Minimum Feedback Vertex Set for graphs of restricted degree

I.G. de Wolff

Bachelor thesis

Supervisor: dr. J.A. Hoogeveen

Daily supervisor: T.C. van der Zanden

June 15, 2017

Abstract

Finding a minimum feedback vertex set, a set whose removal makes a graph acyclic, is an NP-complete problem. We study the behaviour of existing algorithms for this problem on bounded degree graphs. Using the notion of generalized degree, we show that the algorithm of Fomin et al. [2] runs in time $\mathcal{O}^*(1.6181^n)$ on graphs of maximum degree three (although this case is polynomial-time solvable using a different algorithm) and in contrast, show that for graphs of degree four the generalized degree can become arbitrarily large. We show that the algorithm of Xiao [7] runs in time $\mathcal{O}^*(1.7180^n)$ on degree four graphs, which improves the best known bound for this case. Finally, we show that we cannot extend the measure of Fomin et al. [2] by using the generalized degree.

Utrecht University

1 Introduction

Algorithms can be categorized by whether they run in polynomial time or not. If the running time of an algorithm is in $O(n^k)$ for some k, it runs in polynomial time and it is considered a *fast* algorithm. For a wide range of problems it has been proven that they cannot be solved in polynomial time, except when $P = NP$, one of the major open questions in computing science [3]. The problem of finding a minimum feedback vertex set is one of those problems. A minimum feedback vertex set is a set of vertices, such that the graph without these vertices is acyclic [5, 2]. In this thesis, we will look at the role of the degree in this problem. We will make better analyses and prove that certain analyses are not possible for instances of a restricted degree.

The trivial algorithm, which tries all subsets of vertices of the graph, takes $\mathcal{O}^*(2^n)$ time[1]. The first algorithm that broke this barrier was only targeted at graphs of maximum degree four. This algorithm had a time complexity of $\mathcal{O}^*(1.945^n)$ [4]. Later on, algorithms have been designed which can solve the general case, without a restriction on the degree of graphs. The first algorithm that outperformed the trivial algorithm, without a restriction on the graph ran in $\mathcal{O}^*(1.8899^n)$ [5]. Later, this has been improved to $\mathcal{O}^*(1.7548^n)$ [2] and $\mathcal{O}^*(1.7266^n)$ [7]. These algorithms are faster than the algorithm designed for a maximum degree of four [4].

This thesis focuses on the role of the degree in this problem. We will first define the feedback vertex set problem and provide some background information on exponential algorithms in section 2. We describe some existing algorithms in section 3.

We will look at the relation between the maximum degree and the generalized degree, a property of vertices which is used in the studied algorithms. We will start by looking at graphs of maximum degree three. While this problem is solvable in polynomial time, we will look at the behavior of an existing exponential algorithm [2] in section 4. We will show that the generalized degree is bounded when the maximum degree of a graph is three. Using this result, we find an upper bound of $\mathcal{O}^*(1.6181^n)$ for graphs of maximum degree three for an existing algorithm. An upper bound on generalized degree does not exist for graphs of maximum degree four, as the generalized degree can become arbitrarily large as shown in section 5. However, we do find a better upper bound, $\mathcal{O}^*(1.7180^n)$, based on the measure used in the analysis of Xiao et al. [7].

The studied algorithms use measures which do not use the generalized degree. However, since the generalized degree is heavily used in the algorithms and since it gives information on the reduced instances generated by the branching rules, it sounds like a good candidate to use in the analysis. Furthermore, given that the generalized degree is initially equal to the degree, it can be used to give tighter upper bounds for classes of graphs of restricted degree. We will show in section 6 that such analysis cannot be made.

[1] Similar to Big-O notation, the $\mathcal{O}^*$ ignores all constant *and polynomial* factors.

2 Preliminaries

Let $G = (V, E)$ be an undirected simple graph consisting of a set of vertices V and a set of undirected edges E between those vertices. We denote by $deg(v)$ the degree of a vertex $v \in V$ and by $\Delta(G)$ the maximum vertex degree of G. We write $N(v)$ for the set of neighbours of v [1]. Given $V' \subset V$, we define the induced subgraph, the graph with only the vertices in V' and the edges from E between the vertices in V', as $G[V']$. We write $G \setminus V'$ for $G[V \setminus V']$ [2]. Given a non-empty set $X \subset V$, we write $\Delta^*(X)$ for the maximum degree of vertices in X in graph G. We define $\Delta^*(\emptyset) = 0$.

The connected components of a graph are the equivalence classes under the is-reachable-from relation [1]. Thus, a connected component is a subset of vertices such that all vertices in that set are reachable, and all other vertices are not reachable from that set. Let $F \subset V$ be an acyclic subset of vertices. A connected component of $G[F]$ is non-trivial if it has at least two vertices.

In this thesis we will take a look at the problem of finding a minimum feedback vertex set (FVS).

Definition 1 (Feedback Vertex Set, FVS)**.** *A subset $X \subset V$ is called a feedback vertex set (FVS) if $G \setminus X$ has no cycles. If X has the lowest cardinality among all feedback vertex sets, it is called a minimum feedback vertex set.*

Most algorithms that construct a minimum feedback vertex set actually search for a (forced) maximum induced forest.

Definition 2 (Maximum Induced Forest, MIF)**.** *A subset $Y \subset V$ is called an induced forest if $G[Y]$ is a forest. If Y has maximum cardinality among all induced forests, it is called a maximum induced forest (MIF).*

Definition 3 (Forced Maximum Induced Forest, F-MIF)**.** *Given a set $F \subset V$, a subset $Y \subset V$ is called a forced induced forest if $Y \supset F$ and $G[Y]$ is a forest. If Y has maximum cardinality among all forced induced forests, it is called a forced maximum induced forest (F-MIF).*

Note that finding a MIF is equivalent to finding a $\emptyset$-MIF, and that the complement of a MIF is a minimum FVS. We write $mif(G, F)$ for the size of the forced maximum induced forests of a graph G. The MIF problem is equivalent to $\emptyset$-MIF.

The algorithms we look at [5, 2, 7], require that F is an independent set.

Definition 4 (Independent set)**.** *A subset $Y \subset V$ is called an independent set if $G[Y]$ has no edges.*

When F is not an independent set, the algorithm compresses a component of F into a single vertex. This happens in the Id^* operation. The $Id^*(T, v_T)$ procedure contracts all vertices in T to a single new vertex v_T. Self loops on v_T (edges between v_T and v_T) are removed. If v_T has multiple edges with a vertex w, then vertex w is removed from the graph, since including that vertex would give a cycle.

Definition 5 (Generalized neighbours and generalized degree)**.** *Let (G, F) be an instance of F-MIF and let $v, w \in V \setminus F$. Then v and w are generalized neighbours if they are neighbours or if they share a common neighbour in $F \setminus \{t\}$. Vertex t is the active vertex; we will define this in more detail later on. We write $GD(v)$ for the set of all generalized neighbours of v. We define $gd(v)$ as the generalized degree, the amount of generalized neighbours of v [7].*

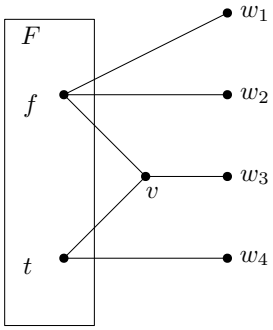

Figure 1: Example to illustrate the generalized degree

Consider the example in Figure 1. The generalized neighbours of v are w_1, w_2 and w_3. Vertex w_3 is a normal neighbour and w_1, w_2 and v share f as a common neighbour. Vertices f and t are not generalized neighbours since they are in F, and w_4 is not a generalized neighour since t does not count as a common neighbour.

We use Big-O notation to write the asymptotic upper bound on time complexity of algorithms [3].

Definition 6 (Big-O notation)**.** *We write $\mathcal{O}(g(n))$ for the following set of functions [1]:*

$$\mathcal{O}(g(n)) = \{\ f(n) \mid \exists\ c, n_0 \geq 0 : \forall n \geq n_0 : 0 \leq f(n) \leq cg(n)\ \}$$

We use $\mathcal{O}^*(g(n))$ to suppress any polynomial factors [3].

$$\mathcal{O}^*(g(n)) = \{\ f(n) \mid f(n) \in \mathcal{O}(g(n)\ poly(n)),\ poly(n)\ \textit{is a polynomial}\ \}$$

It is common to write $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$.

2.1 Branching algorithms

We will give a small introduction to branching algorithms. These algorithms can also be called splitting, backtracking or search tree algorithms. Such algorithm perform the following rules [3]:

- *Reduction rules*: A problem instance is simplified or the algorithm is halted

- *Branching rules*: A problem instance is split in smaller instances

The following rules are examples of reduction rules for the maximum induced forest problem, quoted from the algorithm in [2]:

1) If there is a $v \in N(t)$ with $gd(v) \leq 1$, then

$$mif(G, F) = mif(G, F \cup \{v\})$$

2) If $F = V$, then

$$mif(G, F) = |V|$$

In these algorithms, the original instance ($mif(G, F)$) is simplified (rule 1), or the algorithm is halted and the result is calculated directly (rule 2).

Branching rules may split the instance in multiple instances, as illustrated by the following example, cited from [2]. These rules usually result in an exponential time complexity, as they can recursively branch into an exponential number of instances.

> If there is $v \in N(t)$ with $gd(v) \geq 4$ then either add v to F or remove v from G:
>
> $$mif(G, F) = \max\{mif(G, F \cup v), mif(G \setminus \{v\}, F)\}$$

2.1.1 Analysis

Various methods exist to analyze an algorithm. The branching of algorithms can be modeled using recurrence relations. If a rule branches on r instances and the size of those instances is reduced with at least $r_1 \ldots t_r$ vertices, we can write this as the following linear recurrence [3]:

$$T(n) \leq T(n - t_1) + T(n - t_2) + ... + T(n - t_r) \tag{1}$$

Polynomial factors are ignored here. The equality variant of this recurrence has solutions which can be written as $T(n) = x^n$ [3]. This can be solved by taking the roots of

$$x^n - x^{n-t_1} - x^{n-t_2} - ... - x^{n-t_r} \tag{2}$$

Let α be the (unique) positive root of this polynomial. We call α the branching factor of this branching rule. Let β be the maximum of all branching numbers. The algorithm has a time complexity of $\mathcal{O}^*(\beta^n)$ [3].

More sophisticated analysis can be made using Measure & Conquer [3]. Instead of giving each vertex the same weight, a vertex can get a weight based on some metric related to the problem. One can for instance give different weights to vertices based on their degree or whether they are in some set. The rest of the analysis can be done in the same manner. Note that this analysis finds the running time expressed in the weight of an instance. If the weight of a vertex is at most one, this will directly give the time complexity expressed in the amount of vertices. The algorithms that we will discuss use Measure & Conquer in their analysis.

3 Algorithms

We will discuss various algorithms that find a minimum FVS. These algorithms search for a maximum induced forest (MIF). Clearly, F is a maximum induced forest iff $V \setminus F$ is a minimum feedback vertex set.

We will discuss the following algorithms, which all solve the F-MIF problem:

- Razgons algorithm, $\mathcal{O}^*(1.8899^n)$ [5].
- Fomins algorithm, $\mathcal{O}^*(1.7548^n)$ [2].
- Xiaos algorithm, $\mathcal{O}^*(1.7266^n)$ [7].

Razgons [5] algorithm was the first algorithm to beat the trivial $\mathcal{O}^*(2^n)$ algorithm. Later he found an algorithm which runs in $\mathcal{O}^*(1.7548^n)$ with Fomin et al. [2]. Xiao and Nagamochi [7] improved this algorithm with additional branching rules and a different measure in their analysis.

3.1 Reduction & branching rules

We will describe the algorithm of Fomin et al. [2] here. We will use this algorithm in most of our analyses.

Choose the first procedure that applies:

Preprocessing

1. If G consists of multiple connected components, invoke the algorithm on each connected component and take the sum of these results.
2. If G is not independent, then apply $Id^*(T, v_T)$ on a non-trivial component T of F, which compresses all vertices in T to a new vertex v_T. If $t \in T$, v_T will become the active vertex.

Main procedures

1. If $F = V$, no other vertices can be added to the induced forest, thus $mif(G, F) = |V|$.
2. If $F = \emptyset$ and $\Delta(G) \leq 1$, G has no cycles, thus $mif(G, F) = |V|$.
3. If $F = \emptyset$, branch on a vertex v with a degree of at least two:

$$mif(G, F) = \max\{mif(G, \{v\}), \\ mif(G \setminus \{v\}, \emptyset)\}$$

4. If F has no active vertex, then choose an arbitrary vertex as the active vertex and denote it by t.

5. If $V \setminus F = N(t)$, then we must choose vertices from $V \setminus F$, without choosing two vertices that are generalized neighbours. Thus, construct a graph with all vertices in $V \setminus F$ and add an edge between two vertices if they are generalized neighbours and construct a maximum independent set for this graph. Let m be the size of this set. Then $mif(G, F) = |F| + m$.
6. If the graph has a vertex $v \in N(t)$ with $gd(v) \leq 1$, then there is an F-MIF containing v. Thus, $mif(G, F) = mif(G, F \cup \{v\})$.
7. If the graph has a vertex $v \in N(t)$ with $gd(v) \geq 4$, branch on that vertex:

$$mif(G, F) = \max\{mif(G, F \cup \{v\}), \\ mif(G \setminus \{v\}, F)\}$$

8. If the graph has a vertex $v \in N(t)$ with $gd(v) = 2$, then we will branch on v. Let w_1 and w_2 be the generalized neighbours of v. Then

$$mif(G, F) = \max\{mif(G, F \cup \{v\}), \\ mif(G \setminus \{v\}, F \cup \{w_1, w_2\})\}$$

   If the last branch gives a cycle, we ignore that branch.
9. If all neighbours of t have generalized degree three, at least one vertex must have a generalized neighbour which is not a neighbour of t, since Main 5 did not apply. Let $v \in N(t)$ be such vertex. We will branch on v. Let w_1, w_2 and w_3 be the generalized neighbours of v, such that w_1 is not a neighbour of t. Then

$$mif(G, F) = \max\{mif(G, F \cup \{v\}), \\ mif(G \setminus \{v\}, F \cup \{w_1\}), \\ mif(G \setminus \{v, w_1\}, F \cup \{w_2, w_3\})\}$$

   If the last branch gives a cycle, we ignore that branch.

By removing Main 5 and 9 and by replacing $gd(v) \geq 4$ by $gd(v) \geq 3$ in Main 7, we get the algorithm of Razgon [2], as pointed out in [2]. The algorithm of Xiao et al. [7] has additional branching rules. Whereas the rules of the algorithm of Fomin et al. only determine the applied rule based on generalized degrees, Xiao et al. also use the degree of vertices. Furthermore, they also use the degree of vertices in the measure of their analysis.

4 Maximum degree 3

For graphs of maximum degree three, the problem can be solved in polynomial time [6]. However, the mentioned algorithms run in exponential time on these

graphs since they will branch on vertices of generalized degree two. We will make a more precise analysis of the algorithm of Fomin et al. for graphs of maximum degree three. The analysis is only targeted at the MIF problem, not its forced variant, since we assume that the algorithm starts with $F = \emptyset$.

We will prove the following statements in this order:

- The degree of vertices in $F \setminus \{t\}$ is at most two
- The generalized degree of vertices in $F \setminus V$ is at most three and the generalized degree of vertices in $N(t)$ is at most two.
- Procedures Main 7 and 9 will never apply.
- The algorithm of Fomin et al. solves these instances in $\mathcal{O}^*(1.6180^n)$ time.

4.1 Degree of vertices in $F \setminus \{t\}$

We start by showing that the degree of vertices in F, except t, is at most two.

Lemma 1. *When the algorithm of Fomin et al. starts with a graph of maximum degree three, $\Delta^*(F \setminus \{t\}) \leq 2$ holds at any point in execution after preprocessing.*

If there is no active vertex t, $F \setminus \{t\}$ should be read as F. To prove this lemma, we will first prove some auxiliary lemmas.

Lemma 2. *If $\Delta^*(F \setminus \{t\}) \leq 2$ holds before Preprocessing 1, this will also hold on the instances on which the program goes in recursion.*

Proof. If Preprocessing 1 is applied, the graph is split into its connected components. This does not alter the degree of the vertices.

Let F_i be a connected component of F. Then $F_i \setminus \{t\}$ is a subset of $F \setminus \{t\}$. Thus $\Delta^*(F_i \setminus \{t\}) \leq \Delta^*(F_i \setminus \{t\}) \leq 2$. □

Lemma 3. *If $\Delta^*(F \setminus \{t\}) \leq 2$ holds before Preprocessing 2, this will also hold on the reduced instance.*

Proof. If Preprocessing 2 is applied, a non-trivial component T of G is compressed to a single vertex. If $t \notin T$, T contains only vertices of degree 2. Since these vertices are connected, and F does not contain cycles, they must lay in a path. The first and last vertex may have an edge with a vertex outside of F. The constructed vertex v_T will be connected with only those neighbours and will thus have a degree of at most two.

If $t \in T$, the new vertex v_T will become the new active vertex. The reduced instance has $F' = (F \setminus T) \cup \{v_T\}$. Ignoring the new active vertex gives $F \setminus T$, which is a subset of $F \setminus \{t\}$. Thus $\Delta^*(F \setminus T) \leq \Delta^*(F_i \setminus \{t\}) \leq 2$. □

In the following proofs, we will often show that property $\Delta^*(F \setminus \{t\}) \leq 2$ holds after some operation and Preprocessing 2. However, the algorithm might first apply Preprocessing 1. To justify this, we will prove that the order on which Preprocessing 1 and Preprocessing 2 occur does not alter the generated instances.

Lemma 4. *When applying Preprocessing 2 before Preprocessing 1, the same instances are created as in the normal order.*

Proof. Let T be a subset of a connected component in F. The operation $Id^*(T, v_T)$ affects the vertices in T and neighbours of those. Given that T is connected, this affects only a single connected component of the graph and does not change other connected components. So, applying Preprocessing 2 does not alter the behavior of Preprocessing 1.

Since Preprocessing 1 splits the graph into its connected components, it will preserve any connected components. Thus, applying Preprocessing 1 does not affect Preprocessing 2. □

We will also use that two vertices will be compressed and show that $\Delta^*(F \setminus \{t\}) \leq 2$ holds after that compression. However, the algorithm might compress more vertices than those two. The following lemma justifies that.

Lemma 5. *If $\Delta^*(F \setminus \{t\}) \leq 2$ holds after applying Id^* on a set of vertices A, which is a subset of a component $T \subset F$, it will also hold after applying both Preprocessing 1 and Preprocessing 2.*

Proof. We will first show that applying $Id^*(T, v_T)$ gives the same result as applying $Id^*(A, v_A)$ and $Id^*(T, v_T)$. The first will compress all vertices in T to a single vertex, whereas the second compresses all vertices in $\{v_A\} \cup T \setminus A$. Thus, both operations compress the same set of vertices. Both operations remove vertices connected with a multi-edge. The second will first remove vertices connected with a multi-edge with vertices in A. Both operations will remove the same vertices, namely all vertices connected with a multi-edge with vertices in T.

This shows that $\Delta^*(F \setminus \{t\}) \leq 2$ holds after applying Preprocessing 2. We have seen before that Preprocessing 1 preserves this property and that the order of the preprocessing procedures does not alter the generated instances. □

We will now show that $\Delta^*(F \setminus \{t\}) \leq 2$ is preserved in the instances on which the algorithm goes in recursion.

Lemma 6. *If $\Delta^*(F \setminus \{t\}) \leq 2$ holds before the main procedures, this will also holds on the recursive instances generated by the main procedures, after these have been modified by the preprocessing steps.*

Proof. Main 1, 2 and 5 are the base cases of the algorithm and do not go in recursion. Main 4 marks a vertex in F as the active vertex. No degrees change and $F \setminus \{t\}$ contains one fewer vertex. Thus, $\Delta^*(F \setminus \{t\}) \leq \Delta^*(F) \leq 2$

In main 3, $F = \emptyset$. The first case adds a vertex to F, which will be marked as the active vertex t. Thus, $F \setminus \{t\} = \emptyset$ and $\Delta * (\emptyset) = 0$. In the second case, F is still empty and $\Delta^*(\emptyset) \leq 2$ will thus also hold.

We will now look at Main 6, 7, 8 and 9. In the first case of these procedures, a vertex v is added to F. In recursion, preprocessing 1 will not apply since the graph itself has not changed. In preprocessing 2, vertex v will be compressed

with t. Other vertices in $F \setminus \{t\}$ may also be compressed with t. Let F' and t' be the values of the generated instance. Then $F' \setminus \{t'\} \subset F \setminus \{t\}$, thus $\Delta^*(F' \setminus \{t'\}) \leq \Delta^*(F \setminus \{t\}) \leq 2$.

The second case of Main 7 removes a vertex from the graph. This does not increase degrees, nor does it change F, so the generalized degree of vertices in $F \setminus \{t\}$ is at most 2 before preprocessing. As seen in Lemma 2 and 3, preprocessing preserves this property.

The second recursion of Main 8 removes v from the graph and adds w_1 and w_2 to F. Vertices w_1 and w_2 are generalized neighbours of v and have a degree of at most three. Thus, w_1 is either a neighbour of v, or they share a common neighbour in $F \setminus \{t\}$. If w_1 is a neighbour of v, the degree of w_1 will be at most two when v is removed from the graph. If w_1 and v share a common neighbour in $F \setminus \{t\}$, say f, then w_1 will be compressed with f during Preprocessing 2. The degree of f is at most two, since $f \in F \setminus \{t\}$, thus the only neighbours of f are v and w_1. After the removal of v and the compression of w_1 and f, the only neighbours of the created vertex are the neighbours of w_1, except f. Thus, the degree of this vertex is at most two. Note that these vertices might be compressed with more vertices, in case w_1 has neighbours in $F \setminus \{t\}$ or if w_1 and w_2 are neighbours. Based on Lemma 5, we do not have to consider those cases. The same reasoning applies to w_2. Vertices w_1 and w_2 have a degree of at most two, so the property holds on the generated instances.

A similar reasoning as in Main 8 can be applied to the second and third recursion of Main 9. In these cases, vertices are added to F and a generalized neighbour (v) of these vertices is removed from the graph, which reduces their degree by one. □

Now we will prove Lemma 1.

Proof. The algorithm starts with $F = \emptyset$. During Preprocessing 1 and 2 no vertices are added to F, so the statements holds at the start of the algorithm. We have seen in Lemma 6 that this property is preserved during the execution of the algorithm. □

4.2 Bound on generalized degree

Based on the results of the previous section, we can now deduce a bound on the generalized degree:

Theorem 1 (Maximum generalized degree for graphs of degree three)**.** *When the algorithm of Fomin et al. starts with a graph of maximum degree three, the generalized degree of a vertex in $V \setminus F$ is at most three, at any point in the execution after preprocessing. Furthermore, the generalized degree of a vertex in $N(t)$ is at most two.*

Proof. Let v be an arbitrary vertex in $V \setminus F$. The degree of v is at most three. We will first show that the generalized degree of v is at most three.

On the contrary, assume that v has a generalized degree of at least four. We will show that we can find a neighbour for each generalized neighbour. Let w be a generalized neighbour of w. Then w is either a neighbour of v or they share a common neighbour in $F \setminus \{t\}$. In both cases we found a neighbour of v.

Since the degree of v is at most three, there must be at least two generalized neighbours w_1 and w_2, which share the same common neighbour with v. Let f be that neighbour. Then w_1, w_2 and v are neighbours of f, thus the degree of f is at least three. This contradicts with Lemma 1, thus the assumption that the generalized degree is at least four is false. So, the generalized degree of v is at most three.

Consider the case when $v \in N(t)$. Vertex v has at most two neighbours other than t. Using the same reasoning, we can show that v has at most two generalized neighbours. □

4.3 Time analysis

Since an instance cannot have a vertex of generalized degree 4 or higher, Main 7 will never be applied. Fomin et al. show that this is the only tight recurrence [2], so ignoring this case will yield a tighter time bound. Furthermore, the generalized degree of neighbours of t is at most two, thus Main 9 will also never be applied. In this section we will make a better analysis for graphs of maximum degree three.

Theorem 2. *The algorithm of Fomin et al. solves instances of maximum degree three in* $\mathcal{O}^*(1.6180^n)$ *time.*

Proof. We use Measure & Conquer [3] with the same measure as [2]:

$$\mu = |V \setminus F| + \alpha |V \setminus (F \cup N(t))| \tag{3}$$

All vertices in F have no weight, all vertices in $N(t)$ have weight 1 and all other vertices have weight $1 + \alpha$. Fomin et al. used $\alpha = 0.955$ and found an upper bound of $\mathcal{O}(1.33328^\mu)$. Given that we are looking at a different problem, we will use a different value to get a tighter time bound, satisfying $0 \leq \alpha \leq 1$. Note that the following analysis is largely adapted from [2].

Main 1 and 2 can be computed in polynomial time. Main 4 does not alter the weight, but cannot be executed in two consecutive runs: after applying Main 4, the instance has an active vertex, thus Main 4 will not be applied again in the next run. Thus, we may ignore these cases. Main 6 only recurses into a single instance. Preprocessing 1 splits the instance into smaller, distinct instances and Preprocessing 2 simplifies the problem without changing its weight. Thus, Preprocessing 1, 2 and Main 1, 2, 4 and 6 do not contribute to the exponential factor. Main 5 can be solved in $\mathcal{O}(1.2278^\mu)$, by theorem 4 in [2].

Let $f(\mu)$ be the largest number of recursive calls for an instance of size μ. We will show that $f(\mu) \leq x^\mu$, for some $x \in \mathbb{R}$ with induction. The induction base case clearly holds, as $f(0) = 1$.

We will now cite an equation for each main procedure from [2]. First we consider Main 3. Each vertex has weight $1 + \alpha$, so including or excluding t reduces the weight with at least $1 + \alpha$. By including it, t becomes the active vertex. This reduces the weight of at least two neighbours by α.

$$f(\mu) \leq f(\mu - 1 - \alpha) + f(\mu - 1 - 3\alpha) \leq x^{\mu-1-\alpha} + x^{\mu-1-3\alpha} \leq x^{\mu} \tag{4}$$

Based on our Theorem 1, we may ignore Main 7 and 9. In Main 8, we parameterize the relation based on the number of generalized neighbours with weight $1 + \alpha$. Let $i \in \{0, 1, 2\}$ be that number. Then we get the following recurrence relation:

$$\begin{aligned} f(\mu) &\leq f(\mu - (3 - i) - i\alpha) + f(\mu - 3 - i\alpha) \\ &\leq x^{\mu-(3-i)-i\alpha} + x^{\mu-3-i\alpha} \leq x^{\mu} \end{aligned} \tag{5}$$

These conditions hold when $\alpha = 1$ and $x = 1.2721$. These values have been calculated numerically using a Haskell program[2], which tried values of α between zero and one on a scale of 0.005. It searched for the corresponding best value for x using binary search. These values give an upper bound of $\mathcal{O}^*(1.27204^{\mu})$. When the algorithm starts, each vertex has weight $1 + \alpha$, thus $\mu = (1 + \alpha)n$. This yields an upper bound of $\mathcal{O}^*\left(\left(1.27204^{1+\alpha}\right)^n\right)$. By rounding this value we get $\mathcal{O}^*(1.6181^n)$.

□

5 Maximum degree 4

In the previous section, we have shown that a maximum degree of three implies that the generalized degree of vertices in $V \setminus F$ will be at most three. Sadly, a similar result cannot be found for graphs of maximum degree four. We will show that the generalized degree can become arbitrarily large when constraining the input to graphs of maximum degree four. We will show that in the following steps:

- We can construct graph $A(p, 0)$ $(p \geq 0)$ by starting with a graph of maximum degree four.
- We can transform graph $A(p, q)$ $(p \geq 1)$ into $A(p - 1, q + 1)$.
- The graph $A(0, q)$ has a vertex whose generalized degree is at least q.

We use these steps to show that we can make the generalized degree arbitrarily large. This demonstrates that the same analysis as the previous section cannot be done here. We can however use the measure in the algorithm of Xiao et al. [7] to find a better upper bound. Their measure uses the degree of vertices, which we can use to find an upper bound of $\mathcal{O}^*(1.7180^n)$.

[2]The source code can be found at
`https://gist.github.com/ivogabe/c4c192a492cf83d3bc868ba9c2fa721e`.

5.1 Graph construction

We use the following construction for our graph. Given $p, q \geq 0$, let $A(p, q)$ denote the instance in Figure 2, with $F = \{t, x\}$ and t as the active vertex. The center part is repeated p times and vertex x is connected with q other vertices, thus $deg(x) = q + 2$.

Lemma 7. *The graph $A(p, 0)$ can be constructed by starting with an instance of maximum degree four and $F = \emptyset$.*

Proof. We start with the graph in Figure 3. The center part in this graph is repeated p times. The graph has maximum degree four. We apply the first case of Main 3 on t and make it the active vertex. We apply the second case of Main 8 on vertex z and remove the isolated vertex with Preprocessing 1. The resulting graph is $A(p, 0)$. □

Lemma 8. *An instance $A(p, q)$ with active vertex t can yield $A(p-1, q+1)$ as a recursive instance.*

Proof. We start with the instance $A(p, q)$. All generalized neighbours of t have generalized degree two, so the algorithm will apply Main 8. First we apply the first case of Main 8 on v_p. Vertex v_p is then compressed with t. This is shown in Figure 4, with $F = \{t, x\}$. All neighbours of t have again generalized degree two, so Main 8 will be applied. We apply the second case of Main 8 on u_p. Vertex u_p is removed, the isolated vertex is removed during Preprocessing 1 and w_p is compressed with x. This results in $A(p-1, q+1)$. □

We can now prove that the generalized degree can become arbitrarily large for graphs of maximum degree four.

Theorem 3. *Given a positive integer k, an instance of the MIF problem exists such that during the execution of the algorithm of Fomin et al., the generalized degree of a vertex in $V(t)$ is at least k in some recursive instance.*

Proof. Let k be an arbitrary positive integer. We will construct a graph such that the generalized degree of a vertex will be at least k. We also show that this applies to vertices in $N(t)$ and for vertices in $V \setminus (F \cup N(t))$.

We have demonstrated that we can construct a graph which is reduced to $A(k, 0)$. This instance can be reduced to $A(k-1, 1)$ and further reduced to $A(0, k)$. Such instance is shown in Figure 5. The generalized degree of vertex y is now $k + 1$ and $y \in N(t)$, which proves this theorem. □

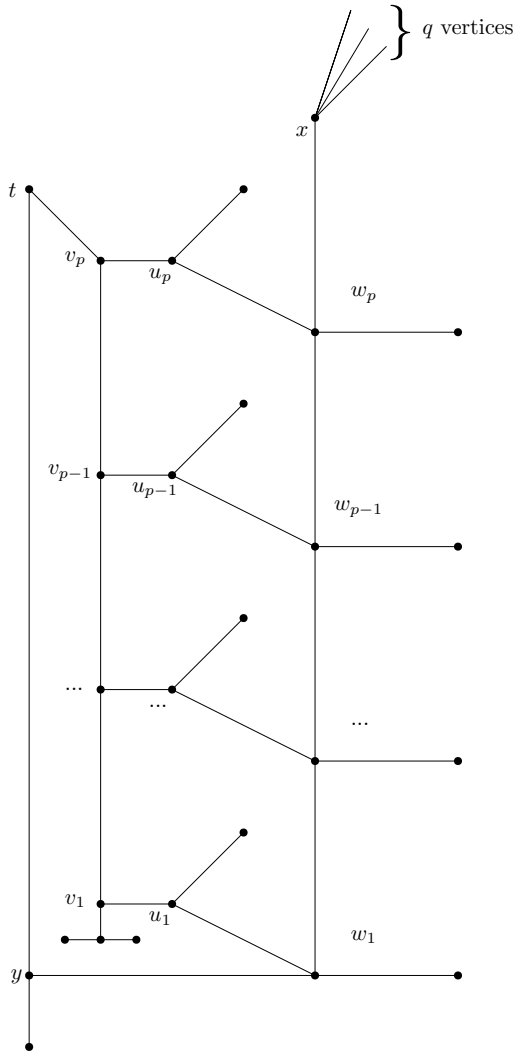

Figure 2: The instance $A(p, q)$, with $F = \{t, x\}$

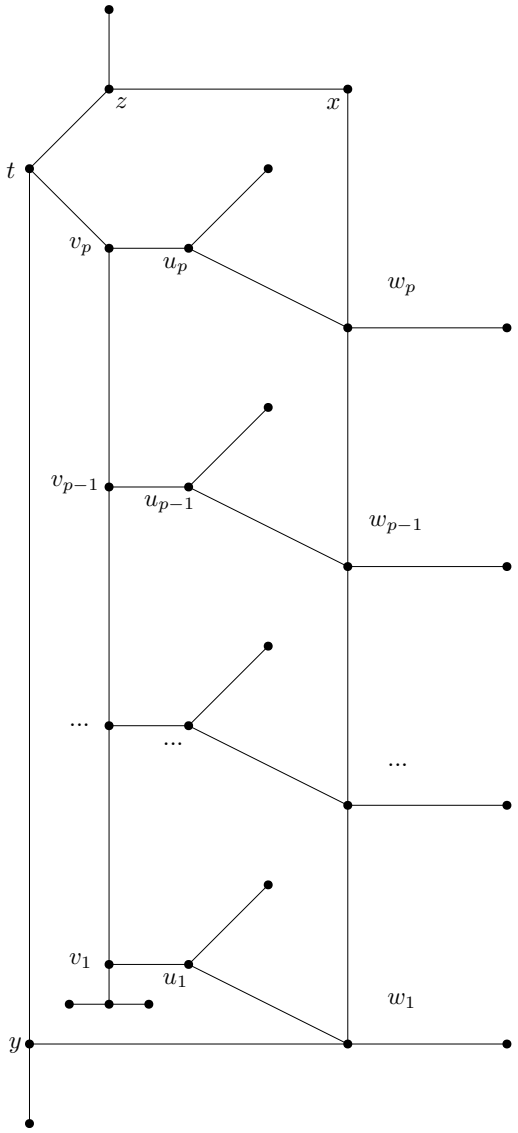

Figure 3: Instance of MIF which can be transformed to $A(p, 0)$

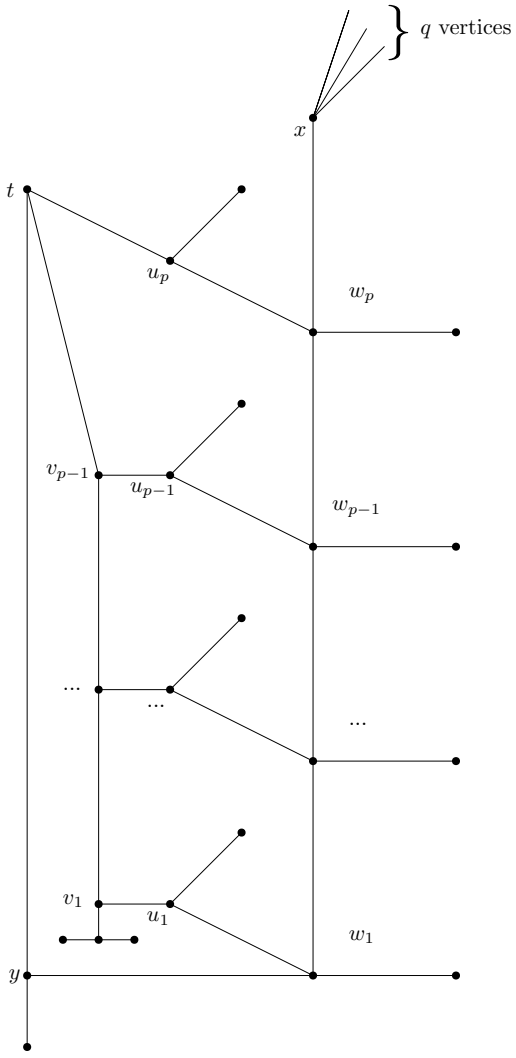

Figure 4: Instance during transition from $A(p, q)$ to $A(p-1, q+1)$, with $F = \{t, x\}$

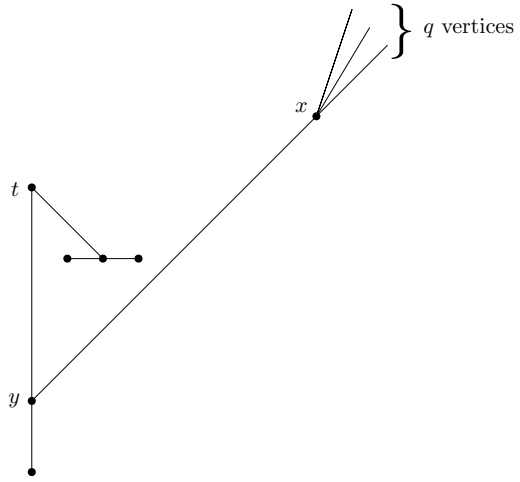

Figure 5: Instance $A(0, q)$, with $F = \{t, x\}$

5.2 Degree based measure

The algorithm of Xiao et al. used a measure based on the degree of vertices. The weight of a vertex $v \in V$ is defined as [7]:

$$w(v) = \begin{cases} \alpha + \beta & \text{if } v = t \\ 0 & \text{if } v \in V \setminus \{t\} \text{ and } d(v) \in \{0, 1\} \\ \beta & \text{if } v \in F \setminus \{t\} \text{ and } d(v) \geq 2 \\ \alpha & \text{if } v \in N(t) \cap (V \setminus F) \text{ and } d(v) \geq 2 \\ w_i & \text{if } v \in V \setminus (F \cup N(t)) \text{ and } d(v) = i \text{ for } i \in \{2, 3, 4\} \\ 1 & \text{if } v \in V \setminus (F \cup N(t)) \text{ and } d(v) \geq 5 \end{cases} \tag{6}$$

where $w_2 = 0.1695, w_3 = 0.9760, w_4 = 0.9898, \alpha = 0.5176, \beta = 0.1668$

We use this measure to find a tighter time bound for instances of MIF with a degree of at most four.

Theorem 4. *The algorithm of Xiao et al. can solve instances of MIF with a degree of at most four in* $\mathcal{O}^*(1.7180^n)$ *time.*

Proof. Assume that the degree of vertices in G is at most four. When the algorithm starts, $F = \emptyset$ and the instance has no active vertex. Thus, all vertices are in $V \setminus (F \cup N(t))$. Since all vertices have a degree of at most four, the weight of n vertices is at most nw_4.

Let μ be the weight of an instance. Xiao et al. prove in their analysis that they can solve such instance in $\mathcal{O}^*(1.7266^{\mu})$ time. For a graph of a degree of at most four, we have $\mu \leq nw_4$. Such instance can be solved in $\mathcal{O}^*\left(1.7266^{0.9898n}\right)$. Thus, their algorithm solves these instances in $\mathcal{O}^*(1.7180^n)$. □

6 Weight functions

The algorithms we looked at give different weight to vertices in F, $N(t)$ and $V \setminus (F \cup N(t))$. The weight functions, which give the weight of a single vertex [3], used in the analysis of the algorithm of Razgon [5] and the algorithm of Fomin et al. [2] can be written as:

$$w(v) = \begin{cases} 0 & \text{if } v \in F \\ 1 & \text{if } v \in N(t) \\ 1+\alpha & \text{otherwise} \end{cases} \tag{7}$$

Xiao et al. noticed that most branching algorithms on graphs use a measure based on the degree of vertices. They added additional branching rules, and have made a more precise analysis with a more complex weight function [7].

$$w(v) = \begin{cases} \alpha+\beta & \text{if } v = t \\ 0 & \text{if } v \in V \setminus \{t\} \text{ and } d(v) \in \{0,1\} \\ \beta & \text{if } v \in F \setminus \{t\} \text{ and } d(v) \geq 2 \\ \alpha & \text{if } v \in N(t) \cap (V \setminus F) \text{ and } d(v) \geq 2 \\ w_i & \text{if } v \in V \setminus (F \cup N(t)) \text{ and } d(v) = i \text{ for } i \in \{2,3,4\} \\ 1 & \text{if } v \in V \setminus (F \cup N(t)) \text{ and } d(v) \geq 5 \end{cases} \tag{8}$$

6.1 Generalized degree based measure

Given that the algorithm chooses a branching rule based on the generalized degree, one might consider a measure based on the generalized degree of a vertex. We will consider a measure of the following form:

$$w(v) = \begin{cases} 0 & \text{if } v \in F \\ x_{gd(v)} & \text{if } v \in N(t) \\ y_{gd(v)} & \text{otherwise} \end{cases} \tag{9}$$

Advantages of such a measure include that this measure is closer to the behavior of the algorithm than a degree based measure: the algorithm chooses a vertex to branch on based on the generalized degrees of vertices, and the generalized degree gives information on how many vertices will become neighbour of the active vertex after the execution of some procedure of the algorithm.

Furthermore, when the algorithm starts with an MIF instance, $F = \emptyset$, thus the generalized degree of each vertex is equal to its degree. One might want to use this to find a better time bound with a measure that depends on the generalized degree. Especially for this case, we will not add the restriction that y_i should have an upper bound. In general, an unbounded weight function would give trouble, as one cannot give an upper bound on the weight of an instance as a function of the input size. However, when only looking at graphs of a restricted degree, one can restrict the weight μ by $\mu \leq y_{\Delta(G)} n$.

We will make no assumptions on whether y_i is increasing, decreasing or monotone. We do assume that $x_i \leq y_{i+1}$ for all $i \geq 0$, meaning that when a vertex becomes a neighbour of t, its weight will decrease. In those cases, the generalized degree will decrease by one since t does not count as a generalized neighbour, hence the $i+1$ in the index.

In this section, we will prove that such a measure is only possible if $y_i = y_j$ for all $i, j \geq 1$. This means that one cannot use this measure to find a tighter upper bound for graphs of a restricted degree, as only vertices of generalized degree zero, meaning that they are not connected with other vertices and thus removed during Preprocessing 1, can have a different weight.

Theorem 5. *Let w be a weight function satisfying equation 9, with $x_i \leq y_{i+1}$ for all $i \geq 0$. If the weight of an instance will not increase by applying any main procedure followed by any applicable preprocessing steps, then $y_i = y_j$ for all $i, j \geq 1$.*

We define $\delta_i = y_{i+1} - y_i$, for $i \geq 0$. We will now prove this theorem in the following steps:

- The sequence δ_i is bounded from above.
- No values $c > 0, N \in \mathbb{N}$ exist such that $\delta_i > c$ for all $i \geq N$.
- The sequence δ_i is not negative.
- For all $i \geq 1$, $\delta_i = 0$.

Lemma 9. *The sequence δ_i is bounded from above.*

Proof. Assume to the contrary that δ_i is not bounded from above. For each $\epsilon > 0$, there exists an $i \geq 0$ such that $\delta_i > \epsilon$. This implies that such i also exists satisfying $i \geq 1$.

Let ϵ be arbitrary and let $i \geq 1$ be an integer such that $\delta_i > \epsilon$. Consider the F-MIF instance in Figure 6, where $F = \{t\}$ and $deg(u) = i$. By applying the second case of Main 8, w will be added to F and the generalized degree of u will increase from i to $i+1$.

We can make this increase arbitrarily large. Note that the weight of other vertices does not depend on i and that the weight of vertices v_k does not change. Thus, the change of weight of all vertices except v is constant, whereas the weight increase of u can become arbitrarily large. When choosing ϵ large enough, the weight will increase whereas we assumed that the weight will always decrease. Thus, the assumption that δ_i has no upper bound is false. □

Let d be an upper bound such that $\delta_i < d$ for all $i \geq 0$.

Lemma 10. *No values $c > 0, N \in \mathbb{N}$ exist such that $\delta_i > c$ for all $i \geq N$.*

Proof. Assume to the contrary that these values exist. Thus, $c > 0, N \in \mathbb{N}$ exist such that $\delta_i > c$ for all $i \geq N$. Let $n > N$ be arbitrary. We consider the instance

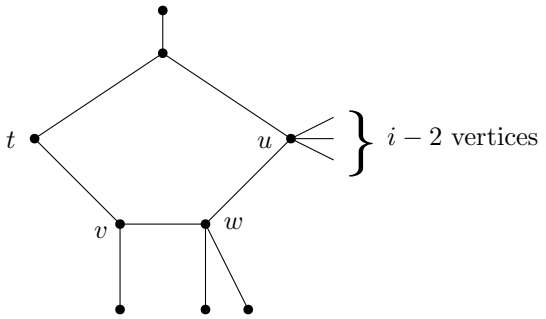

Figure 6: Instance of F-MIF where the generalized degree of u can increase

from Figure 7, where $F = \{t\}$. We apply the second case of Main 8 on vertex v. Vertex w will be added to F, which increases the generalized degree of vertices v_i from 1 to n. This increases the weight by at least nc for each vertex u_i, thus n^2c in total. Only the weight of w can depend on n, so this vertex should have a weight which is at least quadratic in n. This means that δ_i is at least linear, which means that δ_i is not bounded. This contradicts with the previous lemma, thus the assumption that c and N exist is not true. □

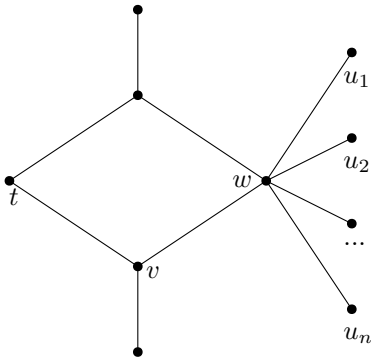

Figure 7: Instance of F-MIF where the generalized of n vertices can increase

Lemma 11. *The sequence δ_i is not negative.*

Proof. Assume to the contrary that there exists a $k \geq 0$ such that $\delta_k < 0$. Consider the instance in Figure 8 where $F = \{t\}$, t has two neighbours, v and w. Vertices v and w have the same neighbours, namely $u_1, u_2, ...u_n$, for an arbitrary $n \geq 4$. Vertices u_i have $k-1$ other neighbours. Thus, $deg(u_i) = gd(u_i) = k+1$.

We apply the second case of Main 7 to this instance. The generalized degrees of vertices u_i are reduced by one, increasing the weight by $n\delta_k$. The only vertex

which can compensate for this is v, since the weights of other vertices do not change. Thus, v should compensate for this increase. Thus, for large enough values of n, x_n should be at least linear: $x_n > nc$ if $n \geq N$, with $N \in \mathbb{N}$, $c > 0$. Given that $y_{n+1} \geq x_n$, sequence y_i is also at least linear for large enough values for i, thus $\delta_i \geq d$, for some $d > 0$. This contradicts the previous lemma. □

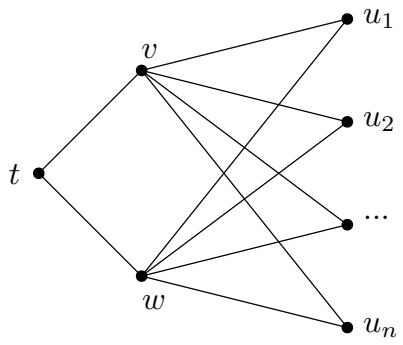

Figure 8: Instance of F-MIF where the generalized of n vertices can decrease

Lemma 12. *For all $i \geq 1$, $\delta_i = 0$.*

Proof. Assume to the contrary that there exists a k such that $\delta_k \neq 0$. We have shown that the sequence δ_i is not negative, thus $\delta_k > 0$. Let $n > k$ be arbitrary. We once more consider the instance from Figure 7, with $F = \{t\}$. We apply the second case of Main 8 on vertex v. Vertex w will be added to F. The generalized degree of vertices u_j will increase from 1 to n. Given that δ_i is not negative, this will increase the weight by at least δ_k per vertex u_j, thus $\delta_k n$ in total.

Only vertex w can have a weight that depends on n. Thus, this vertex should have a weight linear in n, to compensate for the decrease in weight of $\delta_k n$. This contradicts with Lemma 10, thus the assumption that k exists is false. Thus, $\delta_i = 0$ for all $i \geq 1$. □

We can now finish the proof for Theorem 5.

Proof. The previous lemma shows that $\delta_i = 0$ for all $i \geq 1$, thus $y_i = y_{i+1}$. It follows that $y_i = y_j$ for all $i, j \geq 1$. □

7 Conclusion

We have looked at various algorithms that solve the problem of finding a minimum feedback vertex set. The fastest known algorithm has a time upper bound of $\mathcal{O}^*(1.7266^n)$ [7]. The problem is solvable in polynomial time for graphs of maximum degree three [6]. We made a new analysis for the algorithm of Fomin

et al. [2] for graphs of maximum degree three and found an upper bound of $\mathcal{O}^*(1.6181^n)$. We have proven that a similar analysis cannot be made for graphs of maximum degree four. We did find a better upper bound, $\mathcal{O}^*(1.7180^n)$, for graphs of maximum degree four with the algorithm of Xiao et al., [7].

In 2005, an algorithm was shown which could solve the feedback vertex set faster than the trivial algorithm for graphs of maximum degree four [4]. This algorithm got beaten by several algorithms which do not have a restriction on the graph [5, 2, 7]. With the results in this thesis, the problem restricted to a maximum degree of four again has a better upper bound compared to the unrestricted problem.

Furthermore, we have looked at extending the analysis of Fomin et al. [2] with a measure that uses the generalized degree. We put strong constraints on such a measure.

7.1 Future work

In this thesis we looked at a maximum degree of three and four. The first gives an upper bound on the generalized degree, but makes the problem solvable in polynomial time. In the latter case, the generalized degree can become arbitrarily large. It can be interesting to search for a class of graphs, for which the problem is NP-complete, such that the generalized degree is bounded. Graph minors might be related, as the compression step can be compared to taking minors.

During the writing of this thesis, the following question came up multiple times: is the problem NP-complete for graphs where a few vertices have degree four, and all other vertices have degree three or less? This might sound as branching once on the vertex of degree four and then solving it in polynomial time, but the polynomial time algorithm [6] is only targeted at MIF, not F-MIF. This class of graphs might also be related to the previous suggestion for future work.

References

[1] Rivest R. L. Stein C. Cormen T. H., Leiserson C. E. *Introduction to Algorithms*. The MIT Press, 2009.

[2] Fedor V Fomin, Serge Gaspers, Artem V Pyatkin, and Igor Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. *Algorithmica*, 52(2):293–307, 2008.

[3] Fedor V Fomin and Dieter Kratsch. *Exact exponential algorithms*. Springer Science & Business Media, 2010.

[4] Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. Improved exact exponential algorithms for vertex bipartization and other problems. In *Italian Conference on Theoretical Computer Science*, pages 375–389. Springer, 2005.

[5] Igor Razgon. Exact computation of maximum induced forest. In *Scandinavian Workshop on Algorithm Theory*, pages 160–171. Springer, 2006.

[6] Shuichi Ueno, Yoji Kajitani, and Shin'ya Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1-3):355–360, 1988.

[7] Mingyu Xiao and Hiroshi Nagamochi. An improved exact algorithm for undirected feedback vertex set. *Journal of Combinatorial Optimization*, 30(2):214–241, 2015.