3 and the size of P_i.

1. If $|P_i| \geq 2$, then for $j \in [p]$ we compute

$$(\widehat{P}_j^1, \widehat{P}_j^2, \widehat{P}_j^3) = \begin{cases} (P_j^1, P_j^2, P_j^3) & j \neq i \\ (\emptyset, \emptyset, P_i^2) & j = i \wedge v \in P_i^1 \wedge P_i^2 \neq \emptyset \\ (P_i^1, P_i^2 \setminus \{v\}, \emptyset) & j = i \wedge v \in P_i^2 \wedge |P_i^1| = 1 \\ (\emptyset, \emptyset, P_i^3 \setminus \{v\}) & j = i \wedge \{v\} \subsetneq P_i^3 \end{cases} \tag{5.7}$$

2. If $(v \in P_i^1 \wedge P_i^2 = \emptyset)$ or $(\{v\} = P_i^3)$, then for $j \in [p-1]$ we compute

$$(\widehat{P}_j^1, \widehat{P}_j^2, \widehat{P}_j^3) = \begin{cases} (P_j^1, P_j^2, P_j^3) & j < i \\ (P_{j+1}^1, P_{j+1}^2, P_{j+1}^3) & j \geq i \end{cases} \tag{5.8}$$

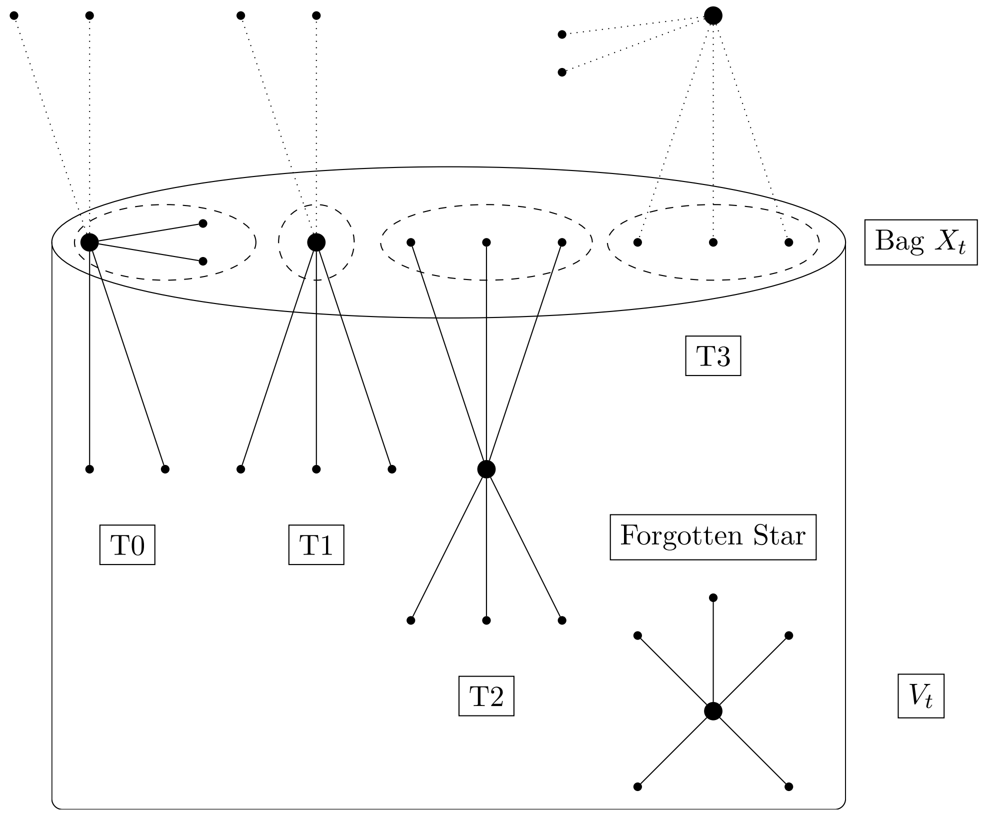

Figure 5.1: A partition $P = (P_1, P_2, P_3, P_4)$ of a bag X_t. Set P_1, P_2, P_3, P_4 are of type T0, T1, T2, T3, respectively. Let S_i be compatible set for P_i at t for $i \in [4]$. Then, S_1 has the center and some leaf vertices in X_t, S_2 compatible for P_2 at t has only the center in the bag, S_3 has some leaves in the bag and the center has been forgotten, S_4 has some leaves in the bag and the center has yet to be discovered. Vertices below the bag are forgotten, vertices above the bag are yet to be discovered. Smaller dots represent leaf vertices, bigger dots represent the center. Dotted lines indicate edges incident with a vertex that has not been discovered yet, solid lines indicate edges that have been processed. Edges between different stars are not present in the figure.

We now give a brief intuition of the procedure. The vertex v is not present in $G_{t'}$ and is newly introduced in t.

If $v \in P_i^1$, then the leaf vertices can „see“ the center in G_t and the leaf vertices are in P_i^2. Conversely, the leaf vertices cannot see the center in $G_{t'}$, thus we require the leaf vertices to be in $\widehat{P}_i^3$ in $G_{t'}$.

Else if $v \in P_i^2$, then we introduced a new leaf vertex that is adjacent to the center, therefore we require the center to also be present in the bag X_t.

Finally if $v \in P_i^3$, then we introduced a new leaf vertex that requires that the center of the star is yet to be discovered.

We also do not want to include an empty set in $\widehat{P}$ as an invariant. For this reason we create $\widehat{P}$ of size $p-1$ in Equation 5.8 if $P_i \setminus \{v\}$ would be empty.

5.2.3 Forget node

Let t be a forget node with child t', such that $X_t = X_{t'} \setminus \{v\}$. Assume also that P is a valid partitioning of X_t. We try all positions, where v could have been before it was forgotten and choose the optimal configuration. In t' the vertex v could have been part of a partition that still exists in P or it was the last vertex of a forgotten star in t. Let $\widehat{\mathscr{P}}(P)$ be a family of all sets, where v was part of an existing set, and $\widetilde{\mathscr{P}}(P)$ be a family of sets, where v is the lone vertex of a partition. We compute $C[t,P]$ as follows:

$$C[t,P] = \min\left\{ \min_{\widehat{P}\in\widehat{\mathscr{P}}(P)} C[t',\widehat{P}], 1 + \min_{\widetilde{P}\in\widetilde{\mathscr{P}}(P)} C[t',\widetilde{P}] \right\} \tag{5.9}$$

Now we describe the family of sets $\widehat{\mathscr{P}}(P)$. Let $A = \bigcup_{g=1}^{p} \left\{\widehat{\mathscr{P}}_g^1(P), \widehat{\mathscr{P}}_g^2(P)\right\}$ be a family of sets, where the element $\widehat{\mathscr{P}}_g^\ell(P) = (\widehat{P}_1, \dots, \widehat{P}_p)$ is a partitioning of $X_{t'}$ and each $\widehat{P}_j$ was computed as:

$$(\widehat{P}_j^1, \widehat{P}_j^2, \widehat{P}_j^3) = \begin{cases} (P_j^1, P_j^2, P_j^3) & j \neq g \\ (P_g^1 \cup \{v\}, P_g^2, P_g^3) & j = g \land \ell = 1 \\ (P_g^1, P_g^2 \cup \{v\}, P_g^3) & j = g \land \ell = 2 \end{cases} \tag{5.10}$$

The partitioning $\widehat{\mathscr{P}}_j^\ell(P)$ contains v within one of its p existing partitions, furthermore v could have been in $\widehat{P}_j^1$ as a center of a star or as a leaf of a star. We used the index ℓ to emphasize these two options. Then, we create $\widehat{\mathscr{P}}(P) = \left\{\widehat{\mathscr{P}}_g^\ell(P) \in A \mid \widehat{\mathscr{P}}_g^\ell(P) \text{ is valid}\right\}$ which filters out invalid partitionings of $X_{t'}$.

Finally we describe $\widetilde{\mathscr{P}}(P)$. First, we introduce $B = \{\widetilde{\mathscr{P}}^1(P), \widetilde{\mathscr{P}}^2(P)\}$, which has only two elements. The partitioning $\widetilde{\mathscr{P}}^\ell(P) = (\widetilde{P}_1, \dots, \widetilde{P}_{p+1})$ was created with the following semantics:

$$(\widetilde{P}_j^1, \widetilde{P}_j^2, \widetilde{P}_j^3) = \begin{cases} (P_j^1, P_j^2, P_j^3) & i \leq p \\ (\{v\}, \emptyset, \emptyset) & j = p+1 \wedge r = 1 \\ (\emptyset, \{v\}, \emptyset) & j = p+1 \wedge r = 2 \end{cases} \tag{5.11}$$

Then we create $\widetilde{\mathscr{P}}(P) = \left\{ \widetilde{\mathscr{P}}^\ell(P) \in B \mid \widetilde{\mathscr{P}}^\ell(P) \text{ is valid} \right\}$.

We do not try to insert v as a leaf in $\widehat{P}_g^3$ nor into $\widetilde{P}_{p+1}^3$. The vertices in the third set have yet to see the center and any compatible solution for t' would not be compatible with t as the partial star S_i will not be able to induce a star (v will not have an edge with center).

To compute the forget node, we get the minimum from at most $2(p+1)$ values: at most $2p$ values for $\widehat{\mathscr{P}}(P)$ and at most 2 for $\widetilde{\mathscr{P}}(P)$.

5.2.4 Join node

Suppose t is a join node with children $\widehat{t}$ and $\widetilde{t}$, such that $X_t = X_{\widehat{t}} = X_{\widetilde{t}}$. We compute the solution as follows:

$$C[t, P] = \min_{(\widehat{P}, \widetilde{P}) \in \mathscr{P}(P)} \left\{ C[\widehat{t}, \widehat{P}] + C[\widetilde{t}, \widetilde{P}] \right\}. \tag{5.12}$$

We compute the minimum using all pairs $(\widehat{P}, \widetilde{P})$ from family of sets $\mathscr{P}(P)$. Suppose $\mathit{\Gamma} = \{i \in [p] \mid P_i \text{ is of type T2}\}$. Then, for each $I \subseteq \mathit{\Gamma}$ there exists exactly one pair $(\widehat{P}, \widetilde{P}) \in \mathscr{P}(P)$ if and only if:

1. $\forall i \in [p] \setminus \mathit{\Gamma} : \widetilde{P}_i = \widetilde{P}_i = P_i$,
2. $\forall i \in I : (\widehat{P}_i^1, \widehat{P}_i^2, \widehat{P}_i^3) = (\emptyset, P_i^2, \emptyset) \wedge (\widetilde{P}_i^1, \widetilde{P}_i^2, \widetilde{P}_i^3) = (\emptyset, \emptyset, P_i^2)$,
3. $\forall i \in \mathit{\Gamma} \setminus I : (\widehat{P}_i^1, \widehat{P}_i^2, \widehat{P}_i^3) = (\emptyset, \emptyset, P_i^2) \wedge (\widetilde{P}_i^1, \widetilde{P}_i^2, \widetilde{P}_i^3) = (\emptyset, P_i^2, \emptyset)$.

For each $P_i \in P$, such that $P_i = (\emptyset, P_i^2, \emptyset)$ (P_i is of type T2), we do not know, if the subtree $V_{\widehat{t}}$ or $V_{\widetilde{t}}$ contains the forgotten center v. We can not let the partition have 2 centers, as such we try for each such P_i two instances. If $i \in I$, then the center is in $V_{\widehat{t}}$, otherwise the center is supposed to be in $V_{\widetilde{t}}$. All other subsets P_j of type T0, T1 and T3 are unchanged and $(\widehat{P}_j^1, \widehat{P}_j^2, \widehat{P}_j^3) = (\widetilde{P}_j^1, \widetilde{P}_j^2, \widetilde{P}_j^3) = (P_j^1, P_j^2, P_j^3)$.

Note that $\mathit{\Gamma}$ can be empty, subsequently $\mathscr{P}(P) = \{(P, P)\}$. In other words, the only available partitioning of subtrees $\widehat{t}$ and $\widetilde{t}$ is $\widehat{P} = P$ and $\widetilde{P} = P$. This corresponds to $I = \emptyset$.

There are at most p sets in P, therefore the resulting number of combinations we have to try is at most 2^p.

5.3 Proof of correctness

We will now show that the previously described algorithm can correctly return the minimum number of induced star partitions on graph G by returning $C[r,\emptyset]$. We will prove this in two parts. First we show in Theorem 5.2 that the dynamic programming table $C[\cdot,\cdot]$ is filled correctly. Then in Theorem 5.3 we show that the value stored in $C[r,\emptyset]$ represents the solution for the INDUCED STAR PARTITION of the input graph G.

Theorem 5.2. *Let t be a node of a tree decomposition T of graph G and $P=(P_1,\dots,P_p)$ be a valid partitioning of X_t of size p. Then for each (t,P), the algorithm stores into $C[t,P]$ the minimum value h, such that there exists a partial solution $S=(S_1,\dots,S_s)$ compatible for (t,P) and of size $s=p+h$.*

Proof. We remind the reader that each set P_i in a valid partitioning P of X_t can be one of the 4 types introduced in Definition 5.1 and a compatible partial solution has to satisfy the 4 conditions from Definition 5.3.

Leaf node. The subgraph G_t has no vertices, therefore the only valid partitioning is $P=\emptyset$. The algorithm stores $C[t,\emptyset]=0$ and the only compatible set is $S=\emptyset$. Such a partial solution is compatible for (t,P).

Introduce node. Let v be the newly introduced vertex. Assume that $C[t,P]=C[t',\widehat{P}]=h$ is the value computed by the algorithm. By induction hypothesis there is a partial solution $\widehat{S}$ compatible for $(t',\widehat{P})$ of size $\widehat{p}+h$, where $|\widehat{P}|=\widehat{p}$. We show that a partial solution S compatible for (t,P) and of size $s=p+h$ exists and it simply extends $\widehat{S}$.

Let i be the index of the set P_i that contains the newly introduced vertex v. We split the proof based on the size of P_i.

If $|P_i|\geq 2$, then $\widehat{P}_i=P_i\setminus\{v\}$ is not empty. This means that by induction hypothesis there is a set $\widehat{S}_i$ that intersects $X_{t'}$ and equals exactly $\widehat{P}_i$.

First consider all forgotten stars $\widehat{S}_j$ in t'. These stars had an empty intersection with $X_{t'}$ and still have an empty intersection with $X_t=X_{t'}\cup\{v\}$ as $v\notin V_{t'}$. Thus, we set $S_j=\widehat{S}_j$ if $\widehat{S}_j$ is a forgotten star. Then consider all sets $\widehat{S}_j$ that have an intersection $\widehat{P}_j$ with $X_{t'}$. If $v\notin P_j$, then $S=\widehat{S}_j$ is still compatible for $P_j=\widehat{P}_j$. Finally consider the set $\widehat{S}_i$ such that $\widehat{S}_i\cap X_t=\widehat{P}_i=P_i\setminus\{v\}$. We analyze the types of $\widehat{P}_i$ and P_i and then show that $\widehat{S}_i$ can be extended in a simple manner to be compatible with P_i.

There are 3 cases we need to analyze, distinguished by which subset of P_i contains v. First, v could have been part of P_i^1: We assumed that P is valid, thus $|P_i^1|\leq 1$ and the set P_i^2 has to contain another vertex in order to satisfy $|P_i|\geq 2$. Another case we need to consider is that v could have been part of P_i^2—for the combination $|P_i|=0$ and $v\in P_i^2$ the algorithm returns $C[t,P]=+\infty$ (the proof for this case is left for the last part of the introduce node). Therefore we only analyze $v\in P_i^2\wedge|P_i^1|=1$. Finally, v could have been in P_i^3: from validity of P_i we know that $P_i^1=P_i^2=\emptyset$ and P_i^3 has to be of size at least 2.

Now we analyze the above mentioned valid cases. For each case we describe the construction of S_i and then prove that S_i is compatible for P_i at t in three steps: (1) we show that the intersection of S_i with X_t equals exactly P_i, (2) we show that S_i has the correct structure as prescribed by the *type* of P_i, (3) We show that S_i is either a star or an independent set.

$v \in P_i^1 \wedge P_i^2 \neq \emptyset$ **:** The algorithm used $(\widehat{P}_i^1, \widehat{P}_i^2, \widehat{P}_i^3) = (\emptyset, \emptyset, P_i^2)$ as one of the sets in the partitioning $\widehat{P}$ of $X_{t'}$. On one hand, the set $\widehat{P}_i$ is of type T3 and by induction hypothesis it holds that $\widehat{S}_i^c$ is empty and $\widehat{S}_i^\ell = \widehat{P}_i^3 = P_i^2$. On the other hand, P_i is of type T0. We set $(S_i^c, S_i^\ell) = (\{v\}, \widehat{S}_i^\ell)$, meaning we extend $\widehat{S}_i$ with a new center.

First, we show the intersection of S_i with X_t equals exactly P_i: $S_i \cap X_t = (\widehat{S}_i \cup \{v\}) \cap (X_{t'} \cup \{v\}) = (\widehat{S}_i \cap X_{t'}) \cup \{v\} = \widehat{P}_i \cup \{v\} = P_i$.

Second, we know that P_i is of type T0, therefore we need to show that $S_i^c = P_i^1$ and $P_i^2 = S_i^\ell \cap X_t$. The center $v \in S_i^c$ was introduced and is the only vertex in P_i^1 as P_i is valid ($|P_i| \leq 1$) and indeed $P_i^1 = S_i^c$. The set of leaves S_i^ℓ equal P_i^2, thus $S_i^\ell \cap X_t = P_i^2 \cap X_t = P_i^2$.

Finally, we prove that S_i induces a star. By induction hypothesis we know that $S_i^\ell = \widehat{S}_i^\ell = \widehat{P}_i^3 = P_i^2$ is an independent set. Furthermore, all leaf vertices in $S_i^\ell = P_i^2$ are adjacent to u as prescribed by validity of P_i ($P_i^2 \subseteq N_{G_t}(v)$).

$v \in P_i^2 \wedge |P_i^1| = 1$ **:** Then P_i is of type T0 and subsequently $(\widehat{P}_i^1, \widehat{P}_i^2, \widehat{P}_i^3) = (P_i^1, P_i^2 \setminus \{v\}, \emptyset)$ is of type T0 or T1 depending on emptiness of $\widehat{P}_i^2$. In both cases, the set $\widehat{S}_i$ can be extended to $(S_i^c, S_i^\ell) = (\widehat{S}_i^c, \widehat{S}_i^\ell \cup \{v\})$.

We first need to show the intersection of S_i with X_t equals exactly P_i—this was already proven in the previous case.

The set P_i is of type T0 or T1, which means that we need to prove that $P_i^1 = S_i^c$ and $S_i^\ell \cap X_t$ equals P_i^2. From the algorithm, we have that the set $\widehat{P}_i^1$ equals P_i^1 and by induction hypothesis it holds that $\widehat{P}_i^1 = \widehat{S}_i^c$. This proves that $P_i^1 = S_i^c$. If $\widehat{P}_i^2 \neq \emptyset$, then $\widehat{P}_i$ is of type T0 and subsequently $\widehat{S}_i^\ell \cap X_{t'} = \widehat{P}_i^2$. Otherwise $\widehat{P}_i^2 = \emptyset$, then $\widehat{P}_i$ is of type T1 and by induction hypothesis we know that $\widehat{S}_i^\ell \cap X_{t'} = \emptyset = \widehat{P}_i^2$. Then $S_i^\ell \cap X_t = (\widehat{S}_i^\ell \cup \{v\}) \cap (X_{t'} \cup \{v\}) = (\widehat{S}_i^\ell \cap X_{t'}) \cup \{v\} = \widehat{P}_i^2 \cup \{v\} = P_i^2$. Thus the set S_i^ℓ has the correct intersection with X_t.

Finally, we prove that S_i induces a star. By induction hypothesis it holds that $\widehat{S}_i^\ell = S_i^\ell \setminus \{v\}$ is independent in $G_{t'} = G_t \setminus \{v\}$. The set $S_i^\ell \cap X_t = P_i^2$ is an independent set in G_t from validity of P_i, which proves v is not adjacent to any vertex $u \in (S_i \cap X_t) \setminus \{v\}$. The vertex v is also not adjacent to any vertex in $V_t \setminus X_t = V_{t'} \setminus X_{t'}$ which is a superset of $S_i^\ell \setminus P_i^2$ — refer to Lemma 2.2. We showed that v is not adjacent to any vertices in $\widehat{S}_i^\ell$, which implies that $\widehat{S}_i^\ell \cup \{v\} = S_i^\ell$ is an independent set. All that is left is to prove that the leaf vertices S_i^ℓ are adjacent to the center in $S_i^c = P_i^1$. The vertices in $S_i^\ell \setminus \{v\} = \widehat{S}_i^\ell$ were adjacent to the center by induction hypothesis. The vertex v is also adjacent to the center $S_i^c = P_i^2$ because P_i is valid, which requires $P_i^2 \subseteq N_{G_t}(P_i^1)$.

$\{v\} \subsetneq P_i^3$: Following the algorithm, we know that $\widehat{P}_i$ is of type T3 and by assumption P_i is also of type T3. The pair $(S_i^c, S_i^\ell) = (\emptyset, \widehat{S}_i^\ell \cup \{v\})$ is the solution we are looking for.

The fact that $S_i \cap X_t = P_i$ has already been proven and we omit it for this case.

The set P_i is of type T3, which means that we need to prove that $S_i^c = \emptyset$ and $P_i^3 = S_i^\ell$. By induction hypothesis we know all the leaf vertices are present in $\widehat{P}_i^3 = \widehat{S}_i^\ell = \widehat{S}_i$. Then, $S_i^\ell = \widehat{S}_i^\ell \cup \{v\} = \widehat{P}_i^3 \cup \{v\} = P_i^3$ proves that all leaf vertices are also in P_i^3 and $S_i \cap X_t = P_i^3 = P_i$. The center $\widehat{S}_i^c$ is by induction hypothesis empty and S_i also does not have a defined center.

Finally, we prove that S_i^ℓ is independent, but this is true as $P_i^3 = S_i^\ell$ is an independent set due to validity of P_i.

This concludes the description of partial solution S when $|P_i| \geq 2$.

Now we analyze the case $|P_i| = |\{v\}| = 1$, meaning v is the only vertex in P_i. The set $\widehat{P}_i = P_i \setminus \{v\}$ would be empty and we want to hold an invariant that $\emptyset \notin \widehat{P}$. This case only happens if v is the only vertex in P_i^1 or P_i^3. We again cannot have $v \in P_i^2 \wedge |P_i| = 0$.

From the algorithm, the number of sets in $\widehat{P}$ is $p-1$ while the original P was of size p. By induction hypothesis $\widehat{S}$ is a partial solution of size $\widehat{s} = (p-1)+h$. Then $S = (\widehat{S}_1, \dots, \widehat{S}_{i-1}, \{v\}, \widehat{S}_i, \dots \widehat{S}_{\widehat{s}})$ of size $\widehat{s}+1$ is the partial solution we are looking for.

For each set S_j such that $j < i$ we simply copied $S_j = \widehat{S}_j$. The intersection $\widehat{P}_j = P_j$ is the same in $X_{t'}$ and every other property prescribed by the definition of compatibility stays the same as nothing changed for the sets, nor the intersection.

Now consider the set $S_i = \{v\}$. We split the compatibility proof based on where v is in P_i. As we previously analyzed, there are only 2 cases we need to consider.

$v \in P_i^1 \wedge P_i^2 = \emptyset$: We create a new set $S_i = S_i^c = \{v\}$ with an empty set of leaves S_i^ℓ. Then for S_i to be compatible with P_i of type T1, we only need to show $P_i^1 = \{v\} = S_i^c$, which is trivially true. The set S_i trivially is a star with one vertex.

$\{v\} = P_i^3$: We create a new set $S_i = S_i^\ell = \{v\}$ with an empty center S_i^c. For S_i to be compatible with P_i of type T3, we only need to show $S_i^\ell = \{v\} = P_i^3$ and the center is not defined as required in the compatibility. The set S_i^ℓ with one vertex is trivially independent.

For every set S_j where $i < j \leq p$ we have $S_j = \widehat{S}_{j-1}$. From the algorithm it holds that $\widehat{P}_j = P_{j+1}$ whenever $j \in \widehat{p}$ and $j \geq i$. These two facts together prove that $S_j = \widehat{S}_{j-1}$ is compatible for $P_j = \widehat{P}_{j-1}$ for $i < j \leq p$.

Every forgotten star $\widehat{S}_j$ in t' has an equivalent forgotten star in t because there is a one-to-one mapping between $\widehat{S}_j$ and S_{j+1}.

This concludes the proof for one of the implications of the join node. To summarize, for all cases when $C[t', \widehat{P}]$ returned a finite value, we just take a compatible partial solution $\widehat{S}$ from the child node and add the newly introduced vertex v as center if $v \in P_i^1$ or as a leaf if $v \in P_i^2$ or $v \in P_i^3$. We either extend one of the existing sets $\widehat{S}_i$ that existed by induction hypothesis or create a new set S_i if $P_i = \{v\}$.

Now we show that if a partial solution S of size $s = p+h$ that is compatible for (t, P) exists, then the algorithm stores at most h into $C[t, P]$. To prove this, we slightly modify S and create $\widehat{S}$ which will be compatible for $(t', \widehat{P})$. The set $\widehat{S}$ will be of size $|\widehat{S}| = \widehat{s} = |\widehat{P}| + h = \widehat{p} + h$ and $\widehat{S}$ will still have h forgotten stars. The partial solution $\widehat{S}$ will be one of the sets considered by the definition of $C[t', \widehat{P}]$ and by induction hypothesis $C[t', \widehat{P}] \leq h$.

Let i be the index of P_i that contains the newly introduced vertex v. We again have two cases based on the size of $|P_i|$.

First consider the case $|P_i| \geq 2$. For each S_j such that $v \notin S_j$, we set $\widehat{S}_j = S_j$ and either $\widehat{S}_j$ has the same intersection $P_j = \widehat{P}_j$ with $X_{t'}$ (then S_i is still compatible with the intersection) or $\widehat{S}_j$ still is a forgotten star in t'. Again, when the set nor the intersection do not change, the proof of compatibility is straightforward.

We now describe the modification of S_i that contains v. We know by compatibility that P_i is the intersection of a star S_i with the bag X_t, thus S_i must contain at least another vertex that is not v. The idea is to create a set $\widehat{S}_i$ by removing v from S_i. The set $\widehat{S}_i$ is not empty and will still have a nonempty intersection with $X_{t'}$.

Vertex v could have been in P_i^1 or P_i^2 or P_i^3. Together with the condition $|P_i| \geq 2$, we again need to analyze three cases. Similarly as in the proof of the previous implication, for each case we describe the construction of $\widehat{S}_i$ and then prove that $\widehat{S}_i$ is compatible for $\widehat{P}_i$ at t' in three steps: (1) we show that the intersection of $\widehat{S}_i$ with $\widehat{X}_t$ equals exactly $\widehat{P}_i$, (2) we show that $\widehat{S}_i$ has the correct structure as prescribed by the *type* of $\widehat{P}_i$, (3) We show that $\widehat{S}_i$ is either a star or an independent set.

$v \in P_i^1 \wedge P_i^2 \neq \emptyset$: The set P_i is of type T0 and $(\widehat{P}_i^1, \widehat{P}_i^2, \widehat{P}_i^3) = (\emptyset, \emptyset, P_i^2)$ from the algorithm is of type T3. Then $(\widehat{S}_i^c, \widehat{S}_i^\ell) = (\emptyset, S_i^\ell)$ is the set we are looking for. By compatibility it holds that $S_i^c = P_i^c = \{v\}$.

First, $\widehat{S}_i$ has the correct intersection with $X_{t'}$: $\widehat{S}_i \cap X_{t'} = (S_i \setminus \{v\}) \cap (X_t \setminus \{v\}) = (S_i \cap X_t) \setminus \{v\} = P_i \setminus \{v\} = \widehat{P}_i$.

Second, we prove that $\widehat{S}_i^\ell = \widehat{P}_i^3$ and $\widehat{S}_i^c = \emptyset$. The center is indeed empty by construction and now we only need to show that $\widehat{S}_i^\ell = \widehat{P}_i^3$. Due to the definition of the tree decomposition, it holds that $N_{G_t}(v) \subseteq X_t$ because vertex v was introduced in t. It also holds that $S_i^\ell \subseteq N_{G_t}(S_i^c) = N_{G_t}(v)$ from compatibility, which means all the leaves S_i^ℓ are part of X_t. No leaf vertex $u \in S_i^\ell$ can be in $V_t \setminus X_t$ because then u would not be adjacent to the center v. From compatibility of S_i for P_i we know that $P_i^2 = S_i^\ell \cap X_t$. Therefore $\widehat{P}_i^3 = P_i^2 = S_i^\ell \cap X_t = S_i^\ell = \widehat{S}_i^\ell$.

Finally we $\widehat{S}_i$ is an independent set because $\widehat{S}_i = \widehat{S}_i^\ell = S_i^\ell$ was by assumption an independent set.

$v \in P_i^2 \wedge |P_i^1| = 1$ **:** Then P_i is of type T0 and $(\widehat{P}_i^1, \widehat{P}_i^2, \widehat{P}_i^3) = (P_i^1, P_i^2 \setminus \{v\}, \emptyset)$ is of type T0 or T1 depending on emptiness of $\widehat{P}_i^2$. In both cases $(\widehat{S}_i^c, \widehat{S}_i^\ell) = (S_i^c, S_i^\ell \setminus \{v\})$ is the set $\widehat{S}_i$ we need.

The fact that the intersection of $\widehat{S}_i$ with $X_{t'}$ equals $\widehat{P}_i$ has been proven in the previous case.

Now we need to show that $\widehat{S}_i^c = \widehat{P}_i^1$ and the intersection $\widehat{S}_i^\ell \cap X_{t'}$ equals $\widehat{P}_i^2$. The fact that the center equals $\widehat{P}_i^1$ can be shown trivially. Then, $\widehat{S}_i^\ell \cap X_{t'} = (S_i^\ell \setminus \{v\}) \cap (X_t \setminus \{v\}) = (S_i^\ell \cap X_t) \setminus \{v\} = P_i^2 \setminus \{v\} = \widehat{P}_i^2$. If $|P_i^2| \geq 2$, then $\widehat{P}_i^2 \neq \emptyset$ and we proved the compatibility for $\widehat{P}_i$ of type T0. Otherwise $|P_i^2| = 1$, therefore $\widehat{P}_i^2 = \emptyset$ and we proved that $\widehat{S}_i^\ell \cap X_{t'} = \emptyset$ as required by $\widehat{P}_i$ of type T1.

The set S_i induced a star and $\widehat{S}_i$ was created by removing a leaf vertex from a star, which again creates a star.

$\{v\} \subsetneq P_i^3$ **:** Then both P_i and $\widehat{P}_i$ are of type T3. We assign $(\widehat{S}_i^c \widehat{S}_i^\ell) = (\emptyset, S_i^\ell \setminus \{v\})$.

We again omit the proof that $\widehat{S}_i \cap X_{t'} = \widehat{P}_i$.

We know using compatibility of S_i with P_i, that $S_i^\ell = P_i^3$. Therefore we get a chain $\widehat{S}_i^\ell = S_i^\ell \setminus \{v\} = P_i^3 \setminus \{v\} = \widehat{P}_i^3$. By compatibility it holds that $S_i^c = \emptyset$ and $\widehat{S}_i^c = \emptyset$. Both facts together prove that $\widehat{S}_i$ has the correct structure as prescribed by $\widehat{P}_i$ of type T3.

The set $\widehat{P}_i^3$ is a subset of P_i^3, which is an independent set, thus $\widehat{P}_i^3 = \widehat{S}_i^\ell$ is also independent.

To sum it up, if $|P_i| \geq 2$, then for each S_j such that $v \notin S_j$ we copy $\widehat{S}_j = S_j$, and for S_i such that $v \in S_i$, we construct $\widehat{S}_i$ by simply remove vertex v from S_i.

Now let us move on to the case $|P_i| = 1$. We have two cases, either $P_i = P_i^1 = \{v\}$ or $P_i = P_i^3 = \{v\}$ (again, we cannot have the case $P_i = P_i^2 = \{v\}$). In both cases, we show that $S_i = \{v\}$ and the set S_i cannot have any other vertices. Thus by removing v from S_i, the set $\widehat{S}_i$ would become empty. Therefore $\widehat{S}$ will be of size $s-1$ but the number of forgotten stars will stay the same compared to the number of forgotten stars in S. This implies $C[t', \widehat{P}] \leq h$ as $\widehat{S}$ is one of the considered sets in the definition of $C[t', \widehat{P}]$.

$v \in P_i^1 \wedge P_i^2 = \emptyset$: The set P_i is of type T1 which means that $S_i^c = P_i^1 = \{v\}$ and $S_i^\ell \cap X_t = P_i^2$. The vertex v was newly introduced in t and can only be adjacent to vertices in X_t. That also implies that $S_i^\ell \subseteq X_t$ and subsequently it holds that $S_i^\ell = P_i^2 = \emptyset$. Using all these facts, we deduced that $|S_i| = 1$.

$\{v\} = P_i^3$: The set P_i is of type T3, which means that $P_i^3 = S_i^\ell = \{v\}$ and $S_i^c = \emptyset$. Therefore $|S_i| = |S_i^c| + |S_i^\ell| = 1$.

This concludes the proof of the implied by direction. We again briefly summarize the results of the previous proof. Assume that S is compatible for P at t, then create a $\widehat{S}$ based on the size of the set P_i that contains v. If $|P_i| \geq 2$, then $|\widehat{S}| = |S|$ and each set $\widehat{S}_j = S_j$ (if v is not part of S_j) or $\widehat{S}_i = S_i \setminus \{v\}$ if $v \in S_i$. Otherwise if $|P_i| = 1$, then it holds that $|S_i| = 1$ and we can create $\widehat{S}$ of size $s-1$ by removing S_i from S. In both cases the number of forgotten stars still stays the same and $C[t', \widehat{P}] \leq h$, which implies that $C[t, P]$ also is at most h.

There is one special case where $v \in P_i^2 \wedge P_i^1 = \emptyset$ and the algorithm stores $C[t, P] = \infty$.

Assume towards contradiction that there exists a partial solution S compatible for P at t, where $v \in P_i^2 \wedge P_i^1 = \emptyset$ (the set P_i is of type T2). The vertex v was newly introduced in t which implies that $N_{G_t}(v) \subseteq X_t$. Using compatibility, we know that the set S_i^c containing the center is not empty and $S_i^c \cap X_t = \emptyset$. On the other hand we also have $S_i^\ell \subseteq N_{G_t}(S_i^c)$ which means that for all $u \in S_i^\ell$ it holds that u is adjacent to the center $c \in S_i^c$. More importantly, it also implies that v and c are adjacent as $v \in S_i^\ell$. This is a contradiction, therefore no such S can exist and we indicate this by storing $+\infty$ into $C[t, P]$.

Forget node. Let $C[t, P] = h$ be a finite value that was stored by the algorithm, we need to prove that a partial solution S compatible for (t, P) of size $p + h$ exists. The value was computed by getting the minimum from one of two different cases. Either $C[t, P] = C[t', \widehat{P}] = h$ for some $\widehat{P} \in \widehat{\mathscr{P}}(P)$ or $C[t, P] = 1 + C[t', \widetilde{P}] = 1 + h''$ for some $\widetilde{P} \in \widetilde{\mathscr{P}}(P)$. Let v be the forgotten vertex.

If $C[t, P] = C[t', \widehat{P}] = h$, then by induction hypothesis there is a partial solution $S = (S_1, \ldots, S_s)$ compatible for $(t', \widehat{P})$ of size $s = \widehat{p} + h$, where $\widehat{p} = |\widehat{P}|$. Let $i \in [p]$ be the index of $\widehat{P}_i$ that contains the forgotten vertex v. From our algorithm, it holds that $|P| = |\widehat{P}|$ and $p = \widehat{p}$. We now prove that the same S is also compatible for P at t.

It can easily be shown that S is a partitioning of $V_t = V_{t'}$ and for each $j \in [s]$ it holds that the set S_j still has the correct structure in $G_t = G_{t'}$ (induces a star or is an independent set).

For each set S_j such that $p < j \leq s$: we know that $S_j \subseteq V_{t'} \setminus X_{t'}$, thus S_j is a forgotten star in $G_{t'}$. Subsequently S_j is also a forgotten star in G_t as $v \notin S_j$.

For all sets S_j such that $j \in [p] \setminus \{i\}$: we know that $S_j \cap X_{t'} = \widehat{P}_j$. Furthermore $v \notin \widehat{P}_j$ and for this reason, v is also not in S_j and S_j is still compatible for $P_j = \widehat{P}_j$.

Finally for $S_i = S_i^c \cup S_i^\ell$ such that $S_i \cap X_{t'} = \widehat{P}_i$ and $v \in \widehat{P}_i$, we know that $v \in S_i$. It holds that $P_i = \widehat{P}_i \setminus \{v\}$ and $X_t = X_{t'} \setminus \{v\}$. From this fact, we can conclude that $S_i \cap X_t = S_i \cap (X_{t'} \setminus \{v\}) = (S_i \cap X_{t'}) \setminus \{v\} = \widehat{P}_i \setminus \{v\} = P_i$, meaning S_i has the correct intersection with X_t (and the intersection equals exactly P_i).

We now proceed to show that S_i has the correct structure as prescribed by the *type* of P_i. We first analyze the types of $\widehat{P}_i$ and P_i and then prove that the conditions as described in Definition 5.3 are satisfied.

1. If the value h was obtained for case $v \in \widehat{P}_i^1$, then the algorithm also assigned $\widehat{P}_i^2 = P_i^2$ and $\widehat{P}_i^3 = P_i^3$. We know that $P_i^1 = \emptyset$ and $P_i^3 = \widehat{P}_i^3 = \emptyset$ because $|\widehat{P}_i^1| = 1$ and $\widehat{P}_i$ is valid. We can conclude that $P_i^2 \neq \emptyset$ because $(P_i^1, P_i^2, P_i^3) \neq (\emptyset, \emptyset, \emptyset)$. This implies that P_i is of type T2 and $\widehat{P}_i$ is of type T1.

   We now show compatibility of S_i for P_i of type T2 in X_t, meaning we show that the center S_i^c is not empty in G_t, the center is not present in X_t, and $P_i^2 = S_i^\ell \cap X_t$. Vertex v is part of $\widehat{P}_i^1 = S_i^c$ so the center still exists in $G_t = G_{t'}$. But the intersection of the center with X_t is now empty: $S_i^c \cap X_t = \{v\} \cap (X_{t'} \setminus \{v\}) = \emptyset$. Finally the set P_i^2 was unchanged and $v \notin S_i^\ell$, from this we can conclude that $S_i^\ell \cap X_t = S_i^\ell \cap X_{t'} = \widehat{P}_i^2 = P_i^2$.

2. Else if $v \in \widehat{P}_i^2$, then it holds that $P_i^1 = \widehat{P}_i^1, P_i^2 = \widehat{P}_i^2 \setminus \{v\}$ and $P_i^3 = \widehat{P}_i^3 = \emptyset$. The set $\widehat{P}_i^2$ is not empty, therefore $\widehat{P}_i$ was type T0 or T2.

   Assume that $\widehat{P}_i$ is of type T0. Then P_i is of type T1 if $\{v\} = \widehat{P}_i^2$, or T0 if $|\widehat{P}_i^2| \geq 2$. In both cases $P_i^1 = \widehat{P}_i^1$ is unchanged and it still equals S_i^c. Furthermore, $S_i^\ell \cap X_t = S_i^\ell \cap (X_{t'} \setminus \{v\}) = (S_i^\ell \cap X_{t'}) \setminus \{v\} = \widehat{P}_i^2 \setminus \{v\} = P_i^2$.

   Otherwise $\widehat{P}_i$ could have been of type T2, then because $P_i \neq (\emptyset, \emptyset, \emptyset)$ we can conclude that P_i is of type T2 and $|\widehat{P}_i| \geq 2$. By induction hypothesis it holds that $S_i^c = \emptyset$, thus $S_i^c \cap X_t = \emptyset \cap X_t = \emptyset$. The proof that $P_i^2 = S_i^\ell \cap X_t$ was given in the previous case.

This concludes the proof that $C[t, P] = C[t', \widehat{P}] = h$, then a partial solution S compatible for (t', P) is also compatible for (t, P).

Now we analyze the case $C[t, P] = h = 1 + C[t', \widetilde{P}] = 1 + h''$. By induction hypothesis there is a partitioning $S = (S_1, \ldots, S_s)$ compatible for $(t', \widetilde{P})$ of size

$s = \widetilde{p} + h''$, where $\widetilde{p} = p + 1$ is the size of $\widetilde{P}$ and $h'' = h - 1$ is the number of forgotten stars. Let v be the forgotten vertex and $\widetilde{P}_{p+1} = \{v\}$ (as prescribed by the algorithm). We now prove that S is also compatible for (t, P) and the number of forgotten stars in t is $h'' + 1 = h$.

For each set S_j such that $v \notin S_j$ the proof can be given the same way as in the previous case. We now analyze the case $v \in S_i$ for $i = p + 1$. First, by induction hypothesis it holds that $S_i \cap X_{t'} = \widetilde{P}_i = \{v\}$. Also $X_t = X_{t'} \setminus \{v\}$ and we get $S_i \cap X_t = S_i \cap (X_{t'} \setminus \{v\}) = (S_i \cap X_t) \setminus \{v\} = \emptyset$ which proves that the star S_i no longer has an intersection with the bag X_t. Now we proceed to show that S_i induces a forgotten star in G_t.

By construction of $\widetilde{P}$ either $v \in \widetilde{P}_i^1$ or $v \in \widetilde{P}_i^2$, meaning $\widetilde{P}_i$ is of type T1 or T2, respectively. In both cases, for S_i to be compatible with $\widetilde{P}_i$, we must have $S_i^c \neq \emptyset$ and $S_i^\ell \subseteq N_{G_t}(S_i^c)$. Altogether S_i indeed induces a star.

To wrap it up, if $C[t, P] = C[t', \widehat{P}]$, then the number of forgotten stars is still the same, which means the algorithm can safely store the value $s - p = (|\widehat{P}| + h) - |P| = h$. Otherwise for the case $C[t, P] = C[t', \widetilde{P}]$ it holds that S has $h'' = h - 1$ forgotten stars in $G_{t'}$ by induction hypothesis. But in t, the corresponding partial solution has $h'' + 1 = h$ forgotten stars and the stored value in $C[t, P] = 1 + C[t', \widetilde{P}]$ represents a partial solution compatible for P at t.

Now we get to the second part of the proof for the forget node. We show that if there is a partial solution S compatible for (t, P) of size $|S| = s = p + h$ with h forgotten stars, then the algorithm stores at most h in $C[t, P]$. To prove this, we show that there is a partition $\widehat{P} \in \widehat{\mathscr{P}}(P)$ or $\widetilde{P} \in \widetilde{\mathscr{P}}(P)$ of $X_{t'}$, such that S is compatible for $(t', \widehat{P})$ or $(t', \widetilde{P})$. This implies that S is one of the sets considered in the definition of $C[t', \widehat{P}]$ or $C[t', \widetilde{P}]$. Then, the algorithm will store a value that is at most h using the value $C[t', \widehat{P}]$ or $C[t', \widetilde{P}] + 1$.

Let $(S_i^c \cup S_i^\ell) = S_i \in S$ be a partial star, such that $v \in S_i$. We have two cases, either $S_i \cap X_t \neq \emptyset$ or $S_i \cap X_t = \emptyset$. Without loss of generality, assume that if $S_i \cap X_t = \emptyset$, then $i = p + 1$—we just permute the order of forgotten stars in S.

We know that $G_{t'} = G_t$ so S is a partitioning of $V_{t'}$ and furthermore if $v \notin S_j$, then either S_j is also compatible for $P_j = \widehat{P}_j = \widetilde{P}_j$, or S_j is still a forgotten star in $G_{t'}$.

Now consider the case $v \in S_i$. We have two cases, either $S_i \cap X_t \neq \emptyset$ as S_i has to have a correct structure as prescribed by P_i, or $S_i \cap X_t = \emptyset$ and S_i is a forgotten star.

In both cases, we first show that there is a partition $\widehat{P}$ or $\widetilde{P}$ such that S is also compatible for $(\widehat{P}, t')$ or $(\widetilde{P}, t)$. Then, we analyze the types of $\widehat{P}_i$ or $\widetilde{P}_i$ and P_i and then prove that S satisfies the conditions prescribed by the *type* of $\widehat{P}_i$ or $\widetilde{P}_i$.

First, consider the case $S_i \cap X_t = P_i$ and $P_i \neq \emptyset$.

a) If $S_i^c = \{v\} \subseteq V_t \setminus X_t$ then we know by compatibility of S with (t, P) that P_i is of type T2. Meaning $P_i^1 = \emptyset$ and the set of leaves P_i^2 is not empty as the intersection with X_t is not empty. Consider a set $(\widehat{P}_i^1, \widehat{P}_i^2, \widehat{P}_i^3) = (P_i^1 \cup \{v\}, P_i^2, P_i^3)$, which is of type T0.

First, we show that S_i has the correct intersection with $\widehat{P}_i$: $S_i \cap X_{t'} = S_i \cap (X_t \cup \{v\}) = P_i \cup \{v\} = \widehat{P}_i$.

Now we prove the compatibility of S_i with $\widehat{P}_i$ of type T0, meaning we need to show that the center equals to $\widehat{P}_i^1$ and the set $\widehat{P}_i^2$ is the intersection of S_i^ℓ with $X_{t'}$.

- The set P_i^1 is empty by compatibility of S_i with P_i which implies that $\widehat{P}_i^1 = P_i^1 \cup \{v\} = \{v\} = S_i^c$.
- The set $\widehat{P}_i^2 = P_i^2$ is unchanged and $S_i^\ell \cap X_{t'} = S_i^\ell \cap (X_t \cup \{v\}) = S_i^\ell \cap X_t = P_i^2 = \widehat{P}_i^2$.

Altogether the partitioning $\widehat{\mathcal{P}}_i^1(P)$ of $X_{t'}$ is a valid partitioning of $X_{t'}$ and S is compatible for $(t', \widehat{\mathcal{P}}_i^1(P))$.

b) If $v \in S_i^\ell$, then we need to distinguish the cases by where S_i^c can be found.

$S_i^c = \emptyset$ **:** The only type that is compatible with such S_i is P_i of type T3. That implies that $S_i^\ell = P_i^3$ which means that $v \in P_i^3$. Vertex v was forgotten and $v \in X_{t'}$ and $v \notin X_t$, which means that $v \notin P_i$. This is a contradiction and S_i^c needs to be nonempty.

$S_i^c \subseteq (V_t \setminus X_t)$: We assumed that $S_i \cap X_t \neq \emptyset$, thus at least one leaf vertex has to intersect X_t. The only type that is compatible with such S_i at t is P_i of type T2, meaning $P_i^1 = \emptyset$ and $P_i^2 \neq \emptyset$. The set S_i is compatible for $(\widehat{P}_i^1, \widehat{P}_i^2, \widehat{P}_i^3) = (P_i^1, P_i^2 \cup \{v\}, P_i^3)$ which is also of type T2. This case corresponds to the partition $\widehat{\mathcal{P}}_i^2(P)$.

The proof that S has the correct intersection with $X_{t'}$ is trivial and the details are left for the reader to fill in.

The set $\widehat{P}_i$ is of type T2, which means we have to show that (1) $S_i^c \neq \emptyset$, (2) $S_i^c \cap X_{t'} = \emptyset$, (3) $\widehat{P}_i^2 = S_i^\ell \cap X_{t'}$. The conditions (1) and (2) are satisfied trivially. For the condition (3) we know that $v \notin S_i^\ell$, thus $S_i^\ell \cap X_{t'} = S_i^\ell \cap X_t = P_i^2 = \widehat{P}_i^2$.

$S_i^c \subseteq X_t$: Then P_i is type T0 or T1 by compatibility of S_i for P_i and the set $(\widehat{P}_i^1, \widehat{P}_i^2, \widehat{P}_i^3) = (P_i^1, P_i^2 \cup \{v\}, P_i^3)$ is of type T0. This case corresponds to the partition $\widehat{\mathcal{P}}_i^2(P)$.

We again omit the proof that $S_i \cap X_{t'} = \widehat{P}_i$.

The set $\widehat{P}_i$ is type T0: We know that $S_i^c \cap X_t = P_i^1$ by compatibility and $v \notin S_i^c$, therefore $S_i^c \cap X_{t'} = P_i^1 = \widehat{P}_i^1$. Then for the set of leaf vertices it holds that $S_i^\ell \cap X_t = P_i^2$ and $X_{t'} = X_t \cup \{v\}$, therefore

$S_i^\ell \cap X_{t'} = P_i^2 \cup \{v\} = \widehat{P}_i^2$. Altogether we proved that S_i has the correct structure as prescribed by $\widehat{P}_i$.

Both valid cases correspond to the partition $\widehat{\mathscr{P}}_i^2(P)$ and we proved that S is compatible for $(t', \widehat{\mathscr{P}}_i^2(P))$.

We proved that if $v \in S_i$ and $S_i \cap X_t \neq \emptyset$, then he number of forgotten stars in $G_{t'}$ equals to $s - p = h$. The partial solution S is one of the sets considered in the definition of $C[t', \widehat{P}]$ for some $\widehat{P} \in \widehat{\mathscr{P}}(P)$, thus $C[t', \widehat{P}] \leq h$ and subsequently $C[t, P] \leq h$.

Now we analyze the case $S_i \cap X_t = \emptyset$. As mentioned in the beginning, we assume that $v \in S_i$ for $i = p + 1$. The number of forgotten stars in t' is $s - (p + 1) = h - 1$ as the only star that changed the number is S_i which was forgotten in t, but has a non empty intersection with $X_{t'}$ (it holds that $S_i \cap X_{t'} = \{v\}$).

The vertex $v \in X_{t'}$ could have been a center vertex or a leaf vertex in S_i. Then either $\widetilde{\mathscr{P}}^1(P)$ or $\widetilde{\mathscr{P}}^2(P)$ is the set we are looking for.

$v \in S_i^\ell$ **:** Consider a set $(\widetilde{P}_i^1, \widetilde{P}_i^2, \widetilde{P}_i^3) = (\emptyset, \{v\}, \emptyset)$ of type T2. The intersection $S_i \cap X_{t'} = S_i \cap (X_t \cup \{v\}) = (S_i \cap X_t) \cup \{v\} = \emptyset \cup \{v\}$ which shows that S_i now has a non empty intersection with $X_{t'}$. Then, we need to show that the center is defined, but is not in the bag $X_{t'}$, and the intersection $\widetilde{P}_i^2 = S_i^\ell \cap X_{t'}$.

By compatibility we have $|S_i^c| = 1$ in G_t and the center is still defined in $G_{t'}$, as the two graphs equal. Additionally, $S_i^c \cap X_{t'} = S_i^c \cap (X_t \cup \{v\}) = (S_i^c \cap X_t) \cup (S_i^c \cap \{v\}) = \emptyset \cup \emptyset$. Finally $S_i^\ell \cap X_{t'} = S_i^\ell \cap (X_t \cup \{v\}) = (S_i^\ell \cap X_t) \cup (S_i^\ell \cap \{v\}) = \emptyset \cup \{v\} = \{v\} = \widetilde{P}_i^2$.

$v \in S_i^c$ **:** Then consider a set $(\widetilde{P}_i^1, \widetilde{P}_i^2, \widetilde{P}_i^3) = (\{v\}, \emptyset, \emptyset)$ of type T1. As required by the definition of compatibility, $\widetilde{P}_i^1 = \{v\} = S_i^c$ and the fact that $S_i^\ell \cap X_{t'} = \emptyset$ is trivial.

Again, the partial solution S is one of the considered sets in the definition of $C[t', \widetilde{P}]$ and the value is at most $h - 1$. The algorithm will store a value at most $(h - 1) + 1 = h$ into $C[t, P]$.

Join node. Assume that the value h was stored into $C[t, P]$ by the algorithm using partitionings $\widehat{P}$ and $\widetilde{P}$ for $\widehat{t}$ and $\widetilde{t}$, respectively. Also let $\widehat{h} = C[\widehat{t}, \widehat{P}]$ and $\widetilde{h} = C[\widetilde{t}, \widetilde{P}]$ and $h = \widehat{h} + \widetilde{h}$.

By induction hypothesis there is a partial solution $\widehat{S}$ of size $\widehat{s} = p + \widehat{h}$ compatible for $(\widehat{t}, \widehat{P})$. The same applies to $\widetilde{S}$ of size $p + \widetilde{h}$ which is compatible for $(\widetilde{t}, \widetilde{P})$. We now show that a partial solution S of size $p + h$ compatible for (t, P) exists. The set S is in some way a combination of $\widehat{S}$ and $\widetilde{S}$.

First we analyze forgotten stars in $G_{\widehat{t}}$ and $G_{\widetilde{t}}$. We claim that every star $\widehat{S}_i \in \widehat{S}$, which is forgotten in $G_{\widehat{t}}$, has an empty intersection with $G_{\widetilde{t}}$. Assume towards a contradiction that there exists a vertex $v \in \widehat{S}_i$ which is in $V_{\widehat{t}}$ and in $V_{\widetilde{t}}$. Then vertex v also needs to be present in X_t from the definition of tree decomposition. Therefore the star $\widehat{S}_i$ would have a non-empty intersection with X_t, meaning it is in fact not a forgotten star. This is a contradiction. The same proof applies for a forgotten star $\widetilde{S}_i \in \widetilde{S}$. We set S' as the union of all forgotten stars in $\widehat{S}$ and $\widetilde{S}$. Thus $|S'| = \widehat{h} + \widetilde{h} = h$. Then, we set $(S_{p+1}, \ldots, S_s) = (S'_1, \ldots, S'_h)$ where $s = p + h$. These stars are indeed also forgotten in G_t because $X_t = X_{\widehat{t}} = X_{\widetilde{t}}$.

Now we analyze the stars that have an intersection with X_t. Due to compatibility, we know that $\widehat{S}_i \cap X_{\widehat{t}} = \widehat{P}_i$ and $\widetilde{S}_i \cap X_{\widetilde{t}} = \widetilde{P}_i$. We can combine the two stars and create $(S_i^c, S_i^\ell) = (\widehat{S}_i^c \cup \widetilde{S}_i^c, \widehat{S}_i^\ell \cup \widetilde{S}_i^\ell)$.

We claim that (S_i^c, S_i^ℓ) is compatible with P_i at t. We prove this in three stages: (1) we show that $S_i \cap X_t = P_i$, (2) we show that S_i has the correct structure as prescribed by the *type* of P_i, (3) we show that S_i is either a star or an independent set.

The fact that $S_i \cap X_t = P_i$ can be proven easily: $S_i \cap X_t = (\widehat{S}_i \cup \widetilde{S}_i) \cap X_t = \widehat{P}_i \cup \widetilde{P}_i = P_i$.

Then, we distinguish four cases based on the type of P_i. In all four cases, we first analyze the type of P_i, $\widehat{P}_i$ and $\widetilde{P}_i$ and then show that S_i has all the correct properties as prescribed by the type of P_i in Definition 5.3. Finally we also prove that S_i induces a star or is an independent set.

$P_i = \widehat{P}_i = \widetilde{P}_i$ **of type T0:** The set P_i is of type T1, which means we have to prove that $P_i^1 = S_i^c$ and $S_i^\ell \cap X_t = P_i^2$.

- By induction hypothesis it holds that $\widehat{S}_i^c = \widehat{P}_i^1 = P_i^1$ and $P_i^1 = \widetilde{P}_i^1 = \widetilde{S}_i^c$. We can conclude that $\widehat{S}_i^c = \widetilde{S}_i^c = S_i^c$, thus $S_i^c = P_i^1$.
- By induction hypothesis it holds that $P_i^2 = \widehat{P}_i^2 = \widehat{S}_i^\ell \cap X_{\widehat{t}}$ and $\widehat{S}_i^\ell \cap X_{\widehat{t}} = \widehat{P}_i^2 = P_i^2$. We also know that $X_t = X_{\widehat{t}} = X_{\widetilde{t}}$. Thus, $S_i^\ell \cap X_t = (\widehat{S}_i^\ell \cup \widetilde{S}_i^\ell) \cap X_t = (\widehat{S}_i^\ell \cap X_t) \cup (\widetilde{S}_i^\ell \cap X_t) = \widehat{P}_i^2 \cup \widetilde{P}_i^2 = P_i^2$.

Now we show that S_i induces a star in G_t. By induction hypothesis it holds that the sets $\widehat{S}_i^\ell$ and $\widetilde{S}_i^\ell$ are independent. We show that the union of the 2 sets also creates an independent set: For every pair of vertices $u, v \in S_i^\ell$ we can have 3 general cases.

1. If $u, v \in X_t$, then both vertices are part of $\widehat{P}_i^2 = P_i^2$ and by validity of P_i we checked that they are not adjacent.
2. Without loss of generality assume that $u \in X_t$ and $v \in V_{\widehat{t}} \setminus X_t$. Then the vertices are part of the same $\widehat{S}_i^\ell$ which is independent. The same applies symmetrically to $u \in X_t$ and $v \in V_{\widetilde{t}} \setminus X_t$.
3. If $u \in V_{\widehat{t}} \setminus X_t \wedge v \in V_{\widetilde{t}} \setminus X_t$, then u and v are not adjacent due to Lemma 2.2.

Furthermore, each individual leaf $v \in S_i^\ell$ was part of $\widehat{S}_i^\ell$ or $\widetilde{S}_i^\ell$ which is by induction hypothesis subset of $N_{G_{\widehat{t}}}(S_i^c)$ or $N_{G_{\widetilde{t}}}(S_i^c)$, respectively. This implies that the set of leaves S_i^ℓ is a subset of $N_{G_t}(S_i^c)$, thus the leaves are adjacent to the center and S_i induces a star in G_t.

$P_i = \widehat{P}_i = \widetilde{P}_i$ **of type T1:** The set P_i of type T1 prescribes that $P_i^1 = S_i^c$ (the proof is the same as in the previous case) and the leaf vertices are not present in X_t: $(S_i^\ell \cap X_t) = (\widehat{S}_i^\ell \cup \widetilde{S}_i^\ell) \cap X_t = (\widehat{P}_i \setminus \widehat{P}_i^1) \cup (\widetilde{P}_i \setminus \widetilde{P}_i^1) = \emptyset$.

The fact that S_i induces a star can be proven in a similar way as in the previous case.

$P_i = \widehat{P}_i = \widetilde{P}_i$ **of type T3:** Then by induction hypothesis we have $\widehat{S}_i^\ell = \widehat{P}_i^3 = P_i^3 = \widetilde{P}_i^3 = \widetilde{S}_i^\ell$ which in turns means all the sets used in the previous equation also equal S_i^ℓ. The set of leaves S_i^ℓ is independent as P_i^3 was assumed to be independent. The center S_i^c is empty because by induction hypothesis it holds that $\widehat{S}_i^c = \widetilde{S}_i^c = \emptyset$. Altogether, S_i is compatible with P_i of type T3.

We can also conclude that $S_i = S_i^\ell$ is an independent set.

P_i **of type T2** Assume that the algorithm used $(\widehat{P}_i^1, \widehat{P}_i^2, \widehat{P}_i^3) = (\emptyset, P_i^2, \emptyset)$ and $(\widetilde{P}_i^1, \widetilde{P}_i^2, \widetilde{P}_i^3) = (\emptyset, \emptyset, P_i^2)$. Then $\widehat{P}_i$ is of type T2 and $\widetilde{P}_i$ is of type T3.

By induction hypothesis we know that $|\widehat{S}_i^c| = 1$ and $\widetilde{S}_i^c = \emptyset$, which implies that $S_i^c = \widehat{S}_i^c$ and $|S| = |S_i^c| = 1$. The set $\widehat{S}_i^c$ is by induction hypothesis not part of the bag $X_{\widehat{t}}$, thus it also is not present in X_t which proves that $S_i^c \cap X_t = \emptyset$. Also $\widetilde{S}_i^\ell = \widetilde{P}_i^3 = P_i^2 = \widehat{P}_i^2 = \widehat{S}_i^\ell \cap X_{t'} \subseteq \widehat{S}_i^\ell$, and we can conclude that $S_i^\ell = \widehat{S}_i^\ell \cup \widetilde{S}_i^\ell = \widehat{S}_i^\ell$. This implies that $S_i^\ell \cap X_t = \widehat{S}_i^\ell \cap X_{\widehat{t}} = \widehat{P}_i^2 = P_i^2$.

We showed that $\widetilde{S}_i \subseteq \widehat{S}_i$, therefore $S_i = \widehat{S}_i$ and S_i induces a star.

The other case where $\widehat{P}$ is of type T3 and $\widetilde{P}$ is of type T2 is symmetric.

Now assume that there is a partial solution S of size $s = p + h$ compatible for (t, P), where $|P| = p$ and h is the number of forgotten stars in at t. Also let $(S_1, \dots, S_p)$ be the sets that have a non empty intersection with X_t, while $S_{p+1}, \dots S_s$ are stars that are forgotten in G_t. Then we show that the algorithm will store at most h into $C[t, P]$.

Consider a set $\widehat{S} = \{S_i \cap V_{\widehat{t}} \mid S_i \cap V_{\widehat{t}} \neq \emptyset\}$ such that the relative order of included elements is the same as in S. The first p sets S_i for $i \in [p]$ have a nonempty intersection with X_t. For such S_i, we include $S_i \cap V_{\widehat{t}}$ into $\widehat{S}$ because $X_t \subseteq V_{\widehat{t}}$. If S_i is a forgotten star in $G_{\widehat{t}}$ then it also is included. Otherwise the star S_i could also have been forgotten in $G_{\widetilde{t}}$, then such a star would have an empty intersection with $V_{\widehat{t}}$ and is not included in $\widehat{S}$. For $\widetilde{S} = \{S_i \cap V_{\widetilde{t}} \mid S_i \cap V_{\widetilde{t}} \neq \emptyset\}$ we have a symmetrical partial solution.

We now show that $\widehat{S}$ is compatible for some $\widehat{P}$ at $\widehat{t}$. First, we show that $\widehat{S}$ is a partitioning of $V_{\widehat{t}}$. Every vertex $v \in V_{\widehat{t}}$ was part of some set S_i because S was by assumption a partitioning. Then $S_i \cap V_{\widehat{t}} \neq \emptyset$ and indeed v is included in some $\widehat{S}_i$. Also, every vertex $v \in V_t \setminus V_{\widehat{t}}$ is not included in $\widehat{S}$ because we took $S_i \cap V_{\widehat{t}}$.

Let S_i be a forgotten star in G_t. By compatibility we know that $|S_i^c| = 1$, thus either $v \in V_{\widehat{t}} \setminus X_t$ or $v \in V_{\widetilde{t}} \setminus X_t$. Assume that $v \in V_{\widehat{t}} \setminus X_t$ (the case $v \in V_{\widetilde{t}} \setminus X_t$ is symmetrical and is omitted). We know by compatibility that $S_i^\ell \subseteq N_{G_t}(v)$. Using Lemma 2.2 and the assumption that $v \notin X_t$ we subsequently know that $N_{G_{\widehat{t}}}(v) \subseteq V_{\widehat{t}}$. Together with the fact that $S_i^\ell \cap X_t = \emptyset$ we get $S_i \subseteq (V_{\widehat{t}} \setminus X_t)$. This means that $S_i \cap V_{\widehat{t}} = S_i = \widehat{S}_i$ and $\widehat{S}_i$ is still a forgotten star in $G_{\widehat{t}}$.

Now consider S_i that intersect X_t. Using compatibility, we know that $S_i \cap X_t = P_i$. Then $\widehat{S}_i \cap X_{\widehat{t}} = (S_i \cap V_{\widehat{t}}) \cap X_{\widehat{t}} = S_i \cap X_{\widehat{t}} = S_i \cap X_t = P_i = \widehat{P}_i$. We just proved that that each $\widehat{S}_i$ still has a correct intersection with $X_{\widehat{t}}$ (and symmetrically also for $\widetilde{S}_i$ with $X_{\widetilde{t}}$ too).

In the following part we analyze the type of $S_i \cap X_t = P_i$ and proceed to show $(\widehat{S}, \widetilde{S})$ is compatible for one of the pair $(\widehat{P}, \widetilde{P}) \in \mathscr{P}(P)$. We have in total four cases. For each case, we show the compatibility of $\widehat{S}_i$ for $\widehat{P}_i$ and the compatibility of $\widetilde{S}_i$ for $\widetilde{P}_i$ in two steps: (1) we show that $\widehat{S}_i$ has the correct structure as prescribed by the *type* of $\widehat{P}_i$ (and symmetrically for $\widetilde{S}_i$ and $\widetilde{P}_i$), (2) we show that $\widehat{S}_i$ is either a star or an independent set (and symmetrically for $\widetilde{S}_i$).

$S_i \cap X_t = P_i$ **of type T0:** Then the algorithm assigned $\widehat{P}_i = P_i$ and both are of type T0.

The type T0 prescribes that (1) the center equals $\widehat{P}_1$: $\widehat{P}_i^1 = P_i^1 = S_i^c = \widehat{S}_i^c$ and (2) $\widehat{S}_i^\ell \cap X_{\widehat{t}} = (S_i^\ell \cap V_{\widehat{t}}) \cap X_t = S_i^\ell \cap X_t = P_i^2 = \widehat{P}_i^2$.

Furthermore, $\widehat{S}_i^\ell \subseteq S_i$, so $\widehat{S}_i^\ell$ is still an independent set and the leaves are still adjacent to the center $v \in \widehat{S}_i^c = \widehat{P}_i^1$.

For $\widetilde{P}_i$ and $\widetilde{S}_i$ the proof is symmetrical.

$S_i \cap X_t = P_i$ **of type T1:** Then the algorithm assigned $\widehat{P}_i = P_i$ and both are of type T1.

The center $\widehat{S}_i^c$ equals to $\widehat{P}_i^1$ trivially. None of the leaves S_i^ℓ intersected $X_t = X_{\widehat{t}}$, therefore $\widehat{S}_i^\ell \subseteq S_i^\ell$ also will not intersect $X_{\widehat{t}}$.

Again, $\widehat{S}_i \subseteq S_i$ and the center $\widehat{S}_i^c = S_i^c = \widehat{P}_i^1 \neq \emptyset$, thus $\widehat{S}_i$ indeed still induces a star.

For $\widetilde{P}_i$ and $\widetilde{S}_i$ the proof is symmetrical.

$S_i \cap X_t = P_i$ **of type T3:** Then the algorithm assigned $\widehat{P}_i = P_i$ and both are of type T3.

From compatibility we have $S_i^\ell = P_i^3$. This implies that $\widehat{S}_i^\ell = S_i^\ell \cap V_{\widehat{t}} = P_i^3 \cap V_{\widehat{t}} = P_i^3 = \widehat{P}_i^3$. The center was by compatibility empty, so $\widehat{S}_i^c = S_i^c \cap V_{\widehat{t}} = \emptyset$.

The set $\widehat{S}_i = \widehat{S}_i^\ell$ is trivially an independent set.

For $\widetilde{P}_i$ and $\widetilde{S}_i$ the proof is symmetrical.

$S_i \cap X_t = P_i$ **of type T2:** From compatibility we have $S_i^c \neq \emptyset$ and S_i^c is not part of X_t. This means either $S_i^c \subseteq V_{\widehat{t}} \setminus X_t$ or $S_i^c \subseteq V_{\widetilde{t}} \setminus X_t$.

First assume that $S_i^c \subseteq V_{\widehat{t}} \setminus X_t$. Then $(\widehat{P}_i^1, \widehat{P}_i^2, \widehat{P}_i^3) = (\emptyset, P_i^2, \emptyset)$ is the set we are looking for at $\widehat{t}$. The set $\widehat{P}_i$ is of type T2, therefore we need to show three properties: (1) The center $\widehat{S}_i^c \neq \emptyset$ and also (2) $\widehat{S}_i^c \cap X_{\widehat{t}} = \emptyset$. This is true because we assumed that the center is part of $V_{\widehat{t}} \setminus X_t$. (3) The intersection $\widehat{S}_i^\ell \cap X_{\widehat{t}} = (S_i^\ell \cap V_{\widehat{t}}) \cap X_{\widehat{t}} = S_i^\ell \cap X_t = P_i^2 = \widehat{P}_i^2$ which proves that the intersection of leaves with $X_{\widehat{t}}$ is as prescribed by $\widehat{P}_i^2$.

From the definition of tree decomposition we know that $S_i^c \cap (V_{\widetilde{t}} \setminus X_t) = \emptyset$. Then $(\widetilde{P}_i^1, \widetilde{P}_i^2, \widetilde{P}_i^3) = (\emptyset, \emptyset, P_i^2)$ is the set we are looking for at $\widetilde{t}$. The set $\widetilde{P}_i$ is of type T3 and we only need to prove two properties: (1) The center is by assumption empty and the definition requires $\widetilde{S}_i^c = \emptyset$. (2) Also, $\widetilde{S}_i^\ell \cap X_{\widetilde{t}} = (S_i^\ell \cap V_{\widetilde{t}}) \cap X_{\widetilde{t}} = S_i^\ell \cap X_t = P_i^2 = \widetilde{P}_i^3$.

The case $S_i^c \subseteq V_{\widetilde{t}} \setminus X_t$ is symmetrical. We use $(\widehat{P}_i^1, \widehat{P}_i^2, \widehat{P}_i^3) = (\emptyset, \emptyset, P_i^2)$ and $(\widetilde{P}_i^1, \widetilde{P}_i^2, \widetilde{P}_i^3) = (\emptyset, P_i^2, \emptyset)$.

We just proved that $\widehat{S}$ is compatible for $\widehat{P}$ at $\widehat{t}$ and $\widetilde{S}$ is compatible for $\widetilde{P}$ at $\widetilde{t}$ and furthermore $(\widehat{P}, \widetilde{P}) \in \mathscr{P}(P)$. Then $\widehat{S}$ is one of the considered sets in the definition of $C[\widehat{t}, \widehat{P}]$ and symmetrically for $\widetilde{S}$ with $C[\widetilde{t}, \widetilde{P}]$. Let $\widehat{h}$ be the number of forgotten stars $\widehat{S}_i \in \widehat{S}$ at $\widehat{t}$ and $\widetilde{h}$ be the number of forgotten stars $\widetilde{S}_i \in \widetilde{S}$ at $\widetilde{t}$. This all implies that $C[\widehat{t}, \widehat{P}] \leq \widehat{h}$ and $C[\widetilde{t}, \widetilde{P}] \leq \widetilde{h}$. Each forgotten star S_i is included either in $\widehat{S}$ or $\widetilde{S}$ but not in both of them, thus $\widehat{h} + \widetilde{h} = h$ and therefore $C[t, P] \leq C[\widehat{t}, \widehat{P}] + C[\widetilde{t}, \widetilde{P}] = h$. □

Theorem 5.3. The value at $C[r, \emptyset]$ equals q, if and only if G admits an induced star partition of size q.

Proof. First we show that if there exists a partial solution S of size s compatible for $(r, \emptyset)$, then there exists a way to partition G into q induced stars. From the algorithm, we know that $C[t, P]$ returns the number of forgotten stars in optimal partial solution S compatible for P at t. In this case, we have $P = \emptyset$ and $|\emptyset| = 0$, thus $|S| = C[r, \emptyset] = q$. Using compatibility, we know that each forgotten star S_i induces a star and S is a partitioning of $V_r = V(G)$. Each forgotten star S_i create its own partition and we have q of them.

Now we show that if there is a way to partition G into q induced stars, then $C[r, \emptyset] = q$. Let $S_1, \ldots, S_q$ be the partitioning of vertices $V(G)$ and each

$S_i \subseteq V(G)$ induces a star in G. We assign S_i^c as the center of the induced star S_i and $S_i^\ell = S_i \setminus S_i^c$. Then $S = (S_1, \ldots, S_q)$ is one of the considered partial solution in the definition of $C[r, \emptyset]$ because each set S_i trivially has an empty intersection with $P = \emptyset$ and each set S_i induces a star. The number of forgotten stars in S is q, thus $C[r, \emptyset] \leq q$. □

Theorem 5.4. Let G be a graph on n vertices given together with its nice tree decomposition T of width at most $tw(G)$. Then the Induced Star Partition problem on G is solvable in time $O(tw(G)^{2tw(G)} \cdot n)$.

Proof. The dynamic algorithm we just described works on a nice tree decomposition of width at most $tw(G) = k$. From the definition of tree width we get $|X_t| \leq k + 1$ for every node t. We also assumed the algorithm only works with valid partitioning, meaning every vertex $v \in X_t$ has to be part of a P_i and $\emptyset \notin P$, therefore $|P| \leq |X_t|$. Thus at node t we compute at most $|X_t|^{|P|} \leq (k + 1)^{(k+1)}$ values of $C[t, P]$.

The computation of $C[t, P]$ of a leaf node t can be done in $O(1)$ time. We just set a constant value.

The running time to compute $C[t, P]$ of an introduce node t using a valid partitioning P is $O(k)$: Finding where the newly introduced vertex v is in P takes $O(k)$ time and checking the size of each P_i^ℓ is in $O(1)$. The construction of $\widehat{P}$ can be done in $O(|P|) = O(k)$ time. We just copy the sets $\widehat{P}_j = P_j$ that do not contain v and for P_i that contains v, we create $\widehat{P}_i = P \setminus \{v\}$ or omit the set entirely.

For forget node t with valid P, the value $C[t, P]$ can be computed in $O(k^4)$ time: The algorithm needs to create $2(|P| + 1) \leq 2(k + 2)$ sets and remove sets that are invalid. The construction of $\widehat{P}$ and $\widetilde{P}$ can be done in $O(k)$ time. We copy the old P_i most of the time and only add a new vertex (the forgotten one) into one of the sets. The validity check a little bit more complicated. First, we need to check that each P_i^2 and P_i^3 is an independent set. This can be done in $O(k^3)$ time. We iterate through all pair of vertices $u, v \in P_i^2$ or P_i^3, depending on whichever one is not empty, and check if they are adjacent in G in $O(k)$ time (refer to Lemma 2.3). Additionally we need to check that all leaf vertices $u \in P_i^2$ are adjacent to the center $c \in P_i^1$. This can be done in $O(k^2)$ time. In total, the validity check can be finished in $O(k^3)$ time.

The value $C[t, P]$ at a join node t for a valid partitioning P can be computed in $O(2^k \cdot k)$ time. We create at most $2^{|P|}$ pairs $(\widehat{P}, \widetilde{P})$ and each $\widehat{P}$ and $\widetilde{P}$ can be constructed in $O(k)$ time.

To wrap it up, we compute at most $(k + 1)^{(k+1)}$ values of $C[t, P]$ and to compute each $C[t, P]$ we need $O(2^k \cdot k)$ time. We can assume that the number of nodes of the given nice tree decomposition is $O(k \cdot n)$. Thus the total running time is $O(tw(G)^{2tw(G)} \cdot n)$. □

Chapter 6

Implementation, Testing, and Evaluation

This chapter is dedicated to the implementation details of the algorithm presented in Section 4.4. We start with the implementation choices, requirements for installation and usage guide for the program. Then we briefly describe the included unit tests. Finally we analyze the performance of the implementation.

6.1 Choice of algorithm and programming language.

We provide a simple implementation of our algorithm parameterized by the vertex cover number as described in Section 4.4. The main reason why we chose to implement the algorithm for vertex cover and not for treewidth is that the algorithm parameterized by vertex cover is a lot simpler to implement compared to dynamic programming on tree decomposition. Another important reason is that the algorithms heavily depend on the given vertex cover and the tree decomposition, respectively. In practice, these parameters often also need to be computed. From our experience, implementations for the minimum vertex cover problem are easier to use, understand, and implement compared to treewidth decomposition solvers. Refer to PACE challenge [29] for recent treewidth and vertex cover implementation results. A very powerful tool for finding minimum vertex cover is GUROBI [30]. We leverage GUROBI's optimization capabilities to find a minimum vertex cover using integer linear programming, more details will be discussed in Subsection 6.1.3.

The goal was to implement an efficient solver, thus C++ was chosen. The templated library offers optimized implementation of basic data structures and we can use GUROBI's C++ API to calculate the vertex cover.

```
int inducedStarPartitioning_vc(const Graph & g,
                               const std::vector<vertex> & vc,
                               StarPartitions &S);
```

Listing 1: Signature of solver function.

```
bool isValidStarPartition(const Graph &g,
                          const StarPartitions &S,
                          int solutionSize);
```

Listing 2: Signature of validator function.

6.1.1 Requirements

The implementation is written in C++ 17 and a standard compiler for C++ is needed, such as `gcc` or `clang`. There is a `makefile` prepared together with the implementation, therefore we recommend also having the latest version of `make`. On our system we used the version GNU Make 4.2.1.

The algorithm is parameterized by vertex cover, thus a vertex cover needs to be provided. In our implementation, we compute a minimum vertex cover using GUROBI [30], which is a commercial tool. The installation of GUROBI can be omitted, but then a vertex cover needs to be provided on the input.

Together with the implementation we also provide sets of unit tests to test edge cases. The tests are written using Google test framework. We recommend version `v1.13.0` or higher. The installation process of GOOGLE TEST can be omitted, the tests and the solver are not dependent on each other.

6.1.2 Solver

We provide two important functions: one invokes the solver, the other verifies that the given partition is indeed an induced star partition.

The function `inducedStarPartitioning_vc` invokes the solver and is available in `starPartitioning/inducedStarPartitioning.hpp`. The signature of the function is shown in Listing 1. The function accepts a graph g of type `Graph` and a vertex cover as a list of vertices from g. The function returns the induced star partition number q and also provides a certificate—the partition $S = (S_1, \ldots, S_q)$ where each S_i is a star. The time complexity heavily relies of the size of the given vertex cover `vc` as described in the algorithm.

The second important function `isValidStarPartition` verifies that the given `StarPartitions S` has the correct structure. Note that the verifier does not check if the size is minimal. The signature of the function is shown in Listing 2. The function accepts a graph g and its induced star partition S and the size of S.

6.1.3 External solvers

There are two external solvers used in the implementation. One of them is GUROBI to calculate the exact minimum vertex cover of the graph, the other is a solver for the max flow problem.

GUROBI [30] is a powerful multi-purpose optimization tool and we use its capabilities to calculate a minimum vertex cover of the given graph. The vertex cover problem has an Integer linear programming formulation [6, page 33] that we use to model the problem in GUROBI:

$$\begin{aligned} &\text{Minimize} && \sum_{v \in V(G)} x_v \\ &\text{Subject to:} && x_u + x_v \geq 1 && \forall \{u, v\} \in E(G) \\ & && x_v \in \{0, 1\} && \forall v \in V(G). \end{aligned} \tag{6.1}$$

The vertex cover solver is calculated in a function called `getVertexCover`, which can be found in `externalSolver/vertexCoverSolver.hpp`. We wrap the whole function in `__GUROBI__` directive in case GUROBI is not installed.

The other external algorithm that we use in the implementation is Dinic's algorithm available from KACTL [31]. The main reason why we chose this implementation is that the implementation is highly optimized and well tested. The solver can be found in `externalSolver/dinic.hpp`.

6.1.4 Usage

The first step is the compilation of the executable binary file. We provide a simple `makefile` for this purpose. The following expected scenarios are:

1. Just compiling the solver: Use the command `make notests` and then the executable can be found in `exe/main`.

2. Just running the tests: Use the command `make tests` which will compile the tests and execute them (note that `gTest` needs to be installed).

3. Compile without GUROBI: The `makefile` automatically detects if GUROBI is not installed and no further modifications need to be made.

4. Compile with GUROBI: The INCLUDES flag needs to be changed within the makefile. If GUROBI is not installed in the correct location, the include address needs to be changed. The default location is at `~/gurobi952/linux64/include/`. Otherwise no further changes need to be made.

5. Clean up the folder: Use the command `make clean` to remove all compilation files.

```
./exe/main instances/smallVCInstances/in1.gr \
instances/smallVCInstances/vc1.gr
```

Listing 3: An example of starting the program from command line.

After compiling, the executable file can be found in the `exe/` folder. The solver can be executed from the command line as shown in Listing 3. The program expects at least one parameter. The first parameter is mandatory and the program expects a graph in the `.gr` format (described in Subsection 6.1.5). The second parameter is optional and is used to pass the vertex cover of the input graph to the program. The vertex cover file format is also described in Subsection 6.1.5. If no vertex cover is given, then the program computes one using GUROBI.

6.1.5 Input and output format

The program expects a graph in `.gr` format as the first argument. The `.gr` format is used in the PACE challenge [29], more specifically we use the format of the 2019 vertex cover challenge[2]. For completeness we describe the format again in this thesis. Note that the format is similar to the DIMACS graph format[3].

- Each line is separated by a newline `'\n'`.
- Lines starting with character c are interpreted as comments.
- Vertices are consecutively numbered from 1 to n.
- The first line (that is not a comment) is the problem description and has the following structure:
  - Line starts with character p,
  - followed by the problem descriptor (we ignore this descriptor),
  - followed by number n of vertices,
  - followed by number m of edges.

  No other line may start with p.
- Remaining m lines (that are not comments) indicate edges consisting of two integers (vertex identifiers) separated by a space.
- Graphs may contain isolated vertices, but self-loops and multiedges are forbidden.

[2] `https://pacechallenge.org/2019/vc/vc_format/`
[3] `http://archive.dimacs.rutgers.edu/pub/challenge/graph/doc/ccformat.tex`

The second parameter is optional and is used to pass a vertex cover of the input graph to the program. The format is also from the PACE 2019 challenge.

- Each line is separated by a newline `'\n'`.
- Lines starting with character c are interpreted as comments.
- Vertices are consecutively numbered from 1 to n.
- The first line (that is not a comment) is the solution description and has the following structure:
  - Line starts with character s,
  - followed by the problem descriptor (we ignore this descriptor),
  - followed by number n of vertices,
  - followed by number v of vertices in the vertex cover.

  No other line may start with s.
- Remaining v lines (that are not comments) indicate vertices consisting of one integer (vertex identifier).

The output format of an induced star partition S is as follows:

- The first line consists of one integer q—the induced star partition number of the input graph.
- Then follow $2q$ lines—the description of each induced star:
  - The first line is an integer s_i—size of the i-th star,
  - followed by s_i integers on second line—the first integer being the center, followed by $s_i - 1$ leaf vertices.

6.2 Testing

The implementation is also accompanied with unit tests to ensure the correctness of the implemented functions. The unit tests were created with the help of `gTEst` [32] framework provided by Google.

All tests are included in the `tests` folder. It is possible to exclude the tests during compilation using `make notests` and files without tests will be compiled.

To run all the tests, we include a command in makefile: `make tests`.

6.3 Experimental results

In this section we analyze the performance of the program. To the best of our knowledge, there is no other known implementation for this problem and we cannot compare the results with another implementation.

6.3.1 Environment

All measurements were performed on a local machine with an AMD Ryzen 5 CPU, the specifications of the hardware and software as follows:

	Environment
CPU	AMD Ryzen 5 4600H @ 3.00GHz
RAM	DDR4 16 GB @ 2667MHz
OS	Windows 10 with WSL2 (Ubuntu-20.04)
g++	(GCC)9.4.0
optimization flags	-O2

Table 6.1: Specification of the environment used to perform measuring.

6.3.2 Dataset

Our algorithm has to try in the worst case up to k^{2k+1} partitions, where k is the size of the given vertex cover. For this reason, we use graphs with small vertex cover to ensure that the program will terminate. Unfortunately, there are no universal datasets with small vertex cover, therefore all of the graphs we used in the performance evaluation process are generated in the following way.

Suppose we want to construct a graph g on n vertices with a vertex cover k and d being the density parameter of the graph. Then we construct g in using the following steps:

1. generate a graph g on n vertices and no edges,
2. select arbitrary k vertices from the graph g and declare them as the vertex cover C,
3. add edges in two stages:
    a) generate edges between the vertex cover C and $V \setminus C$: enumerate all edges between C and $V \setminus C$, the number of such edges is $m_1 = k \cdot (n-k)$, then select $m_1 \cdot d$ of these edges randomly and add them to g.

```
Graph generateGraph_VC(size_t vertexCount, size_t VC_size,
                                    double edgeRate, int seed,
                                    std::vector<vertex> & VC);
```

Listing 4: Signature of graph generating function.

   b) generate edges within the vertex cover C: enumerate all edges between two vertices in C, the number of such edges is $m_2 = \frac{k(k-1)}{2}$, then select $m_2 \cdot d$ of these edges randomly and add them to g.

4. add additional edges to ensure there are no isolated vertices:

   a) for each $v \in V \setminus C$ such that $deg_g(v) = 0$: select an arbitrary vertex $u \in C$ and add edge $\{v, u\}$ to g,

   b) for each $v \in C$ such that $\deg_g(v) = 0$: select an arbitrary vertex $u \in C$ such that $\deg_g(u) > 0$, then add edge $\{u, v\}$ to g.

The graphs we constructed using the `generateGraph_VC` function available from `utility/instanceGenerators.hpp`. The signature is shown in Listing 4. The constructed graph is guaranteed to have a vertex cover of size at most `VC_size` and no isolated vertices. Then, we return the selected vertex set C through the output parameter `VC` and the graph g as return parameter of the function.

6.3.3 Methodology

For each instance, we measure only the time of the actual partitioning algorithm. We ignore time needed to load the graph and time spent on outputting the solution. We also ignore the time needed to load the vertex cover from memory.

The time is calculated as the difference between two timestamps. The first timestamp is created before calling `inducedStarPartitioning_vc`. The second timestamp is created immediately after returning from the called function. The timestamps are created using `std::chrono::high_resolution_clock`.

The instances we use in the experiments were generated using our generator as described in Subsection 6.3.2. Then, we pass generated vertex cover to the algorithm instead of computing a minimum vertex cover.

6.3.4 Results

We start the experiment with small graphs to get a better understanding of the solver. In our first experiment, we generated graphs with n vertices in range between 10 and 100, with vertex cover size at most 9, and edge rate parameter d from 0.1 to 1. We used 900 instances of different combinations of

	n	m	edge_rate	vc	q	time[ms]
0	100	191	0.2	9	9	21832.2
1	100	262	0.3	9	8	25353.5
2	90	305	0.4	9	7	21164.3
3	100	343	0.4	9	7	20993.4
4	100	427	0.5	9	6	21018.6

Table 6.2: Top five small instances with longest solve time from the first experiment. The running time is displayed in ms. The column n shows the number of vertices of the generated graph, m is the number of edges, edge rate is the parameter d used in the graph generator, vc is the size of the vertex cover used in the algorithm and q is the induced star partition number of the graph.

n, vertex cover size and edge rate parameter and show only a fraction of all measurements can be seen in Table B.1. The full table with all 900 instances can be found on the included external medium as `csv` file. The total run time for all 900 instances was under 10 minutes and the solver successfully computed a solution for each instance within 30 seconds. We can safely conclude that for such small instances the program will terminate and produce an optimal solution in a reasonable time. The instance that ran the longest took 25 seconds and we show 5 instances with the longest solve time in Table 6.2.

In Figure 6.1 we can see the relation of growing number of vertices and the total runtime. We show the value for various edge rate parameters and can conclude that the implementation works well for denser graphs (higher edge rate parameter and subsequently more edges) compared to sparser graphs.

The second experiment we performed was on graphs with bigger vertex cover, more specifically in range 10 to 15. We wanted to know for which vertex cover size we can still solve the problem in a reasonable time. We set the time limit for each instance at 20 minutes.

This experiment ran in two phases. During the initial testing we used graphs with edge rate between 0.2 and 0.8 and n from 70 to 150. We noticed the solver was not able to compute the solution within the designated time for graphs generated with parameters $vc = 12$ and edge rate $d = 0.2$. It seems that for sparse graphs the number of iterations the algorithm has to go through is very high and our branch cutting optimization does not get applied very often. We paused the experiment and adjusted the edge rate. The second phase consists of generating graphs with the same range of n and vc but edge rate d is between 0.6 and 0.8. The total run time of the experiment was around 8 hours and we measured 88 instances in total, of which only 76 instances were successfully solved within the designated 20 minutes. We again

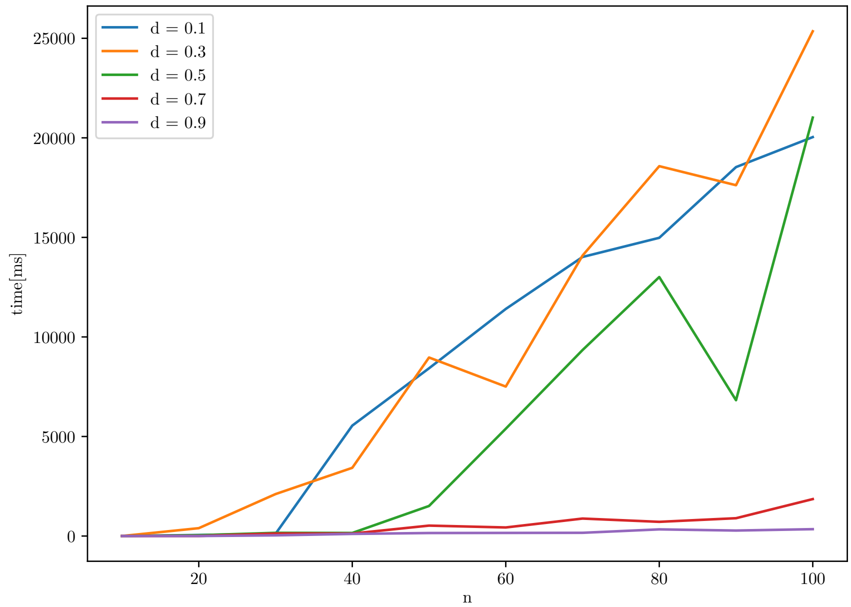

Figure 6.1: Relation between number of vertices and runtime for small instances with $vc = 9$ and various edge rate d values.

show a fragment of all the measurements in Table B.2 and all 88 instances are included in the external medium.

The results are as follows:

1. If the edge rate d is small (between 0.2 and 0.5), then we can only solve for small vertex covers. For $vc \geq 13$ the program can no longer compute a solution within designated 20 minutes.

2. When the edge rate d is between 0.6 and 0.8, then we can still compute a solution for vertex cover size $vc = 15$. We can conclude that $vc \leq 15$ and $n \approx 100$ is the limit of our solver.

So far we have only measured the running time for increasing vertex cover size. In our final experiment, we focused mainly on having fixed vertex cover and edge rate d, but increased the number of vertices n drastically. We generate graphs with $vc = 5$, edge rate $d = 0.6$ and number of vertices n in range 1000 to 5000. The total runtime for 21 instances took around 2.5 hours. The largest instance our solver could still handle was with $n = 4750$ with running time 18 minutes. For $n \geq 5000$ the program was no longer able compute a

	n	m	d	vc	q	time[s]
0	2000	6010	0.6	5	5	83.3
1	2250	6760	0.6	5	5	116.7
2	2500	7516	0.6	5	5	159.9
3	2750	8268	0.6	5	5	221.8
4	3000	9016	0.6	5	5	317.8
5	3250	9782	0.6	5	5	448.8
6	3500	10530	0.6	5	5	451.6
7	3750	11286	0.6	5	5	647.0
8	4000	12023	0.6	5	5	723.8
9	4250	12790	0.6	5	5	840.4
10	4500	13536	0.6	5	5	937.7
11	4750	14291	0.6	5	5	1105.1

Table 6.3: Solve time for instances with large number of vertices. Edge rate parameter d and vertex cover vc size are fixed. Instances with number of vertices $n \geq 5000$ were not solved within the designated 20 minutes and are not shown in the table.

solution within the designated 20 minutes. We show the complete results in Table 6.3.

This concludes our experimental results. We now sum up our results. For small graphs with small vertex cover and low number of vertices ($vc \leq 9$ and $n \leq 100$) we conclude that the program will find an optimal solution very quickly. For vertex cover at most 15 and $n \leq 150$ we can only solve on graphs with edge rate at least 0.6. Lastly, for small vertex cover (at most 5) and edge rate $d \geq 0.6$, our implementation can solve the Induced Star Partition problem on graphs with $n \leq 5000$.

Conclusion

Goals and results

The goal of this thesis was to study and develop new algorithms for the INDUCED STAR PARTITION problem using structural parameters. We present three new results for this problem: (1) The problem is FPT when parameterized by the vertex cover number of the graph and there is an exact $O(k^{2k+1}n^2)$ time algorithm, where k is the vertex cover number of the input graph. (2) The problem is FPT when parameterized by the treewidth of the graph and there is an exact $O(tw(G)^{2tw(G)} \cdot n)$ time algorithm, where $tw(G)$ is the treewidth of the input graph. (3) For a fixed q, the problem can be solved linear time on graphs with bounded cliquewidth.

We also discuss the implementation of the algorithm parameterized by the minimum vertex cover of the graph as described in Section 4.4. The program can successfully compute the exact solution in a reasonable time (under 1 minute) for all of our generated instances with small vertex cover ($k \leq 10$ and $n \leq 100$). The implementation can also solve most of the instances with vertex cover size at most 15 on sparse graphs with $n \leq 150$ vertices within 20 minutes.

Future work

In our work we also proved that the problem can be solved in linear time on graph with bounded cliquewidth using Courcelle's theorem [12] if a construction sequence is given together with the graph. A natural continuation is developing a dynamic programming algorithm on the construction sequence of operations used in the definition of cliquewidth.

The problem parameterized by the vertex cover number is FPT, therefore a natural question arises: Does the INDUCED STAR PARTITION problem admit a polynomial kernel when parameterized by the vertex cover number? We only showed one trivial reduction rule that deals with isolated vertices and we would be interested in more reduction rules that could improve our algorithm.

Another direction of research could be towards improving our algorithm or showing that our algorithm is ETH-tight.

Bibliography

1. SHALU, M.A.; VIJAYAKUMAR, S.; SANDHYA, T.P.; MONDAL, Joyashree. Induced star partition of graphs. *Discrete Applied Mathematics*. 2022, vol. 319, pp. 81–91. ISSN 0166-218X. Available from DOI: `https://doi.org/10.1016/j.dam.2021.04.015`.
2. BJÖRKLUND, Andreas; HUSFELDT, Thore; KOIVISTO, Mikko. Set Partitioning via Inclusion-Exclusion. *SIAM Journal on Computing*. 2009, vol. 39, no. 2, pp. 546–563. Available from DOI: `10.1137/070683933`.
3. BEVERN, René; BREDERECK, Robert; BULTEAU, Laurent; CHEN, Jiehua; FROESE, Vincent; NIEDERMEIER, Rolf; WOEGINGER, Gerhard. Partitioning Perfect Graphs into Stars. *Journal of Graph Theory*. 2017, vol. 85, pp. 297–335. Available from DOI: `10.1002/jgt.22062`.
4. ANDREATTA, Giovanni; DE GIOVANNI, Luigi; SERAFINI, Paolo. Optimal shift partitioning of pharmacies. *Computers and Operations Research*. 2015, vol. 55, pp. 88–98. ISSN 0305-0548. Available from DOI: `https://doi.org/10.1016/j.cor.2014.09.009`.
5. ARORA, Sanjeev; BARAK, Boaz. *Computational Complexity: A Modern Approach*. 1st. USA: Cambridge University Press, 2009. ISBN 0521424267.
6. CYGAN, Marek; FOMIN, Fedor V.; KOWALIK, Lukasz; LOKSHTANOV, Daniel; PILIPCZUK, Marcin; PILIPCZUK, Michal; SAURABH, Saket; MARX, Daniel. *Parameterized Algorithms*. 1st. Springer Publishing Company, Incorporated, 2015. ISBN 3319212745.
7. DOWNEY, Rodney G.; FELLOWS, Michael R. *Fundamentals of Parameterized Complexity*. Springer Publishing Company, Incorporated, 2013. ISBN 1447155580.
8. KARP, Richard M. Reducibility among Combinatorial Problems. In: *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New*

York. Boston, MA: Springer US, 1972, pp. 85–103. ISBN 978-1-4684-2001-2. Available from DOI: 10.1007/978-1-4684-2001-2_9.

9. CHEN, Jianer; KANJ, Iyad A.; XIA, Ge. Improved upper bounds for vertex cover. *Theoretical Computer Science*. 2010, vol. 411, no. 40, pp. 3736–3756. ISSN 0304-3975. Available from DOI: https://doi.org/10.1016/j.tcs.2010.06.026.

10. CORNEIL, Derek G.; ROTICS, Udi. On the Relationship between Clique-Width and Treewidth. In: BRANDSTÄDT, Andreas; LE, Van Bang (eds.). *Graph-Theoretic Concepts in Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 78–90. ISBN 978-3-540-45477-9.

11. COURCELLE, Bruno. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*. 1990, vol. 85, no. 1, pp. 12–75. ISSN 0890-5401. Available from DOI: https://doi.org/10.1016/0890-5401(90)90043-H.

12. COURCELLE, B.; MAKOWSKY, J. A.; ROTICS, U. Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width. *Theory of Computing Systems*. 2000, vol. 33, no. 2, pp. 125–150. ISSN 1433-0490. Available from DOI: 10.1007/s002249910009.

13. CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN 0262033844.

14. FORD, L. R.; FULKERSON, D. R. Maximal Flow Through a Network. In: *Classic Papers in Combinatorics*. Ed. by GESSEL, Ira; ROTA, Gian-Carlo. Boston, MA: Birkhäuser Boston, 1987, pp. 243–248. ISBN 978-0-8176-4842-8. Available from DOI: 10.1007/978-0-8176-4842-8_15.

15. EDMONDS, Jack; KARP, Richard M. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM*. 1972, vol. 19, no. 2, pp. 248–264. ISSN 0004-5411. Available from DOI: 10.1145/321694.321699.

16. MÜLLER, Haiko; BRANDSTÄDT, Andreas. The NP-completeness of steiner tree and dominating set for chordal bipartite graphs. *Theoretical Computer Science*. 1987, vol. 53, no. 2, pp. 257–265. ISSN 0304-3975. Available from DOI: https://doi.org/10.1016/0304-3975(87)90067-3.

17. DUGINOV, Oleg. Partitioning the vertex set of a bipartite graph into complete bipartite subgraphs. *Discrete Mathematics & Theoretical Computer Science*. 2014, vol. Vol. 16 no. 3. Available from DOI: 10.46298/dmtcs.2090.

18. COCKAYNE, E.; GOODMAN, S.; HEDETNIEMI, S. A linear algorithm for the domination number of a tree. *Information Processing Letters*. 1975, vol. 4, no. 2, pp. 41–44. ISSN 0020-0190. Available from DOI: `https://doi.org/10.1016/0020-0190(75)90011-3`.
19. DAMASCHKE, Peter; MÜLLER, Haiko; KRATSCH, Dieter. Domination in convex and chordal bipartite graphs. *Information Processing Letters*. 1990, vol. 36, no. 5, pp. 231–236. ISSN 0020-0190. Available from DOI: `https://doi.org/10.1016/0020-0190(90)90147-P`.
20. JOHNSON, David S. Approximation Algorithms for Combinatorial Problems. In: *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*. Austin, Texas, USA: Association for Computing Machinery, 1973, pp. 38–49. STOC '73. ISBN 9781450374309. Available from DOI: `10.1145/800125.804034`.
21. RAZ, Ran; SAFRA, Shmuel. A Sub-Constant Error-Probability Low-Degree Test, and a Sub-Constant Error-Probability PCP Characterization of NP. In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*. El Paso, Texas, USA: Association for Computing Machinery, 1997, pp. 475–484. STOC '97. ISBN 0897918886. Available from DOI: `10.1145/258533.258641`.
22. VAZIRANI, V.V. *Approximation Algorithms*. Springer Berlin Heidelberg, 2013. ISBN 9783662045657. Available also from: `https://books.google.cz/books?id=bJmqCAAAQBAJ`.
23. RAMAN, Venkatesh; SAURABH, Saket. Short Cycles Make W-hard Problems Hard: FPT Algorithms for W-hard Problems in Graphs with no Short Cycles. *Algorithmica*. 2008, vol. 52, no. 2, pp. 203–225. ISSN 1432-0541. Available from DOI: `10.1007/s00453-007-9148-9`.
24. ANDREATTA, G.; DE FRANCESCO, C.; DE GIOVANNI, L.; SERAFINI, P. Star partitions on graphs. *Discrete Optimization*. 2019, vol. 33, pp. 1–18. ISSN 1572-5286. Available from DOI: `https://doi.org/10.1016/j.disopt.2019.01.002`.
25. KIRKPATRICK, David G.; HELL, Pavol. On the Completeness of a Generalized Matching Problem. In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. San Diego, California, USA: Association for Computing Machinery, 1978, pp. 240–245. STOC '78. ISBN 9781450374378. Available from DOI: `10.1145/800133.804353`.
26. EDMONDS, Jack. Paths, Trees, and Flowers. *Canadian Journal of Mathematics*. 1965, vol. 17, pp. 449–467. Available from DOI: `10.4153/CJM-1965-045-4`.

27. KNOP, Dušan. Partitioning graphs into induced subgraphs. *Discrete Applied Mathematics*. 2020, vol. 272, pp. 31–42. ISSN 0166-218X. Available from DOI: `https://doi.org/10.1016/j.dam.2019.01.010`. 15th Cologne–Twente Workshop on Graphs and Combinatorial Optimization (CTW 2017).

28. FIALA, Jiří; GOLOVACH, Petr A.; KRATOCHVÍL, Jan. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theoretical Computer Science*. 2011, vol. 412, no. 23, pp. 2513–2523. ISSN 0304-3975. Available from DOI: `https://doi.org/10.1016/j.tcs.2010.10.043`.

29. *Parameterized Algorithms and Computational Experiements (PACE)* [online]. 2023 [visited on 2023-04]. Available from: `https://pacechallenge.org/`.

30. GUROBI OPTIMIZATION, LLC. *Gurobi Optimizer Reference Manual* [online]. 2023 [visited on 2023-04]. Available from: `https://www.gurobi.com`.

31. TECHNOLOGY, KTH Royal Institute of. *KACTL* [software]. GitHub, 2021 [visited on 2023-04]. Available from: `https://github.com/kth-competitive-programming/kactl`.

32. GOOGLE LLC. *GoogleTest* [software]. 2023. Version v1.13.0 [visited on 2023-04]. Available from: `https://github.com/google/googletest`.

Appendix A

Acronyms

FPT Fixed-parameter tractable

MSO Monadic second order logic

Appendix B

Measurements

	n	m	vc	q	time[ms]
0	10	12	2	2	1.3
1	30	38	2	2	2.1
2	50	66	2	2	2.1
3	70	91	2	2	4.0
4	90	116	2	2	8.1
5	100	132	2	2	10.4
6	10	13	3	3	3.1
7	30	51	3	3	3.2
8	50	88	3	3	4.3
9	70	124	3	3	6.3
10	90	162	3	3	9.7
11	100	179	3	3	13.0
12	10	17	4	3	1.3
13	30	65	4	3	6.2
14	50	114	4	4	21.3
15	70	162	4	4	16.3
16	90	211	4	4	42.5
17	100	234	4	4	52.6
18	10	21	5	3	1.8
19	30	81	5	4	10.7
20	50	141	5	4	20.8
21	70	203	5	5	54.7
22	90	261	5	4	52.7
23	100	292	5	4	70.0

	n	m	vc	q	time[ms]
24	10	23	6	3	1.6
25	30	95	6	4	9.2
26	50	167	6	4	19.5
27	70	239	6	4	72.0
28	90	311	6	5	236.8
29	100	347	6	5	249.0
30	10	24	7	3	0.9
31	30	108	7	4	14.4
32	50	193	7	4	24.7
33	70	276	7	5	246.6
34	90	360	7	5	905.6
35	100	402	7	5	438.9
36	10	25	8	3	0.7
37	30	121	8	4	83.0
38	50	218	8	5	310.8
39	70	313	8	5	564.9
40	90	409	8	5	2036.7
41	100	457	8	5	1664.7
42	10	26	9	3	1.1
43	30	134	9	4	26.0
44	50	242	9	5	603.2
45	70	350	9	5	3954.9
46	90	458	9	5	830.8
47	100	512	9	5	3636.4

Table B.1: Selected small instances used in the evaluation process. Graphs have 10, 40, 70 or 100 vertices and were generated with edge rate parameter $d = 0.6$. The upper bound of the vertex cover number is shown in the `vc` column and the column q shows the induced star partition number of the generated graph. The running time is displayed in ms.

	n	m	d	vc	time[s]
0	70	132	0.2	10	71.0
1	90	177	0.2	10	105.6
2	110	220	0.2	10	128.4
3	130	263	0.2	10	169.8
4	150	302	0.2	10	236.0
5	70	258	0.4	10	22.7
6	90	338	0.4	10	48.9
7	110	419	0.4	10	145.1
8	130	499	0.4	10	239.3
9	150	579	0.4	10	310.6
10	70	142	0.2	11	432.9
11	90	191	0.2	11	630.8
12	110	236	0.2	11	712.0
13	130	281	0.2	11	1139.1
14	150	329	0.2	11	-
15	70	281	0.4	11	226.7
16	90	369	0.4	11	292.8
17	110	457	0.4	11	607.5
18	130	546	0.4	11	1065.6
19	150	633	0.4	11	1119.7
20	70	153	0.2	12	-
21	90	205	0.2	12	-
22	110	254	0.2	12	-
23	130	304	0.2	12	-
24	150	350	0.2	12	-
25	70	304	0.4	12	315.1
26	90	401	0.4	12	476.1
27	110	496	0.4	12	597.5
28	130	592	0.4	12	-
29	150	688	0.4	12	-

	n	m	d	vc	time[s]
0	70	387	0.6	10	4.5
1	90	507	0.6	10	20.4
2	110	627	0.6	10	20.1
3	130	747	0.6	10	26.4
4	150	867	0.6	10	47.6
5	70	422	0.6	11	12.6
6	90	554	0.6	11	28.0
7	110	686	0.6	11	59.7
8	130	818	0.6	11	59.9
9	150	950	0.6	11	85.5
10	70	456	0.6	12	33.5
11	90	600	0.6	12	38.6
12	110	744	0.6	12	50.9
13	130	888	0.6	12	398.1
14	150	1032	0.6	12	103.9
15	70	490	0.6	13	36.1
16	90	646	0.6	13	80.5
17	110	802	0.6	13	238.0
18	130	958	0.6	13	216.6
19	150	1114	0.6	13	1047.1
20	70	524	0.6	14	83.6
21	90	692	0.6	14	62.2
22	110	860	0.6	14	338.5
23	130	1028	0.6	14	-
24	150	1196	0.6	14	-
25	70	558	0.6	15	55.9
26	90	738	0.6	15	84.2
27	110	918	0.6	15	304.3
28	130	1098	0.6	15	-
29	150	1278	0.6	15	-

Table B.2: Selected instances with bigger vertex cover used in the evaluation process. The left column contains data with edge rate parameter $d < 0.6$ and vertex cover $vc \geq 12$ and the right column contains graphs generated with $d = 0.6$. The displayed time is in seconds. Instances that did not terminate within designated 20 minutes are shown as -.

Appendix C

Contents of enclosed medium

```
readme.txt ......... the file with medium contents description
├── src ............ the directory of source codes
├── text ........... the directory of LaTeX source codes of the thesis
├── extra .......... tables of measured data
└── thesis.pdf ..... the thesis text in PDF format
```