ETH zürich

Department of Computer Science
Algorithms and Didactics

Neighborly Help for the Feedback Vertex Set Problem

Samuel Aggeler

Bachelor's Thesis

2023

Supervisors: Prof. Dr. Dennis Komm
Dr. Hans-Joachim Böckenhauer
Dr. Manuela Fischer

Abstract

The goal of this Bachelor's thesis is to study the minimum feedback vertex set problem in the model of neighborly help. The minimum feedback vertex set of a given graph is a smallest set of vertices, whose removal makes the graph cycle-free. It is known that the problem is NP-hard [14] and we thus make use of the neighborly-help model to see how much additional information is needed to solve the minimum feedback vertex set in polynomial time. The neighborly help model allows us to query an oracle that returns solutions for locally modified instances of the original instance.
In this thesis, we look at the following modifications for a graph $G = (V, E)$: Inserting an edge into G, deleting a vertex in G, deleting an edge in G, and contracting an edge in G.
We show that, for edge insertion and vertex deletion, we can bound the number of queries to the oracle by the length of a shortest cycle in G. For edge deletion and edge contraction, we show that certain instances can be solved in polynomial time and, for the remaining instances, we propose an improved exponential-time algorithm with the help of an iterative compression algorithm.

Acknowledgment

I want to thank Prof. Dr. Komm, Dr. Hans-Joachim Böckenhauer and Dr. Manuela Fischer.

Dr. Hans-Joachim Böckenhauer provided me with introductory material and helped me with choosing a topic for this bachelor thesis. Dr. Manuela Fischer and him gave me valuable feedback and insights for the results discussed in this thesis. Further, they gave me a lot of advice on how to write and structure this thesis. I also want to thank them for proofreading the entire thesis several times.

Finally. I want to thank Prof. Dr. Komm for being the official supervisor of this bachelor thesis.

Contents

Chapter 1

Introduction

Graph theory is a widely used concept in a lot of different domains. When we, for example, pay with our card for a bar of chocolate in a store, there is a series of steps involved. In computer science, the entire process of all these small steps is called transaction processing. Only when all these small steps succeed, the transaction goes through. If any of them fails, the entire process is aborted and rolled back to the state before the transaction started, and our card is declined. For efficiency reasons, these transactions are issued concurrently, and two processes may want to access the same resource and block each other by doing so. Neither of the processes can move forward, but also no error occurs that would allow for a simple rollback. Such a situation is called a deadlock. Detecting deadlocks is no trivial task as can for example be seen in [15]. To keep track of all the resources, they are stored in a graph as vertices. Processes are also stored as vertices and when a process wants to access a resource, there is an edge between them. Further, if a resource is assigned to a process, there is also an edge. Now, if we want to detect deadlocks in this so-called resource allocation graph, we are looking for cycles. The goal is to keep the graph acyclic and remove (abort) as few vertices (processes) as possible. Finding a set of these vertices in a graph whose removal leads to the destruction of all cycles is the main problem investigated in this thesis. In graph theory, we call a set of vertices that destroys all cycles of a graph a *feedback vertex set*. Usually, we want to remove as few vertices as possible and therefore the corresponding optimization problem is the *minimum feedback vertex set problem*.

Since it is known that the minimum feedback vertex problem is NP-hard, which was proven by Karp [14], and little progress is made toward the question N vs NP, we are curious about the additional amount of information that is required to solve the minimum feedback vertex set in polynomial time. For modeling a special kind of additional information, we use the neighborly help model which allows the algorithm to use arbitrarily many queries to an oracle that provides solutions for neighboring instances. Neighboring instances are locally modified instances of the original graph, e.g., the graph after inserting or deleting a vertex or an edge.

We start by introducing some graph-theoretical notation, formally defining the minimum feedback vertex set problem and thoroughly introducing the neighborly help model. Further, we give an overview of related work for the neighborly help model. In Chapter 2, we consider the local modification of inserting an edge into a graph. We design an algorithm that can solve the minimum feedback vertex set with

a bounded number of queries in polynomial time. In Chapter 3, we present a similar result for the local modification of vertex removal. In Chapter 4, we consider the removal of an edge as the local modification and present an algorithm that solves the problem in polynomial time for graphs that meet certain conditions. In the case that the conditions are not met, we improve the running time of an already existing branching algorithm. Finally, in Chapter 5, we look at the contraction of an edge and again present an algorithm that solves the minimum feedback vertex set problem in polynomial time for certain graphs and otherwise makes use of the improved branching algorithm.

1.1 Preliminaries

Before we formally introduce the neighborly help model we introduce some basic graph notation that is used throughout this thesis.

Definition 1.1. *A(n) (undirected) graph G is a pair (V, E) consisting of a finite non-empty set V of vertices and a set E of edges, where $E \subseteq \binom{V}{2} := \{\{x, y\} \mid x, y \in V,\ x \neq y\}$.*

Definition 1.2. *Consider a graph $G = (V, E)$.*

1. *For a vertex $u \in V$, we call the set $N_G(u) = \{v \in V \mid \{u, v\} \in E\}$ the* neighborhood *of the vertex v.*
2. *For a vertex $u \in V$, we call the natural number $deg(u) := |N_G(u)|$ the* degree *of the vertex u.*
3. *By $G[K]$, we denote the subgraph of G* induced *by $K \subseteq V$, i.e., $V(G[K]) = K$, and $E(G[K]) = \{\{x, y\} \in E \mid x, y \in K\}$.*
4. *A* path *is a sequence of vertices $v_1, v_2, \ldots, v_i$ where $(v_j, v_{j+1}) \in E$ for $j \in \{1, \ldots, i-1\}$.*
5. *A* simple path *is a path where each two vertices are pairwise distinct.*
6. *A* cycle *with length l is a path $v_1, \ldots, v_{l+1}$ with $v_1 = v_{l+1}$.*
7. *A* simple cycle *is a cycle $v_1, \ldots, v_{l+1}$, where $v_1, \ldots, v_l$ are pairwise distinct.*
8. *For an edge $e = \{u, v\}$, we say $e \in C$ if u and v are consecutive in C.*
9. *A* chord *of a cycle C is an edge $e = \{u, v\}$, where $u \in C$ and $v \in C$, but e is not in C.*
10. *A* tree *is a graph that does not contain any cycles and is connected.*
11. *A* forest *is a graph that does not contain any cycles.*
12. *We define the short form $G - v$ as the resulting graph $(V - \{v\}, E - \{\{u, v\} \mid u \in V\})$ when removing vertex v. Similarly, we define $G - e$ as the resulting graph $(V, E - \{e\})$ when removing an edge e.*

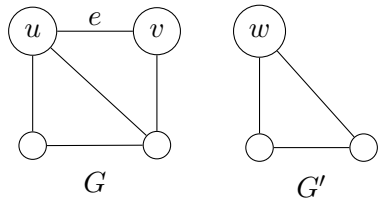


Figure 1.1. Edge contraction of an edge e in G and the resulting graph $G' := G/e$

13. *An* edge contraction *of an edge $e = \{u, v\}$ in a graph G, results in the graph G', where the vertices of G' are defined as the set of all vertices in G without u and v and a newly introduced vertex w. The edges of G' are all edges in G without e, and all edges previously incident to u and v are now incident to w. The process of introducing w with all its new incident edges we denote by contract(u, v). We define the short form G/e for the resulting graph G' by contracting the edge e in G. A corresponding sketch of an edge contraction can also be seen in Figure 1.1.*

Definition 1.3. *If we say a vertex v or an edge e* destroys *a cycle C, we mean that the deletion of v (and its adjacent edges) or the deletion of e removes an edge that is in C.*

Definition 1.4. *A* feedback vertex set *(FVS) is a set of vertices F, such that $G[V - F]$ is a forest. A* minimum FVS *is a FVS of minimum size. By $\alpha(G)$, we denote the cardinality of a minimum feedback vertex set for a graph G.*

Definition 1.5. *The* minimum feedback vertex set problem *(*MIN FVS*) on a graph G is the problem of finding a minimum feedback vertex set F for G.*

1.2 Neighborly-Help Model

The neighborly-help model was first introduced by Burjons et al. [8]. There it was used for the colorability problem and the vertex cover problem. Further, it was applied to the steiner tree problem in [13].
Since we know that MIN FVS is NP-hard, we use the *neighborly-help model* to see how much additional information we need to solve it in polynomial time. For the neighborly-help model, this additional information consists of solutions for neighboring instances that are provided by an oracle. More specifically, an algorithm is allowed to repeatedly send queries in the form of a locally modified instance to the oracle and the oracle will return a solution (i.e., a minimum FVS) for this locally modified instance in constant time. In the general setting, the model allows an arbitrary number of queries, but the goal is to minimize the number. The local modification and the corresponding neighboring instances vary from problem to problem. For the minimum feedback vertex set problem, we are considering the local modifications of edge insertion, vertex removal, edge removal and edge contraction. We formally define how these local modifications are applied to a given graph instance.

Definition 1.6. *Consider a graph $G = (V, E)$.*

- Edge-insertion query*: Ask oracle for a solution to an instance $G' = G + e$, where a single new edge e (not present in G) is added.*
- Vertex-removal query*: Ask oracle for a solution to an instance $G' = G - v$, where a single vertex v along with its incident edges is deleted.*
- Edge-removal query*: Ask oracle for a solution to an instance $G' = G - e$, where a single (existing) edge e is deleted.*
- Edge-contraction query*: Ask oracle for a solution to an instance $G' = G/e$, where a single edge e is contracted in G.*

We will later see that we can bound the number of queries for MIN FVS for the local modifications of edge insertion and vertex removal. For the local modifications of edge deletion and edge contraction, this only holds for certain graphs.

1.3 Related work

The idea to use additional information for NP-hard problems was already present in other models before the introduction of the neighborly-help model in 2019 [8]. In a classical setting, we consider a Turing machine with arbitrary advice bits on a second tape, which can be seen in [2]. We will highlight similarities and differences between the neighborly-help model and the models of auto-reducibility, self-reducibility and reoptimization. Further, we want to explain the notions of criticality and stability.

Auto-Reducibility and Self-Reducibility

A function is auto-reducible if there exists a Turing machine, that for a given input x, outputs $f(x)$ in polynomial time while having access to an oracle that returns $f(y)$ for any $y \neq x$ over some alphabet Σ. We notice that if we restrict the access to neighboring instances, we have the same setting as in the neighborly-help model. On the other hand, a function is self-reducible if there exists a Turing machine that for a given input x outputs $f(x)$ while having access to an oracle that returns $f(y)$ for y strictly smaller than x in some sense. This means that for example edge insertion would not be possible in a self-reducibility model. More about reducibility can be found in [10].

Reoptimization

The model of reoptimization is also closely related to the neighborly-help model. With the reoptimization model, we are also interested in neighboring instances. In the setting of reoptimization we have as input an instance, an optimal solution and a locally modified instance. The task in then to compute an optimal solution for the modified instance. An important difference between reoptimization and the neighborly-help model is that the neighboring instance is part of the input. This means that, we cannot choose the instance ourselves. On the other hand in the neighborly-help model we can query any number of neighbouring instance that we choose. Thus, the reoptimization model is less generous compared to the neighborly-help model. This means that hardness of the neighborly-help model implies hardness in the reoptimization problem. This implication does not hold in the other direction. Further, in reoptimization, the instance we want to figure out is the modified instance

instead of the original one which is the goal of the neighborly-help model. This means that, for example, edge deletion turns into edge insertion.

The model of reoptimization was embossed by Schäffter [16]. Archetti et al. [1] were the first to apply it and use it to improve the running time of an approximation algorithm for the metric traveling salesman problem. The reoptimization model showed even more potential in improving the approximation ratio of problems compared to the original setting. This was first shown by Böckenhauer et al. [5] for the metric traveling salesman problem with the local modification of changing the weight of an edge. Around the same time, Ausiello et al. [3] presented improved approximation results for removing or adding a vertex for the traveling salesman problem under consideration of the reoptimization model. An overview of already existing results can be found in [6]. Further, the reoptimization model has been combined with parameterized problems. There we use it to compute solutions to parameterized problems and improve the size of kernels. Especially, it is possible to show that certain kernels can be improved to polynomial size with the reoptimization model, which is not possible under standard assumptions. An example of this has been presented by Böckenhauer et al. [4].

Criticality and stability

The term criticality was introduced by Dirac in 1952 [9]. It was first used for colorability in combination with vertex deletion. Later the concept was generalized to arbitrary graph modifications by Wessel [17]. Colorability is still prevalent when it comes to criticality. Further, a graph G is called critical if every proper subgraph of G has a smaller chromatic number, meaning that the removal of any vertex or edge would decrease the chromatic number of G. We will introduce a similar concept for the minimum FVS problem with respect to edge deletion. Later Frei et al. [11] introduced the term stability, which expands on criticality. We say that an edge e is ξ-stable with respect to a graph number ξ if deleting e does not change ξ. If ξ changes by removing e, we say e is ξ-critical. Stability is analogously defined for vertices. We notice that every edge and vertex in a graph is either stable or critical with respect to some graph number.

1.4 Basic Observations

Before we start with the main results of this thesis, we state some simple observations that we made in the process.

It already follows from Definition 1.1, but we want to explicitly state that we do not consider multigraphs and graphs with self-loops. Further, for the sake of always being able to insert an edge, we do not consider complete graphs. This is without loss of generality, as it is easy to find a minimum FVS for a complete graph.

We noticed that having an algorithm that can find a shortest cycle in a graph is a useful subroutine.

The basic idea behind the algorithm is to explore the graph via breadth-first search (BFS), starting from each vertex v once. For each run of BFS, when we at some point encounter an edge that leads back to the start vertex, we know that this edge closes a cycle containing v. By this procedure, we find the shortest cycle containing the start vertex v. By running the BFS from every vertex we are guaranteed to find a

shortest cycle in the graph. An overview of the algorithm can be seen in Algorithm 1. The running time of a single BFS iteration is $\mathcal{O}(|V| + |E|)$. Since we are running these iterations $|V|$ times we have a total running time of $\mathcal{O}(|V|(|V| + |E|))$. Besides for finding a shortest cycle, we also use this algorithm to check if a graph is acyclic. If the algorithm terminates without finding a cycle, we are guaranteed to have an acyclic graph. This proves helpful in checking whether a set is a FVS for a given graph. To this end, we can remove the vertices in the set from the graph and check the remaining subgraph for cycles.
We want to elaborate on a few special cases that allow us to neglect some special cases for the rest of the thesis.
For edge deletion queries, we assume that every edge e is part of at least one cycle, since otherwise we can simply query the oracle with the neighboring instance $G - e$ and find a minimum FVS for G in one query, as the minimum FVS for $G - e$ is also a minimum FVS for G.
We do not consider graphs with more than one connected component. We outline how we can solve graphs with more than one component in two queries within the neighborly-help model. We start by identifying two different connected components K_1 and K_2. We modify K_1 resulting in G_1, with the oracle solution S_1. Then we make a second query where we modify K_2 to get a minimum FVS S_2 for G_2. We then return $S_1 - K_1 + S_2 - K_2$. Because we get a minimum FVS for each connected component where we did not modify anything, the returned set is a minimum FVS for the entire graph. Thus, we can always solve the minimum FVS for graphs with more than one connected component in two oracle queries.

Algorithm 1 Shortest cycle in graph

Input: Arbitrary graph G
Output: A shortest cycle in G or NO if no cycle is present

```
for each vertex v in V do
    start BFS from v until back-edge to v is found or entire graph is explored
    if found cycle is shorter than current shortest cycle then
        update shortest cycle
if no cycle found then
    return NO
else
    return shortest cycle
```

Chapter 2

Edge Insertion

In this section, we look at the result using edge insertion queries. As already defined, for edge insertion we are allowing the insertion of a single edge e in a graph G, where $e \notin E$. The main result is an algorithm that finds a minimum FVS for a triangle-free graph in $\lceil \frac{l}{2} \rceil$ queries where l denotes the length of a shortest cycle in G. The main observation used for said algorithm is that by inserting an edge e with an incident vertex v that belongs to at least one minimum FVS of G, the returned oracle solution F' for the instance $G' = G + e$ is also a minimum FVS for G. It thus only remains to find such a vertex. We argue that every cycle C must have at least one such vertex v. By inserting chords to a cycle, we are guaranteed to eventually find such a vertex and hence a minimum FVS for G. The reason behind only considering triangle-free graphs is that we do not want to introduce multiedges by inserting chords in a cycle. We outline how the algorithm works on a high-level. Further, an overview can be found in Algorithm 2. The algorithm starts by finding a shortest cycle in a given triangle-free graph G. If the shortest cycle algorithm returns "NO", the algorithm simply returns the empty set as a minimum FVS for G since no cycles are present in G. Assuming the algorithm finds a shortest cycle C_{min} the algorithm iteratively inserts chords between vertices of C_{min} until every vertex was at least once the endpoint of a chord. For every such modified graph, it compares the size of the oracle solution. As soon as it has two oracle solutions with different sizes, it stops and returns the smaller of the two. If every oracle solution has the same size, the algorithm returns an arbitrary oracle solution.

Algorithm 2 Minimum FVS with edge-insertion queries

```
Input: Arbitrary triangle-free graph G
Output: Minimum FVS of G
C_min ← FindShortestCycle(G)
for each non-consecutive vertices u and v in C_min do
    e = {u, v}
    G' = G + e
    S' ← oracle solution for G'
    if S' is smaller than any previously seen oracle solution then
        return S'
return S'
```

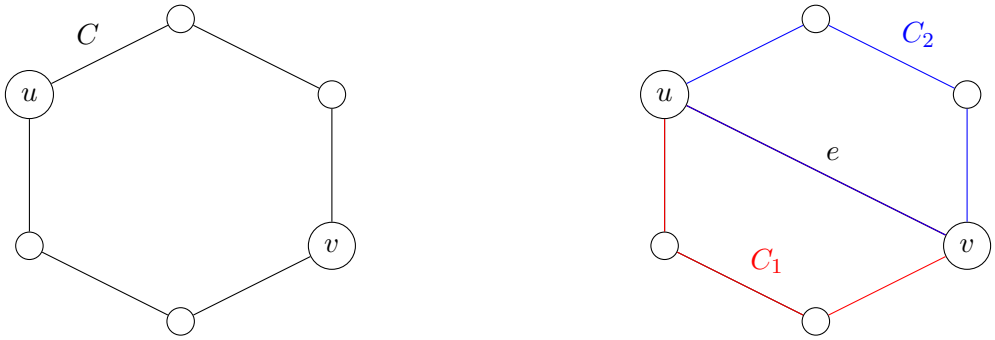


Figure 2.1. Sketch of newly introduced cycles C_1 and C_2 by inserting an edge into an already existing cycle C

We now establish all lemmas that are necessary to prove the correctness of the above-mentioned algorithm. First, we observe that by inserting a new chord $e = \{u, v\}$ into an existing cycle C, we introduce at least two new cycles and each of those newly introduced cycles must contain e. Hence, by either deleting u or v, we destroy every such cycle.

Lemma 2.1. *For a given graph $G$and a cycle C, inserting a chord $e = \{u, v\}$ between u and v in C, introduces at least two new cycles C_1 and C_2. The deletion of either u or v destroys every such newly introduced cycle.*

Proof. We refer to Figure 2.1 to see that by inserting e into C we introduce C_1 and C_2. Further, every newly introduced cycle in G must contain e. The deletion of either u or v deletes e and thus every cycle which was introduced by adding e to G.□

With Lemma 2.1 we can ensure that all new cycles we created by adding an edge e into G can be destroyed by removing one of the endpoints of e.
Next, we are looking for a way to ensure that one of the endpoints of the newly inserted edge e belongs to a minimum FVS F for G. We could of course try to insert an edge e between two non-neighbouring vertices such that for every vertex v we insert at least one incident edge. This would take at least $\lceil \frac{|V|}{2} \rceil$ edge insertions. With an additional observation, we can reduce the number of queries substantially. Instead of inserting edges between arbitrary vertices, we realize that every cycle C has at least one vertex that is in a FVS, so it is enough to focus on one cycle instead.

Lemma 2.2. *For every cycle C in a given graph G and every minimum FVS F of G, at least one vertex v in C is part of a F.*

Proof. Assume towards a contradiction that there exists a minimum FVS F for a graph G and a cycle C where $v \notin F$ for all $v \in C$. Then C is still present in G after the removal of every vertex in F, and therefore it is not a FVS for G. □

Lemma 2.2 gives us now a better way to insert an edge e into a graph G, where one of its endpoints belongs to a minimum FVS of G. Namely, we can search for a cycle C in G, and we are guaranteed that at least one vertex $v \in C$ is in a minimum FVS. Further, since G is a triangle-free graph. We thus can guarantee that we are always able to insert an edge between two non-consecutive vertices.

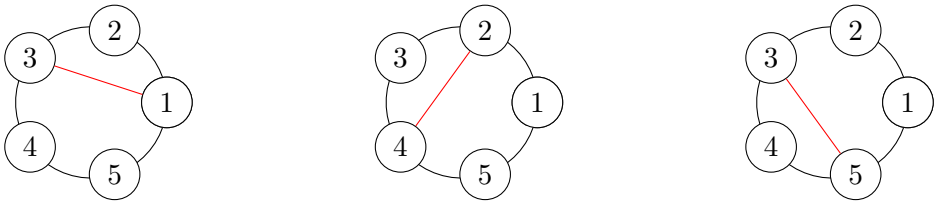


Figure 2.2. Example insertion of edges for a cycle of length 5

Next, we formally show that F is also a minimum FVS for the modified graph $G' = G + e$.

Lemma 2.3. *A minimum FVS F of a graph G is also a minimum FVS for the modified graph $G' = G + e$ where $e = \{u, v\} \notin E(G)$ if $u \in F$ or $v \in F$.*

Proof. Since we want to prove that F is a minimum FVS for G', we show that every cycle C in G' is destroyed by the deletion of F. We consider a cycle C in G' and make a case distinction over whether the newly introduced edge e is in C or not:

Case 1. The edge e is part of C. Then, we know that u or v destroys C. Since either u or v are part of F, C is destroyed by F.

Case 2. The edge e is not part of C. Then C already existed in G. Since F is a minimum FVS for G, there must exist a vertex v in F which destroys C. □

Before reaching the main result of this chapter in the form of an algorithm, we need a criterion to identify the oracle solution that is also a minimum FVS for our original graph G. In the case that at some point we have two oracle solutions with different cardinalities we can stop and return the smaller solution. Intuitively, the oracle solution of the modified graph G' will always be at most larger by one, since all the newly introduced cycles from adding e can be deleted by adding either of the endpoints of e. As soon as we have a smaller solution the addition of either endpoint was not necessary.

Lemma 2.4. *By adding an edge $e = \{u, v\}$ to a graph G, the size of the resulting minimum FVS F' for the graph $G' = G + e$ will at most be one larger than the cardinality of a minimum FVS F for G.*

Proof. We already know from Lemma 2.1 that every newly introduced cycle in G' can be destroyed by either u or v. Hence, the sets $F + \{u\}$ or $F + \{v\}$ will always be a FVS for G'. □

Thus, since we know that the oracle solution can at most be larger by one compared to a minimum FVS for G, we know that if we encounter two different solutions the smaller one is also a minimum FVS for G.

Lemma 2.5. *Consider two oracle solutions F' and F'' for two modified graphs $G' = G + e$ and $G'' = G + e'$. If $|F'| < |F''|$, then F' is a minimum FVS for G.*

Proof. We know by Lemma 2.5 that the cardinality of the oracle solution can at most be larger by one for a single edge insertion. Clearly, we cannot reduce the size

of a minimum FVS by adding an edge. It follows that $|F''| = |F'| + 1$ and therefore F' has the same cardinality as a minimum FVS F for G. Further, since G' contains all cycles that are also in G, the set F'' destroys every cycle in G. □

In some cases, we will have inserted a chord incident to every vertex in a cycle C but all the oracle solutions have the same size. The intuition in this case is that every single vertex in C belongs to a minimum FVS for G, and therefore we can not distinguish between the sizes of the oracle solutions. Instead, we know that in this case, we can pick an arbitrary oracle solution F, which will be a minimum FVS for G.

Lemma 2.6. *Assume that in a given graph G, we inserted a chord for every vertex in a cycle C and all the returned oracle solutions have the same size. Then an arbitrary oracle solution F is a minimum FVS for G.*

Proof. We know by Lemma 2.2 that at least one vertex in C belongs to a minimum FVS. Further, from Lemma 2.3 it follows that at least one of the oracle solutions must be a minimum FVS for G. Since every oracle solution has the same size, we know that all the oracle solutions have the same size as a minimum FVS for G. Again, since any modified instance contains every cycle that is present in G, it follows that every such oracle solution is a minimum FVS for G. □

Finally, we want to conclude this chapter with an analysis of the presented algorithm.

Theorem 2.1. *For a triangle-free graph G, we can find a minimum FVS F in polynomial time with $\lceil \frac{l}{2} \rceil$ edge-insertion queries, where l is the length of a shortest cycle C in G.*

Proof. We start with the correctness of the algorithm. If G does not contain any cycles the algorithm Algorithm 1 will return "NO" and we simply return the empty set as a minimum FVS for G. By iteratively inserting a chord e between two non-consecutive vertices, we will at some point insert a chord where at least one endpoint belongs to a minimum FVS for G. This follows from Lemma 2.2. Further, we are always able to insert these chords without introducing a multigraph since we are looking at triangle-free graphs and hence every shortest cycle has a length of at least 4. This allows us to upper bound the number of edge-insertion queries by $\lceil \frac{l}{2} \rceil$, where l is the length of C. The correctness of every step of the algorithm has been shown in the Lemmas prior, and therefore we conclude that the algorithm is correct.
Since queries to the oracle take constant time, the running time of the algorithm is dominated by finding the shortest cycle which can be done in $\mathcal{O}(|V|(|V| + |E|))$. Further, we need at most $\lceil \frac{l}{2} \rceil$ queries before we can return the optimal solution. We need to round up the number of queries in case the shortest cycle has an odd number of vertices, and we need to insert a chord where one of the two endpoints already has been used for another chord. □

Chapter 3

Vertex Deletion

In this section, we look at vertex deletion queries. As already defined, for vertex deletion, we are allowing the deletion of a single vertex v in a graph G. We design an algorithm that finds a minimum FVS for G with at most l vertex deletion queries, where l denotes the length of a shortest cycle in G. The main observation used for the algorithm is the fact that if v belongs to at least one minimum FVS F of G, the oracle solution of the resulting graph $G' = G - v$ returns a minimum FVS F' which is smaller by one compared to a minimum FVS for G. We combine this observation with the fact that by adding v to the oracle solution F' for a modified graph $G' = G - v$ is an FVS for the original instance G. As with edge insertion, we again search for a shortest cycle in G to minimize the number of queries needed. With these observations, we design an algorithm that for a graph G deletes a vertex in a shortest cycle C and queries the oracle until two of the returned oracle solutions differ by one. The algorithm then takes the smaller of the two solutions and adds back the deleted vertex. The corresponding pseudocode is summarized in Algorithm 3.

We start by showing that we can construct a FVS from an oracle solution by adding back the deleted vertex of a vertex-deletion query.

Algorithm 3 Minimum FVS with vertex-deletion queries

```
Input: arbitrary graph G
Output: minimum FVS for G
C_min ← FindShortestCycle(G)
for each vertex v in C_min do
    G' ← G − v
    S' ← oracle solution for G'
    if S' is smaller than any previously seen oracle solution then
        return S' + v
return S' + v
```

Lemma 3.1. *Consider a minimum FVS F' of a graph $G' = G - v$. Then $F := F' + \{v\}$ is a FVS for G.*

Proof. We want to show that every cycle C in G is destroyed by F. We make a case distinction over whether the removed vertex v lies on C or not:

Case 1. The vertex v lies on C: Then, since v is in F,the cycle C is destroyed by v.

Case 2. The vertex v does not lie on C: Then v does not destroy C, thus C is still present in G'. Since F' is a minimum FVS for G', the cycle C is destroyed by some vertex in F'. □

With Lemma 3.1, we can construct a FVS F for the original instance which is at most bigger by one compared to a minimum FVS for G. In the next step, we want to show that the minimum FVS of a locally modified graph $G' = G - v$ is smaller by one compared to a minimum FVS for original instance G if v belongs to a minimum FVS for G. Before proving this claim, we want to show that removing a vertex from a graph decreases the cardinality of a minimum FVS for the introduced subgraph by at most one compared to a minimum FVS for the original instance. This will be useful to guarantee that the FVS constructed by the algorithm will have the same size as a minimum FVS.

Lemma 3.2. *Removing a vertex v from a graph G implies that the cardinality of a minimum FVS F' of G' can be smaller at most by one compared to the size of a minimum FVS for G.*

Proof. Consider a graph G with a minimum FVS F and a vertex v in V. Assume towards a contradiction that the minimum FVS F' for $G' = G - v$ is smaller by two compared to F. It follows from Lemma 3.1 that $F' + \{v\}$ is a FVS for G and by our assumption still one smaller than F. This contradicts that F is a minimum FVS for G. Hence, F' can be smaller at most by one compared to F. □

Now, that we know that the resulting graph $G = G - v$ has a minimum FVS that is at most smaller by one compared to G, we want to investigate which conditions have to hold such that we find such a minimum FVS.

Lemma 3.3. *Consider a vertex v that belongs to at least one minimum FVS F of G. Then a minimum FVS F' for $G' = G - v$ satisfies $\alpha(G') = \alpha(G) - 1$.*

We now formalize the intuition that the operation of vertex removal is the same as what is done with every vertex in a FVS. Thus, if we remove a vertex in G that belongs to a minimum FVS F for G, the subgraph $G' = G - v$ has a minimum FVS of a size one smaller compared to F.

Proof. Consider the minimum FVS F that contains v. Since v is no longer in G', every cycle that is destroyed by v is not in G'. Thus, $F' := F - \{v\}$ is a FVS for G'. Indeed, if we look at all the cycles in the graph G'', where G'' is the resulting graph from removing every vertex in F', we can convince ourselves that every cycle in G'' is no longer present in G' and thus F' is a FVS for G'. Lastly, since the cardinality of F can decrease at most by one by Lemma 3.2, F' is a minimum FVS for G'. □

We now know that by adding a deleted vertex v back to the minimum FVS F' for $G' = G - v$, we get a FVS F for the original instance G. Further, if v belongs to a minimum FVS of G, we know that $\alpha(G') = \alpha(G) - 1$. We want to prove now that $F + \{v\}$ is a minimum FVS for G.

Lemma 3.4. *Consider a graph G with a minimum FVS F and a modified instance $G' = G - v$ with a minimum FVS F', where $|F| > |F'|$ holds. The set $F' + \{v\}$ is a minimum FVS for G.*

Proof. We know from Lemma 3.3 that $|F| = |F'| + 1$. Further, from Lemma 3.4 we know that by adding v to F' the corresponding set is a FVS for G. Hence, $F' + \{v\}$ is a minimum FVS for G. □

Now we know that if we have two oracle solutions of different sizes, we can construct a minimum FVS for the original instance. But what about the case where every oracle solution has the same size? Before we continue with this case, we introduce the notion of a *critical cycle*. Intuitively, a critical cycle C is a cycle where we need a vertex v in C to be in a FVS solely for the destruction of C. In other words, every other cycle in G is removed by $F - \{v\}$ and we add v to F just for C.

Definition 3.1. *A cycle C is a **critical cycle** in a graph G if there exists a minimum FVS F with v in F and $F - v + u$ is also a minimum FVS for u in C and $v \neq u$.*

For every vertex v in a critical cycle C, we have at least one minimum FVS F that contains v. A problem that arises if the algorithm finds a critical cycle as a shortest cycle is that every oracle solution will have the same size. We can solve the problem rather easily by choosing an arbitrary oracle solution F' of a modified graph $G' = G - v$ and returning $F' + \{v\}$. For an arbitrary vertex v in C, we now want to prove that the returned solution is also a minimum FVS for the original instance G.

Lemma 3.5. *Consider a critical cycle C and an arbitrary vertex v in C. If F' is a minimum FVS for $G' = G - v$, then $F' \cup \{v\}$ is a minimum FVS for G.*

Proof. By the definition of a critical cycle C we know that no matter which vertex v we remove from C, v is part of at least one minimum FVS F for the original instance G. Thus, by Lemma 3.3 it holds that $|F| > |F'|$. Hence, Lemma 3.4 is applicable, which concludes the proof. □

We now know how to construct a FVS for G and by Lemma 2.2 we know that in every cycle there is at least one vertex that belongs to any minimum FVS. Thus we can construct a minimum FVS for G from an oracle solution F' by Lemma 3.4. Thus, we have established all lemmas necessary to prove the correctness of the proposed algorithm. We conclude this section with the analysis of the algorithm.

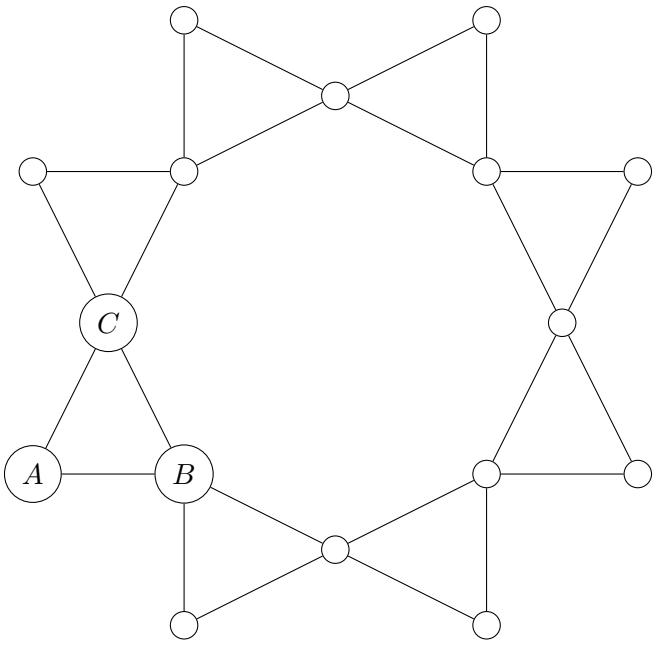
Theorem 3.1. *We can find a minimum FVS F for a graph $G = (V, E)$ in polynomial time with l vertex deletion queries where l is the length of a shortest cycle C in G.*

Proof. We start with the correctness of the algorithm. Unless G does not contain any cycles we will always be able to find a shortest cycle in G with the algorithm Algorithm 1. Further, if no cycle is present in G, the algorithm will return "NO" and we simply return an empty set as the minimum FVS. Now by Lemma 2.2, there exists at least one vertex v in C that belongs to a minimum FVS F. If the algorithm deleted every vertex in C once, it is guaranteed to either have two oracle solutions with different sizes or $|C|$ oracle solutions with the same size. In the first case, the algorithm returns the smaller oracle solution F' with the corresponding deleted

vertex v', which is correct according to Lemma 3.4. In the second case, the shortest cycle is by definition a critical cycle and the algorithm returns an arbitrary solution F'' with its corresponding vertex v'', which is a minimum FVS for G by Lemma 3.5. In either case, the algorithm returns a minimum FVS for G.
Since we assume that queries to the oracle can be done in constant time, the running time is dominated by finding the smallest cycle in G, which can be done in $\mathcal{O}(|V|(|V| + |E|))$. Further, we need at most l queries before we can return the optimal solution. □

Before concluding this chapter, we provide an example of a graph G for which the algorithm requires exactly l queries.

Figure 3.1. Example of a hard instance for the proposed algorithm



The basic idea behind the hardness of a given graph for the algorithm is that no matter which shortest cycle the algorithm chooses, there exists exactly one vertex which decreases the size of the oracle solution. In this case, the algorithm cannot terminate early and always has to check every single vertex v in C.

Lemma 3.6. *The algorithm proposed in Theorem 3.1 needs l vertex deletion queries to return a minimum FVS for the graph in Figure 3.1.*

Proof. We break down the analysis to a single triangle of Figure 3.1. When the algorithm finds a cycle, it deletes vertices and compares the size of the oracle solutions until it finds oracle solutions with different sizes. Assuming that the algorithm deletes first the vertex A and then the vertex B we get both times an oracle solution of size four. When we delete vertex C, we get a solution of size three and the algorithm terminates. Thus, it takes a total of three queries until we find a minimum FVS for the graph in Figure 3.1. By the same argument we can increase the size of the cycles by an arbitrary amount and reach the same conclusion, meaning that in the worst case we always have l queries where l denotes the length of a shortest cycle. □

Chapter 4

Edge Deletion

In this section, we look at edge deletion queries. As already defined in Chapter 1, we allow the deletion of a single existing edge e in $E(G)$ for a given graph G. We present an algorithm that returns a minimum FVS for every graph with a critical cycle as is defined in Definition 3.1, in at most $|E|$ queries. If no critical cycle exists, the algorithm uses a single step of an improved branching algorithm as a subroutine to find a minimum FVS for G in time $\mathcal{O}(4^k \cdot n^3)$, where k denotes the size of a minimum FVS for G. The algorithm presented in Algorithm 4 iteratively deletes every edge once and checks if it can find two solutions of different sizes. If it finds two such solutions, it returns the smaller of the two and adds back one of the endpoints of the deleted edge. If it does not find such two oracle solutions, it checks if the size of the oracle solution is always one smaller than the size of a minimum FVS for the original instance or if all the oracle solutions have the same size as a minimum FVS for the original instance by using a single iteration of the compression algorithm. In the first case, we can return an arbitrary oracle solution with one of the endpoints of the corresponding deleted edge. In the second case, it constructs a FVS for the original instance and decrease the size of it by one with the improved branching algorithm. A high-level overview of the algorithm can be found in Algorithm 4.

Algorithm 4 Minimum FVS with edge-deletion queries

```
Input: arbitrary graph G
Output: Minimum FVS of G
for each edge e in E(G) do
    u ← arbitrary endpoint of e
    G' ← G − e
    S' ← oracle solution for G'
    if S' is smaller than any previously seen oracle solution then
        return S' + u
FVS ← S' + u
output ← ImprovedBranchingAlgo(G, size of oracle solution, FVS)
if ImprovedBranchingAlgo returns no then
    return FVS
else
    return output of ImprovedBranchingAlgo
```

First, we establish how we can construct a FVS from an oracle solution by adding back one of the endpoints of the deleted edge.

Lemma 4.1. *Consider an oracle solution F' of a graph $G' = G - e$. Then $F = F' + \{v\}$, where v is an arbitrary endpoint of e, is an FVS for G.*

Proof. We want to show that every cycle C in G is destroyed by F. We make a case distinction over whether the deleted edge e lies on C or not.

Case 1. The edge e is not part of C: Then, C is present in G'. Since F' is a minimum FVS for G', there exists a vertex in F' that destroys C.

Case 2. The edge e is part of C: If e is part of C, then both endpoints of e lie on C. Since we add either of the endpoints to F, the set destroys C. □

Thus, we know how to construct a FVS for the graph from an oracle solution. By deleting an edge, we surely do not increase the size of the minimum FVS for the instance. Thus, by adding an endpoint of the deleted edge back to the oracle solution, we can guarantee that the constructed solution is at most one larger than a minimum FVS.
We now want to see if we can guarantee a solution that is a minimum FVS for the original instance by getting rid of the plus one. We show what conditions need to be fulfilled to find an oracle solution that is smaller by one compared to the cardinality of a minimum FVS for the original instance. We start by formalizing an observation we make for every minimum FVS F in a graph G. Namely, every vertex in a given minimum FVS destroys at least one cycle which no other vertex in the same minimum FVS does.

Lemma 4.2. *(Minimality principle) For every vertex v in a minimum FVS F of a given graph G, we have that v destroys at least one cycle C which is not destroyed by any other vertex u in F with $v \neq u$.*

Proof. Assume towards a contradiction that we have a minimum FVS F containing a vertex v such that each cycle C which v lies on is destroyed by another vertex u with u in F and $v \neq u$. This is a contradiction with our assumption that F is a minimum FVS since we can omit v and still destroy every cycle in G with the removal of F.□

For future references, we will refer to this lemma as the *minimality principle* for the minimum FVS problem. We introduce the set K_v, for every vertex v in an arbitrary minimum FVS F. This set K_v contains all the cycles C which are uniquely destroyed by v.

We make an interesting observation, namely if we have a vertex v in a minimum FVS F, where v destroys exactly one cycle C uniquely. Then deleting any edge e from C results in that the neighboring graph $G' := G - e$ has a smaller minimum FVS compared to the minimum FVS for G.

Lemma 4.3. *If, for a minimum FVS F, there exists a vertex v with $|K_v| = 1$, then the removal of any edge e in C, for C being the cycle in K_v, leads to a FVS F' for $G' = G - e$ with $|F'| = |F| - 1$.*

Proof. We define the set $F' := F - \{v\}$. We want to show that every cycle C in $G' = G - e$ is destroyed by F'. We make a case distinction over whether the deleted vertex v lies on C or not:

Case 1. The vertex v lies on C: We know that besides a single cycle C_v every cycle that is destroyed by v is also destroyed by at least another vertex u which is still in F'. Further, since we deleted e, C_v is not present in G'.

Case 2. The vertex v does not lie on C: Then C must be destroyed by a vertex u in F different from v and hence is still destroyed by F'. □

We have proven that it is possible to delete an edge and the oracle solution of the corresponding graph $G' = G - e$ is smaller by one compared to the size of a minimum FVS for G. We now show that the minimum FVS of a graph $G' = G - e$ can at most be one smaller compared to the size of a minimum FVS for G.

Lemma 4.4. *The minimum FVS of a graph $G' = G - e$ is at most one smaller compared to a minimum FVS for G.*

Proof. Assume towards contradiction that the minimum FVS of G' is smaller by at least two compared to a minimum FVS for G. We define a graph $G'' = G - v$, where v is an arbitrary endpoint of e. From Lemma 3.3, we already know that the minimum FVS for G'' is at most smaller by one compared to a minimum FVS for G. Thus, a minimum FVS of G'' has cardinality at least $\alpha(G) - 1$. Since G'' also does not contain e and further $E(G'') \subseteq E(G')$ as well as $V(G'') \subset V(G')$. Hence, the graph G'' is a subgraph of G'. Since neither $G - e$ nor $G - v$ have a larger minimum FVS than G, F' cannot be smaller than F''. This is a contraction with our assumption that F' is smaller by at least two compared to the minimum FVS of the original instance. □

So far, we have seen that, whenever we have a vertex v that belongs to a minimum FVS with $|K_v| = 1$, that the graph $G' = G - v$ has a smaller minimum FVS compared to G. We now want to see what happens when this condition is not fulfilled. In other words, what happens if, for every vertex v and every minimum FVS F, it holds that $|K_v| \geq 2$? We only consider the subgraph that is constructed from K_v for all vertices v, since all the other cycles are destroyed by at least another vertex u. We show that the size of the minimum FVS for $G' = G - e$ is the same as for G, unless we have at least one minimum FVS F with a vertex v and $|K_v| = 1$. We consider the case $|K_v| = 2$ and look at all subgraphs that are possible with a vertex destroying exactly two cycles uniquely. We begin with the case that the cycles share a neighboring edge of v.

Lemma 4.5. *If an edge e is part of at least two simple cycles C and C_1 in a graph G then there exists at least a cycle C_2 that does not contain e.*

Proof. We outline the proof with a graphical representation of the cycles, which can be seen in Figure 4.1. By removing e we will destroy the cycles C_1 and C_2, but the cycle C_3 will remain intact. □

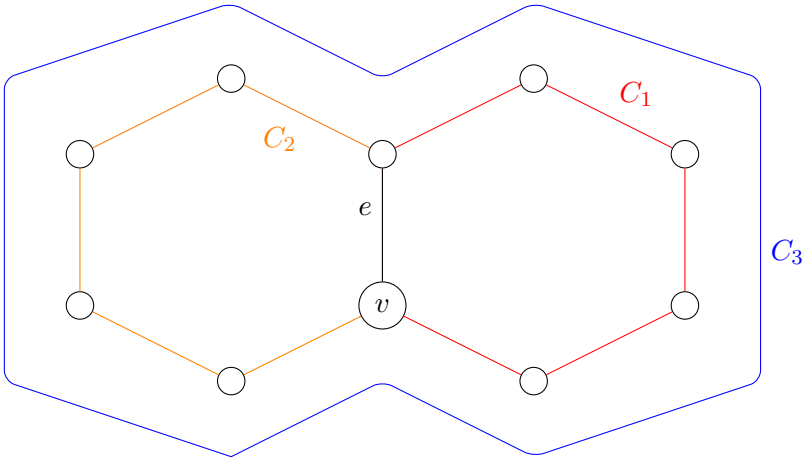


Figure 4.1. The edge e is part of two cycles C_1 and C_2 and the implied third cycle C_3

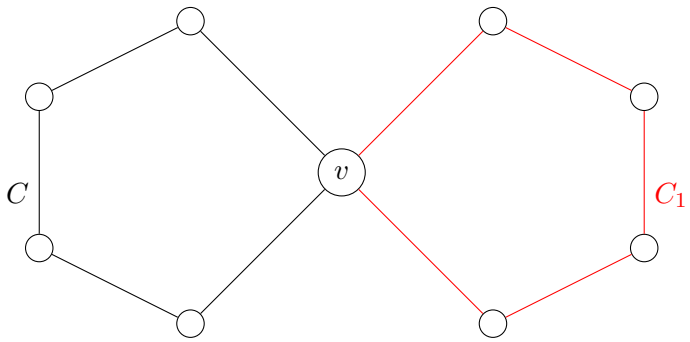


Figure 4.2. Two edge disjoint cycles C and C_1

We see that in this case, there always exists a third cycle. The removal of an arbitrary endpoint of e is enough to destroy all cycles. On the other hand, we can not destroy all three cycles with the removal of a single edge.
Let us now consider the other possible case.

Lemma 4.6. *Removing a neighboring edge from a vertex that is at least part of two edge-disjoint cycles C and C_1, does not destroy both cycles.*

Proof. As one can see in Figure 4.2, we would need to delete at least two edges to destroy both C and C_1. □

We conclude that, for $|K_v| = 2$, we cannot destroy every cycle in the subgraph G' which consists of all the cycles that are only destroyed by v. Further, for $|K_v| > 2$, we observe that every such graph will be a superset of the two graphs we considered and therefore the removal of a single edge e will not suffice to destroy all cycles in the entire graph. From this, it follows that we only decrease the cardinality of a minimum FVS by edge removal if there exists a minimum FVS F and a vertex v in F with $|K_v| = 1$. Next, we show that every such cycle is also a critical cycle.

Lemma 4.7. *If, for a given graph G, a minimum FVS F and a vertex v, we have $|K_v| = 1$, then the cycle C in K_v is a critical cycle.*

Proof. We have already shown in Lemma 4.3 that if $|K_v| = 1$ then the removal of any edge e in C is enough to guarantee that the minimum FVS of $G' = G - e$ is smaller by one compared to a minimum FVS for G. Further, if we consider the minimum F with v in it, we know that v is only in F because it destroys exactly the cycle C that is in K_v. Therefore, we can for every vertex u in C define the set $F' = F - \{v\} + \{u\}$ and F' is also a minimum FVS for G. Thus by definition, the cycle C is a critical cycle. □

From this, we conclude that, if the cardinality of the minimum FVS F decreases by removing an edge e, then e is part of a critical cycle.

We now know how we can find a minimum FVS if we have two oracle solutions with different sizes. Since the minimum FVS for a graph $G' = G - e$ certainly will not be larger than a minimum FVS for G, we can conclude, that if we have two oracle solutions with different sizes, the smaller of the two must be smaller by one compared to the minimum FVS of the original instance by Lemma 4.4. Thus we can simply add one of the endpoints of the deleted edge and have a minimum FVS for the original instance.
In the next step, we want to distinguish between graphs where every oracle solution is smaller by one compared to the original instance. For this, we make use of a single step of an iterative compression algorithm, by Guo et al. [12]. For the following results, we consulted the book [7].

Theorem 4.1. *(Algorithm Design for Hard Problems. [7]) There exists an algorithm called iterative compression algorithm for Min FVS with a running time in $\mathcal{O}(4^k \cdot n^4)$, where k denotes the size of a minimum FVS. This algorithm is based on the following compression result.*

Lemma 4.8. *(FVS compression) For a given graph G, a natural number k, and a feedback vertex set F of G of size at most $k+1$, the compression algorithm computes a feedback vertex set for G of size at most k, if it exists, and answers "NO", otherwise. The time complexity of the algorithm is in $\mathcal{O}(4^k \cdot n^3)$. An overview can be seen in Algorithm 5.*

Proof. The algorithm iterates over all possible intersections of a k-vertex feedback vertex set in G with the given feedback vertex set of size $k+1$. For each such intersection X, the algorithm calls the branching algorithm on the instance $(G - X, S - X, k - |X|)$. If this is a "yes"-instance for the DFVS, then the union of the respective feedback vertex set of size at most $k - |X|$ with X is a feedback vertex set for G of size at most k. Thus, the algorithm computes the right answer. There are at most $2k$ intersections that have to be tested, the running time is dominated by the calls to the branching algorithm, so the total running time can be bounded by $\mathcal{O}(4^k \cdot n^3)$.

The DFVS problem is the disjoint feedback vertex set problem. It takes a graph, a feedback vertex set F, and an integer k as input. It returns yes if there exists a feedback vertex set F' of size at most k with every vertex in F being different from F' and no otherwise. The iterative compression algorithm uses this step to decrease the size of a minimum FVS step by step. Since we can guarantee that the solution of Algorithm 4 is at most one bigger than the size of a minimum FVS we only need a single iteration of the compression algorithm to solve the problem. Thus, we save a factor of n in the running time and have a minimum FVS algorithm with a total running time of $\mathcal{O}(4^k \cdot n^3)$ instead of $\mathcal{O}(4^k \cdot n^4)$.

Algorithm 5 Compress-FVS

Input: A graph G and a feedback vertex set F of G of size at most $k+1$
Output: FVS of G of size at most k or "NO" if this is not possible

```
if |F| ≤ k then
  return S
for each X ⊆ F with |X| ≤ k do
  solve DVFS on (G − X, F − X, k − |X|) using a branching algorithm
  if (G − X, F − X, k − |X|) is a yes-instance then
    return FVS of size at most k
return "NO"
```

From Lemma 4.1, we know that we either construct a FVS which is one larger than a minimum FVS for the original instance or we construct a solution that has the same size as a minimum FVS for the original instance. With Lemma 4.8, we either get a FVS that is smaller by one compared to the FVS F that we gave the algorithm as an input or the algorithm returns "NO" if there does not exist a set of this size. Since F is at most bigger by one compared to a minimum FVS for the original instance we receive a minimum FVS as output from the compression algorithm. In the case that every oracle solution is smaller by one compared to a minimum FVS of the original instance we will give the compression algorithm a minimum FVS for G, due to Lemma 4.1. Hence, the compression algorithm outputs

"NO" and we know that the solution we created is already minimal and by Lemma 4.1 also correct. With this, we conclude with the analysis of the algorithm.

Theorem 4.2. *Algorithm 4 returns a minimum FVS in $|E|$ edge deletion queries for graphs where we have at least a critical cycle and not every edge belonging to a critical cycle in polynomial running time. For graphs without a critical cycle or every edge belonging to a critical cycle, it returns a minimum FVS in $\mathcal{O}(4^k \cdot n^3)$ running time.*

Proof. We start by showing the correctness of the algorithm. We know from Lemma 4.1 that, if we have two solutions with different sizes, the smaller of the two solutions combined with an endpoint of the corresponding deleted edge is a minimum FVS for G. The remaining lemmas shown previously cover the different corner cases, and thus we conclude that the algorithm is correct.
For the running time, we have that the algorithm deletes at most every edge in G once. Since we can make calls to the oracle in constant time, we have a running time of $\mathcal{O}(|E|)$ for graphs where some, but not all edges belong to a critical cycle. In the case that every edge belongs to a critical cycle or no edge belongs to a critical cycle, we have a running time of $\mathcal{O}(4^k \cdot n^3)$. □

This algorithm performs significantly better in the case where we have some edges in the graph belonging to a critical cycle. In the case that every edge or no edge belongs to a critical cycle we improve the running time of the iterative compression algorithm by a factor of n. This is achieved by having an approximation for the size of the minimum FVS for the original instance that can vary by one. Hence, we only need a single iteration of the iterative compression algorithm to shrink the FVS we create by one.

Chapter 5

Edge Contraction

In this section, we look at edge-contraction queries. As already defined in Chapter 1, we allow the contraction of a single existing edge e for a given graph G. In this section, we present an algorithm that computes the minimum FVS for a given graph G where no edge is a chord in at most $|E|$ queries if there exists an edge where both of its endpoints belong to the same minimum FVS for G or the oracle solution does not contain the newly introduced vertex w by contracting an edge in G. If neither of those conditions is met, the algorithm uses a single step of an improved branching algorithm as a subroutine to find a minimum FVS for G in time $\mathcal{O}(4^k \cdot n^3)$, where k denotes the size of a minimum FVS for G. The algorithm presented in Algorithm 6 iteratively deletes every edge once and checks if it can find two solutions of different sizes. If it finds two such solutions, it returns the smaller of the two and adds back both of the endpoints of the contracted edge while removing the newly introduced vertex w. Further, if it gets an oracle solution that does not contain w, it returns the oracle solution. If it does not find such two oracle solutions, it checks if the size of the oracle solution is always one smaller than the size of a minimum FVS for the original instance or if all the oracle solutions have the same size as a minimum FVS for the original instance by using a single iteration of the compression algorithm. In the first case, we can return an arbitrary oracle solution where the algorithm removes w and adds back both endpoints of the contracted edge. In the second case, it constructs a FVS for the original instance and decreases the size of it by one with the improved branching algorithm. A high-level overview of the algorithm can be found in Algorithm 6.

Since we do not want to introduce a multigraph by contracting an edge we from here on only consider cycles of length $l \geq 4$.

Lemma 5.1. *By a single edge contraction in a triangle-free graph G, no cycle of length at least 4 can be destroyed unless by contracting a chord.*

Proof. We prove the lemma by a case distinction over the different relations an edge e can have to a cycle C in G.

Case 1. The edge e is in C: Since we consider triangle-free graphs, there do not exist cycles of length 3. Further, contracting such an edge e decreases the length of C by exactly one and C is in G'.

Case 2. Only one of the vertices is in C: Then C has still the same size and C in G'

Algorithm 6 Minimum FVS with edge-contraction queries

```
Input: Arbitrary triangle-free graph G
Output: Minimum FVS of G
for each edge e = {u, v} ∈ C do
    w ← contract(u, v)
    G' ← G/e.
    S' ← oracle solution for G'
    if S' is smaller than any previously seen oracle solution then
        return S' − {w} + {u, v}
    if w is not in S' then
        return S'
FVS ← S' − {w} + {u, v}
output ← ImprovedBranchingAlgo(G, size of oracle solution, FVS)
if ImprovedBranchingAlgo returns no then
    return FVS
else
    return output of ImprovedBranchingAlgo
```

Case 3. Neither vertex is in C: Then obviously C in G'. □

From Lemma 5.1, we know that, if we choose the edge we contract carefully, the number of cycles in the graph will not decrease. Next, we want to formalize the observation that, when contracting an edge e that is not a chord, then the vertex $w = \mathit{contract}(u, v)$ destroys the same cycles as u and v do.

Lemma 5.2. *For a given graph G and an edge $e = \{u, v\}$ that is not a chord, we have that, in the graph G' where we contracted e and introduced $w = \mathit{contract}(u, v)$, the vertex w destroys every cycle in G' that u and v destroy in G.*

Proof. We want to show that w lies on every cycle that u and v lie on. We make a case distinction over the position of the endpoints of the contracted edge e.

Case 1. Both of the endpoints of e are on C: Since every cycle has at least length $l \geq 4$, we know by Lemma 5.1 that $C - e$ is still present in G'. Further, from the definition of edge contraction, we know that w is in C.

Case 2. One of the endpoints of e is on C: Then we know that, by contracting e, C is still present in G. Further, both of the edges that are adjacent to u or v on C are now adjacent to w and therefore w is on C.

Case 3. None of the endpoints of e is on C: Then neither u nor w lies on C and the claim is trivial. □

Now, by Lemma 5.1, we know that the amount of cycles in a given graph G does not change, unless we contract a chord in a cycle C. Next, we want to investigate what happens when we contract an edge where both of its endpoints belong to the same minimum FVS.

Lemma 5.3. *Contracting an edge $e = \{u, v\}$ in a graph G where there exists a minimum FVS F with u and v in F, we have that $\alpha(G') = \alpha(G) - 1$.*

Proof. From Lemma 5.2 we know that the newly introduced node $w = \mathit{contract}(u, v)$ destroys every cycle that previously was destroyed by u and v. Now, since u and v are in F, and we do not create any new cycles by contracting e, we define the minimum FVS $F' := F - \{u, v\} + \{w\}$. F' is smaller by one compared to F and a minimum FVS for $G' = G/e$. □

We make another observation, namely, if we contract an edge $e = \{u, v\}$ where both u and v are in the same minimum FVS for G, the resulting vertex $w = \mathit{contract}(u, v)$ from contracting e needs to be in the minimum FVS for G'. Intuitively, since w destroys the same cycles as u and v together, including w allows the minimum FVS for G' to be smaller than a minimum FVS for G.

Lemma 5.4. *For a graph G' where we contracted an edge $e = \{u, v\}$ which is not a chord in a graph G with a minimum FVS F and u and v in F, the newly introduced vertex $w = \mathit{contract}(u, v)$ is part of every minimum FVS F' for G'.*

Proof. Assume towards a contradiction that w is not part of a minimum FVS for G'. We know by Lemma 5.1 that G' does still contain the same amount of cycles as G. Further, from Lemma 5.3, we know, that including w allows a minimum FVS for G' with a size one smaller than a minimum FVS for G. Thus, we would need to find a minimum FVS which is also one smaller compared to a minimum FVS for G without w. Since G' does still have the same amount of cycles as G this is a contradiction with our assumption that w is not in the minimum FVS. □

Thus, we can identify such solutions by looking if the newly introduced vertex w is in the oracle solution of a graph $G' = G/e$. Now, we show how to construct an FVS from an oracle solution for $G' = G/e$.

Lemma 5.5. *The set $F' - \{w\} + \{u, v\}$, where F' is an oracle solution for a graph $G' = G/e$, with $e = \{u, v\}$, e not being a chord and $w = \mathit{contract}(u, v)$, is an FVS for G.*

Proof. We know from Lemma 5.1 that G' has the same amount of cycles as G. Thus, we want to show that every cycle C in G is destroyed by F'. We make a case distinction over whether the vertex w lies on C or not.

Case 1. The vertex w is not part of C: Since F' is a minimum FVS for G', there exists a vertex in F' that destroys C.

Case 2. The vertex w is part of C: Then we know from Lemma 5.2 that the vertices u and v destroy every cycle in G that w destroys in G'. □

With this fact, we can construct a minimum FVS for an oracle solution $G' = G/e$, if both endpoints of e belong to the same minimum FVS for G.

Lemma 5.6. *The set $F' - \{w\} + \{u, v\}$, where F' is an oracle solution for a graph $G' = G/e$ and u and v are in the same minimum FVS for G, is a minimum FVS for G.*

Proof. From Lemma 5.5, we know that the set $F' - \{w\} + \{u, v\}$ is a FVS for G. Further, we know from Lemma 4.8 that $|F'| = |F| - 1$, where F is a minimum FVS for G. Thus the set is a FVS for G and has the same size as a minimum FVS for G.□

We now consider the case where at most one of the endpoints belongs to a minimum FVS for G.

Lemma 5.7. *Contracting an edge $e = \{u, v\}$ that is not a chord in a given graph G, where at most u or v belong to a minimal FVS F for G implies that $|F| = |F'|$ where F' is a minimum FVS for G'.*

Proof. Since e is not a chord, we know by Lemma 5.1 that the same amount of cycles is still present in G'. Since at most u or v belongs to F, we know by the minimality principle that the remaining vertices in F are still needed to destroy every cycle in G'. □

Before concluding this chapter, we also want to show a special case, where we can also find a minimum FVS for G without both endpoints of a contracted edge e being in the same minimum FVS for G.

Lemma 5.8. *If $w = \ contract(u, v)$ is not in an oracle solution F' of a graph $G' = G/e$, with $e = \{u, v\}$, where at most either u or v belongs to a minimum FVS for G, then F' is a minimum FVS for G.*

Proof. Since we contracted an edge that is not a chord, we know from Lemma 5.1 that G' has the same amount of cycles as G. Since w is not in F', every vertex in F' destroys the same amount of cycles in G and G'. Further, from Lemma 5.7, it follows that F' has the same size as a minimum FVS of G. Thus, F' is a minimum FVS for G. □

Thus, we either have an oracle solution that is smaller by one compared to a minimum FVS for the original instance and we know how to construct a minimum FVS for G or we have an FVS that has the same size as a minimum FVS for the original instance and, if the introduced vertex w is not in the oracle solution, we also have a minimum FVS for G. It remains to show what we do if we have oracle solutions that all have the same size and if w is in every oracle solution.
In this case, we construct a FVS F for G via an oracle solution F' by removing w and adding u and v. Then we can make use of the improved branching algorithm to check if the size of F is the same as the size of a minimum FVS for G. If not we use the improved branching algorithm to return a FVS that is one smaller compared to F and this set will be a minimum FVS for G.

We conclude this chapter with the analysis of the Algorithm 6.

Theorem 5.1. *The algorithm Algorithm 6 finds a minimum FVS for a chord-free and triangle-free graph G in $|E|$ edge-contraction queries for graphs where either there exists a minimum FVS F for G with two adjacent vertices being both in F or if there exists a minimum FVS F' for the graph $G' = V/e$ where the newly introduced vertex $w = \ contract(u, v)$ is not part of F'. For graphs that do not meet either of these requirements, it returns a minimum FVS in $\mathcal{O}(4^k \cdot n^3)$ running time.*

Proof. We start with the correctness of the algorithm. If G does not contain any cycles, Algorithm 1 will return "NO" and we simply return the empty set as a minimum FVS for G. In the case that we have two oracle solutions with different

sizes or the newly introduced vertex $w = \mathit{contract}(u, v)$ is not in an oracle solution, the algorithm constructs a minimum FVS for G without using the improved branching algorithm. The remaining lemmas shown previously cover the different corner cases, and thus we conclude that the algorithm is correct.
For the running time, we have that the algorithm deletes at most every edge in G once. Since we can make calls to the oracle in constant time, we have a running time of $\mathcal{O}(|E|)$ for graphs where w is not in an oracle solution or where we have two oracle solutions with different sizes. In the case that every oracle solution contains w and has the same size, we have a running time of $\mathcal{O}(4^k \cdot n^3)$. □

Chapter 6

Conclusion and Further Research

We have seen that we can bound the number of queries for the local modification of edge insertion and vertex deletion. For edge deletion, we have seen that we can find a solution in polynomial time if a critical cycle is present in the input graph. An interesting continuation would be if we can find an algorithm that solves minFVS in polynomial time for graphs that do not have a critical cycle. Another promising approach could be a further improvement in the iterative compression algorithm by making better use of the oracle model. Lastly, one could investigate if the edge-contraction query algorithm works on a wider class of graphs.
We can conclude that the neighborly-help model allows us to find an FVS with a size at most one bigger than a minimum FVS with one oracle query for all local modifications we looked at. A different aspect that would be interesting is, to show general lower and upper bounds for certain local modifications within the neighborly-help model.

Bibliography

[1] C. Archetti, L. Bertazzi, and M. G. Speranza. Reoptimizing the Traveling Salesman Problem. *Networks, 42(3):*154-159, 2003.

[2] A. Arora and B. Barak. Computational Complexity: A Modern Approach. *Cambridge University Press,* 2009.

[3] G. Ausiello, B. Escoffier, J. Monnot, and V. Paschos. Reoptimization of minimum and maximum traveling salesman's tours. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006).* Vol. 4059. Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, pp. 196-207, 2006.

[4] Hans-Joachim Böckenhauer, Elisabet Burjons, Martin Raszyk, Peter Rossmanith. Reoptimization of parameterized problems. *Acta Informatica 59(4):*427-450 (2022).

[5] Hans-Joachim Böckenhauer, L. Forlizzi, J. Hromkovič, J. Kneis, J. Kupke, G. Proietti, and P. Widmayer. Reusing optimal TSP solutions for locally modified input instances. In *Proceedings of the 4th IFIP International Conference on Theoretical Computer Science (IFIP TCS 2006).* IFIP, New York, 2006, pp. 251-270.

[6] Hans-Joachim Böckenhauer, Juraj Hromkovič, Dennis Komm. Reoptimization of Hard Optimization Problems. In *Handbook of Approximation Algorithms and Metaheuristics (1):*427-454, 2018.

[7] Hans-Joachim Böckenhauer, Juraj Hromkovič, Dennis Komm Algorithm Design for Hard Problems. *Unpublished manuscript* 2023.

[8] E. Burjons, F.Frei, E. Hemaspaandra, D. Komm, and D. Wehner. Finding Optimal Solutions With Neighborly Help. *CoRR, abs/1906.10078,* 2019

[9] Gabriel A. Dirac. Some theorems on abstract graphs In *Proceedings of the London Mathematical Society,* s3-2(1):69-81, 1952.

[10] Piotr Faliszewski and Mitsunori Ogihara. On the autoreducibility of functions. *Theory of Computing Systems, 46(2):*222-245, 2010.

[11] Fabian Frei, Edith Hemaspaandra, Jörg Rothe. Complexity of stability *J. Comput. Syst. Sci. 123:*103-121, 2022

[12] Jiong Guo, Hannes Moser, Rolf Niedermeier. Iterative compression for exactly solving NP-hard minimization problems In *Algorithmics of Large and Complex Networks,* Lecture Notes in Computer Science, vol. 5515, Springer, pp. 65-80, 2009

[13] Timon Reichert *Neighborly Help for the Steiner Tree Problem* Bachelor thesis, ETH Zürich, Department of Mathematics, 2021

[14] Richard M. Karp. Reducibility Among Combinatorial Problems In *Proc. Symposium on Complexity of Computer Computations, IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., New York: Plenum,* pp. 85-103, 1972.

[15] Singhal, Manish. Deadlock detection in distributed systems. In *Computer, 10.1109/2.43525,* 1989.

[16] Markus W. Schäffter. Scheduling with forbidden sets. *Discrete Applied Mathematics, 72(1-2):*155-166, 1997.

[17] Walter Wessel. Criticity with respect to properties and operations in graph theory In Lás- zló Lovász András Hajnal and Vera T. Sós, editors, *Finite and Infinite Sets. (6th Hungarian Combinatorial Colloquium, Eger, 1981), volume 2 of Colloquia Mathematica Societatis Janos Bolyai,* pages 829-837. North-Holland, 1984.

Eigenständigkeitserklärung

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgeschlossen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und die Betreuerinnen der Arbeit.

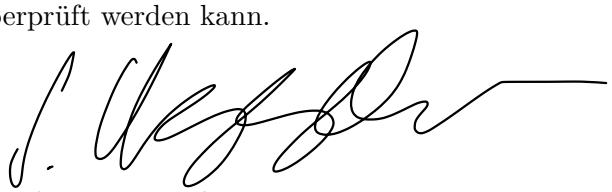
Titel der Arbeit: Neighborly Help for the Feedback Vertex Set Problem

Verfasst von: Samuel Aggeler

Ich bestätige mit meiner Unterschrift

- Ich habe keine im Merkblatt "Zitier-Knigge" beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.



Zürich, 11. Dezember 2023