Parametrisierte Algorithmen für Feedback-Set-Probleme auf Turniergraphen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Informatiker

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA
Fakultät für Mathematik und Informatik

eingereicht von Anke Truß
geb. am 28.01.1980 in Dortmund

Betreuer: Michael Dom
Jiong Guo
Falk Hüffner
Prof. Dr. Rolf Niedermeier

Jena, 23. Dezember 2005

Zusammenfassung

Wir beschäftigen uns in dieser Arbeit mit Feedback Vertex Set und Feedback Arc Set auf *gerichteten Graphen* im Kontext parametrisierter Algorithmen. Das Feedback-Vertex-Set-Problem besteht darin, in einem gegebenen Graphen eine Knotenmenge mit höchstens k Knoten zu finden, deren Löschen alle Kreise des Graphen zerstört. Feedback Arc Set ist das analoge Problem für Kantenlöschungen. Beide Probleme sind auf allgemeinen gerichteten Graphen NP-vollständig.

Wir geben einen Überblick über Komplexitäts- und Approximationsergebnisse für Feedback Vertex Set und Feedback Arc Set auf verschiedenen Klassen gerichteter Graphen. Zudem präsentieren wir einen Polynomialzeitalgorithmus zur Berechnung eines Problemkerns der Größe $k^2 + 2k$ für Feedback Arc Set auf Turniergraphen und stellen zwei neue parametrisierte Algorithmen vor:

- Einen $O(2^k \cdot n^2(\log n + k))$-Algorithmus für Feedback Vertex Set auf Turniergraphen.
- Einen $O(3{,}38^k \cdot n^6)$-Algorithmus für Feedback Arc Set auf bipartiten Turniergraphen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Viele Probleme, die bei der Behandlung praktischer Fragestellungen auftreten, lassen sich mit Graphen modellieren. Das Problem, in einem (gerichteten) Graphen alle Kreise aufzulösen, kommt dabei in zahlreichen Zusammenhängen vor. In dieser Arbeit untersuchen wir zwei bekannte Probleme, die sich mit der Zerstörung von Kreisen beschäftigen: Feedback Vertex Set und Feedback Arc Set auf gerichteten Graphen. Diese Probleme werden seit den sechziger Jahren untersucht und gehören zu den 21 NP-vollständigen Problemen, die Karp 1972 in seinem wegweisenden Artikel „Reducibility among combinatorial problems“ [Kar72] vorstellte. Feedback Vertex Set ist das Problem, für einen gegebenen gerichteten Graphen und eine Zahl $k \geq 0$ zu entscheiden, ob der Graph mit dem Löschen von k Knoten kreisfrei gemacht werden kann. Feedback Arc Set ist ein ähnliches Entscheidungsproblem; statt Knoten werden jedoch Kanten gelöscht. Ein Beispiel für jedes der beiden Probleme findet sich in Abbildung 1.1.

Eine Anwendung des Feedback-Vertex-Set-Problems ist die Auflösung von Deadlocks in Datenbanksystemen [RG00] oder bei der Prozessverwaltung in Betriebssystemen [Ros82]. In einem Datenbanksystem laufen zahlreiche Transaktionen, die jeweils auf Datenbankressourcen zugreifen. Da bei gleichzeitigem Lesen oder Schreiben derselben Ressource durch mehrere Transaktionen Konflikte auftreten können, sperrt jede Transaktion die von ihr momentan bearbeiteten Daten. Benötigt eine andere Transaktion währenddessen Zugriff auf die gesperrten Ressourcen, so muss sie warten, bis diese wieder freigegeben sind. Diese Situation können wir folgendermaßen mit einem gerichteten Graphen, einem sogenannten *Wartegraphen*, modellieren: Jede Transaktion entspricht einem Knoten des Graphen. Es existiert genau dann eine Kante zwischen den zwei Knoten v_i und v_j, wenn Transaktion T_i darauf wartet, dass Transaktion T_j eine von T_i benötigte Ressource freigibt. Jeder Kreis in diesem Graphen steht für eine Deadlocksituation:

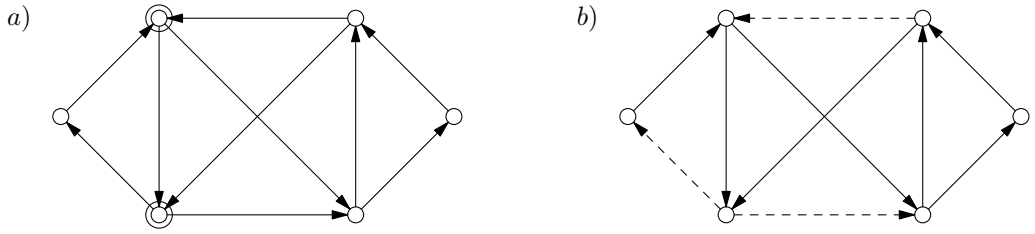

Abbildung 1.1: Beispiele für Feedback Vertex Set und Feedback Arc Set: Die markierten Knoten in a) bilden eine kleinste Knotenmenge, deren Löschen den gerichteten Graphen kreisfrei macht, die gestrichelten Kanten in b) eine entsprechende kleinste Kantenmenge.

Eine Anzahl von Transaktionen wartet zyklisch aufeinander. Diese Transaktionen und alle, die auf diese Transaktionen direkt oder indirekt warten, können nur dann weiterlaufen, wenn mindestens eine der Transaktionen auf dem Kreis abgebrochen wird. Natürlich möchte man möglichst wenige laufende Transaktionen abbrechen, sodass sich automatisch das Problem stellt, mit dem Abbruch möglichst weniger Transaktionen alle Deadlocks im System aufzulösen, also mit dem Löschen möglichst weniger Knoten des Graphen alle Kreise des Graphen zu zerstören.

Auch Feedback-Arc-Set-Probleme auf gerichteten Graphen kommen in der Praxis vor, zum Beispiel, wenn eine Anzahl von Elementen auf der Basis von Messwerten paarweise verglichen und in eine Rangfolge gebracht werden soll [ACN05]. Als Beispiel können wir uns n Sportler vorstellen, die in Einzelwettkämpfen gegeneinander antreten. Die Wettkampfbedingungen lassen keine unentschiedenen Partien zu. Auf Basis der Wettkampfergebnisse sollen die Sportler anschließend nach ihren Leistungen sortiert aufgelistet werden. Wenn ein Sportler s_i gegen einen Sportler s_j gewonnen hat, soll s_i in der Liste möglichst vor s_j stehen. Wie können wir eine Rangfolge bestimmen und dabei die Ergebnisse möglichst vieler Wettkämpfe berücksichtigen?

Diesen Sachverhalt modellieren wir in einem gerichteten Graphen, in dem jeder der Sportler $s_1, \ldots, s_n$ durch einen gleichnamigen Knoten repräsentiert ist. Die Kanten des Graphen bilden die Wettkampfergebnisse ab: Wenn Sportler s_i gegen Sportler s_j angetreten ist und gewonnen hat, fügen wir die Kante (s_i, s_j) in den Graphen ein. Tritt jeder der Sportler genau einmal gegen jeden anderen Sportler an, erhalten wir mit dieser Modellierung einen *Turniergraphen*. Bilden die Sportler zwei Mannschaften und tritt jeder Sportler einer Mannschaft gegen jeden Sportler der anderen Mannschaft genau einmal an, Sportler derselben Mannschaft aber nicht untereinander, so erhalten wir

einen *bipartiten Turniergraphen*.

Die gewünschte Rangfolge der Sportler erhalten wir durch eine *topologische Sortierung* der Knoten des Graphen, also eine Reihenfolge, in der die Knoten so angeordnet sind, dass alle Kanten, die von einem Knoten v ausgehen, auf Knoten zeigen, die in der Folge hinter v stehen. Eine solche Sortierung ist genau dann möglich, wenn der Graph kreisfrei ist. Da wir möglichst wenige Wettkampfergebnisse vernachlässigen wollen, stellt sich also das Problem, den Graphen mit dem Löschen möglichst weniger Kanten kreisfrei zu machen.

Wie wir oben erwähnt haben, sind Feedback Vertex Set und Feedback Arc Set auf allgemeinen gerichteten Graphen NP-vollständig, aller Voraussicht nach also nicht mit „effizienten“, d. h. polynomiellen Algorithmen lösbar. Da es trotz aller Schwierigkeiten nicht vermeidbar ist, diese Probleme in der Praxis zu lösen, werden verschiedene Ansätze verfolgt, um zu einer akzeptablen Lösung zu kommen. Man kann Problemlösungen approximieren, also mit effizienten Algorithmen eine Lösung finden, die nicht optimal, aber je nach Anwendung gut genug ist. Man kann randomisierte Algorithmen benutzen, die nicht mit Sicherheit, aber mit einer gewissen Wahrscheinlichkeit die beste Lösung ausgeben. Man kann Heuristiken anwenden, die Probleminstanzen oft schnell lösen, die jedoch im Worst-case keine optimale Lösung berechnen oder keine beweisbar effiziente Laufzeit aufweisen.

Eine andere Möglichkeit ist die Parametrisierung des Problems und die Nutzung parametrisierter Algorithmen [DF99, Nie06]. Zur Parametrisierung wird ein Parameter für das Problem gewählt, der in der Regel von der eingegebenen Probleminstanz abhängt und im Vergleich zur Größe der Eingabe klein ist. Dann versucht man einen Algorithmus zu finden, der in der Laufzeit $f(k) \cdot n^c$ läuft, wobei k der Parameter, n die Größe der Eingabe, c eine Konstante und f eine berechenbare Funktion ist, die nur von k abhängt. Einen solchen Algorithmus nennen wir *parametrisierten Algorithmus*. Ein solcher Algorithmus kann es bei geeigneter Wahl des Parameters ermöglichen, auch größere Probleminstanzen eines NP-vollständigen Problems exakt und effizient zu lösen.

In dieser Diplomarbeit wenden wir uns diesem Ansatz zu, geben zunächst einen Überblick über die Möglichkeiten, die er für Feedback Vertex Set und Feedback Arc Set nach bisherigem Kenntnisstand bietet und stellen im Anschluss eigene Ergebnisse vor. Wir benutzen im Folgenden immer die Größe der gesuchten Lösung als Parameter.

Es ist noch offen, ob Feedback Vertex Set und Feedback Arc Set auf allgemeinen gerichteten Graphen in FPT, der Klasse der mit parametrisierten Algorithmen lösbaren Probleme, sind. Parametrisierte Algorithmen für diese Probleme gibt es im Wesentlichen auf Turniergraphen und

Graphklasse	NP-vollständig?	in FPT?	Laufzeit des FPT-Algorithmus
Allgemeine ger. Graphen	Ja [Kar72]	offen	—
Turniergraphen	Ja [Spe89]	Ja (s. Kap. 5.1)	$O(2^k \cdot n^2(\lg n + k))$ (s. Kap. 5.1)[1]
bipartite Turniergraphen	Ja [CDZ02]	Ja (Reduktion auf 4-HS)	$O(3{,}12^k + n)$ (4-HS-Alg. aus [Fer05])

Abbildung 1.2: Resultate für Feedback Vertex Set auf den für diese Arbeit wichtigsten Klassen von gerichteten Graphen
[1]Anwendung der *Iterative-Compression*-Methode; Laufzeit des bisher schnellsten parametrisierten Algorithmus: $O(2{,}18^k + n)$ (3-Hitting-Set-Algorithmus nach [Fer04])

Graphklasse	NP-vollständig?	in FPT?	Laufzeit des FPT-Algorithmus
Allgemeine ger. Graphen	Ja [Kar72]	offen	—
Turniergraphen	Ja [Alo05, CTY05, Con05]	Ja [RS04]	$O(2{,}415^k \cdot n^{2{,}38})$ [RS04]
bipartite Turniergraphen	offen	Ja (s. Kap. 5.2)	$O(3{,}38^k \cdot n^6)$ (s. Kap. 5.2)[1]

Abbildung 1.3: Resultate für Feedback Arc Set auf gerichteten Graphen
[1]Charakterisierung durch „verbotene Teilgraphen“, zuvor kein nicht-trivialer parametrisierter Algorithmus bekannt.

bipartiten Turniergraphen. Abbildung 1.2 gibt einen Überblick über die für uns wichtigsten Ergebnisse für Feedback Vertex Set. Der beste parametrisierte Algorithmus für Feedback Vertex Set auf Turniergraphen ist bislang ein Algorithmus für 3-Hitting Set gewesen, weil sich Feedback Vertex Set auf Turniergraphen leicht auf dieses Problem reduzieren lässt. Aus dem selben Grund ist der beste FPT-Algorithmus für Feedback Vertex Set auf bipartiten Turniergraphen ein 4-Hitting-Set-Algorithmus. Abbildung 1.3 zeigt die aktuellen Ergebnisse für Feedback Arc Set.

Im weiteren Verlauf dieser Arbeit werden wir folgendermaßen vorgehen: Zunächst geben wir in Kapitel 2 eine Übersicht über die in dieser Arbeit verwendete Notation und die grundlegenden Definitionen der Graphentheorie

sowie der parametrisierten Komplexitätstheorie.

In Kapitel 3 stellen wir frühere Ergebnisse zur Komplexität von Feedback Vertex Set und Feedback Arc Set auf gerichteten Graphen vor und fassen einige bedeutende Eigenschaften dieser Probleme, besonders auf Turniergraphen und bipartiten Turniergraphen, zusammen.

Mit dem darauffolgenden Kapitel 4 beginnen wir mit der Präsentation eigener Ergebnisse. Wir zeigen eine Problemkernreduktion für Feedback Arc Set auf Turniergraphen. Dies ist ein Polynomialzeitalgorithmus, der den eingegebenen Turniergraphen so verkleinert, dass er höchstens noch $k^2 + 2k$ Knoten besitzt, wobei der Parameter k wie immer die Größe der gesuchten Lösung ist.

Danach stellen wir in Kapitel 5 zwei neue parametrisierte Algorithmen vor. Der Algorithmus FVST löst Feedback Vertex Set auf Turniergraphen mittels der Methode der *iterativen Kompression* in einer Laufzeit von $O(2^k \cdot n^2(\log n + k))$. Der zweite Algorithmus FASbT berechnet Feedback Arc Set auf bipartiten Turniergraphen mit Hilfe einer Charakterisierung durch „verbotene Teilgraphen“ in einer Laufzeit von $O(3{,}38^k \cdot n^6)$.

Zum Ende dieser Arbeit fassen wir in Kapitel 6 noch einmal unsere Ergebnisse zusammen und zeigen Anknüpfungspunkte für zukünftige Untersuchungen auf diesem Gebiet auf.

Kapitel 2

Definitionen und Notation

In diesem Kapitel werden wir die dieser Arbeit zugrunde liegende Notation und zahlreiche wichtige Grundbegriffe, vor allem der Graphentheorie und der parametrisierten Komplexität, formal definieren. Zunächst wenden wir uns allgemeinen Schreibweisen zu, danach einigen graphentheoretischen Grundlagen, der Definition wichtiger Graphklassen und Entscheidungsprobleme sowie zum Abschluss den wichtigsten Begriffen der parametrisierten Komplexitätstheorie.

2.1 Notation

Wir stellen in diesem Abschnitt einige generelle Definitionen und Sprachregelungen, die nicht der Graphentheorie entstammen, vor.

Besonders im Zusammenhang mit Feedback Vertex Sets und Feedback Arc Sets benutzen wir häufig den Begriff der *minimalen* und der *kleinsten* Menge. Da diese beiden Begriffe leicht verwechselt werden können, definieren wir sie hier zur Verdeutlichung. Ein Beispiel ist in Abbildung 2.1 zu sehen.

Definition 2.1.1 (*Minimale* Menge, *kleinste* Menge)
Eine durch eine bestimmte Eigenschaft X charakterisierte Menge heißt (bezüglich X) *minimal*, wenn die Entfernung eines beliebigen Elementes aus der Menge dazu führt, dass sie die Eigenschaft X verliert.

Im Gegensatz dazu nennen wir eine Menge, die die kleinste Menge ist, die die Eigenschaft X besitzt (entsprechend dem englischen *minimum*) die Menge *kleinster Größe* oder kurz *kleinste* Menge.

Daher ist jede kleinste Menge minimal, eine minimale Menge aber nicht immer eine kleinste. Analog zu minimalen und kleinsten Mengen können wir auch *maximale* und *größte* Mengen mit einer Eigenschaft X definieren.

Abbildung 2.1: Ein Beispiel zu Definition 2.1.1: a) Die markierten Knoten bilden ein *minimales* Feedback Vertex Set der Größe zwei für den abgebildeten Graphen, also eine *minimale* Knotenmenge, deren Löschen alle Kreise des Graphen zerstört. b) Der markierte Knoten ist ein *kleinstes* Feedback Vertex Set für den Graphen.

Als nächstes führen wir unsere Schreibweise für die *Potenzmenge* einer Menge X ein.

Definition 2.1.2 (*Potenzmenge* $\mathfrak{P}(X)$)
Sei X eine Menge. Dann nennen wir die Menge aller Teilmengen von X die *Potenzmenge* $\mathfrak{P}(X)$ von X.

Im Zusammenhang mit der Laufzeitbestimmung rekursiv aufgerufener Algorithmen begegnen uns später die Begriffe des *Verzweigungsvektors* und des *Verzweigungsfaktors*.

Definition 2.1.3 (*Verzweigungsvektor*)
Verzweigt sich ein rekursiver Algorithmus, in dem eine Rekursion bis zur Tiefe d durchgeführt werden soll, in l Unterprogrammaufrufe, die wiederum eine Rekursion bis zur Tiefe $d - x_1, \ldots, d - x_l$ mit $x_i \in \mathbb{N}^+$ durchführen, dann ist der zugehörige *Verzweigungsvektor* $(x_1, \ldots, x_l)$.

Definition 2.1.4 (*Verzweigungsfaktor*)
Für einen Verzweigungsvektor $(x_1, \ldots, x_l)$ ist der *Verzweigungsfaktor* b die einzige positive Lösung der Gleichung

$$\sum_{i=1}^{l} b^{-x_i} = 1$$

Der Verzweigungsvektor $(1, 1, 1)$ entspricht so zum Beispiel einem Verzweigungsfaktor von drei, der Vektor $(1, 1, 2, 2, 2, 2)$ einem Verzweigungsfaktor von 3,24.

Man kann sich den rekursiven Algorithmus als Suchbaum vorstellen, in dem jeder Programmaufruf ein Knoten ist und alle von einem Aufruf durchgeführten rekursiven Programmaufrufe die Kinder des entsprechenden Knotens sind. Wenn in einem solchen Suchbaum mit Verzweigungsfaktor b bis zu

einer Tiefe d gesucht werden soll, dann hat der Suchbaum insgesamt $O(b^d)$ Knoten. Ist im Suchalgorithmus für jeden Knoten des Suchbaumes ein Laufzeitaufwand von $O(f(n))$ erforderlich, so erhalten wir eine Gesamtlaufzeit von $O(b^d \cdot f(n))$.

Nach diesen allgemeinen Konventionen ist es an der Zeit, die wichtigsten Begriffe aus dem Bereich der Graphentheorie zu klären.

2.2 Graphentheorie

In dieser Arbeit werden zahlreiche Begriffe aus der Graphentheorie genutzt, die in diesem Abschnitt einleitend definiert werden sollen. Wir beschäftigen uns in den nachfolgenden Kapiteln vor allem mit Feedback-Set-Problemen auf *gerichteten Graphen*.

Definition 2.2.1 (*Gerichteter Graph*)
Ein *gerichteter Graph* G ist ein Paar (V, E) mit

- einer endlichen Menge V von *Knoten*
- und einer Menge $E \subseteq V \times V$ von gerichteten *Kanten*.

Die Kanten eines gerichteten Graphen sind geordnet, dadurch unterscheidet er sich vom *ungerichteten Graphen*.

Definition 2.2.2 ((*Ungerichteter*) *Graph*)
Ein *ungerichteter Graph* $G = (V, E)$ wird analog zum gerichteten Graphen definiert, besitzt jedoch eine Kantenmenge $E \subseteq \{\{u, v\} \mid u, v \in V\}$.

Da die Kanten in gerichteten Graphen geordnete Paare sind, kann ein Knoten u auf zwei unterschiedliche Weisen mit einem Knoten v verbunden sein. Dies rechtfertigt die Einführung zweier unterschiedlicher Begriffe der Nachbarschaft von Knoten in gerichteten Graphen.

Definition 2.2.3 (*Nachfolger, Vorgänger*)
In einem gerichteten Graphen $G = (V, E)$ ist ein Knoten $u \in V$ *Vorgänger* eines Knotens $v \in V$, wenn $(u, v) \in E$. Umgekehrt ist ein Knoten u *Nachfolger* eines Knotens v, wenn $(v, u) \in E$.
Die Menge der Vorgänger von v in G bezeichnen wir mit

$$N_G^-(v) := \{u \mid (u, v) \in E\},$$

die der Nachfolger von v mit

$$N_G^+(v) := \{u \mid (v, u) \in E\}.$$

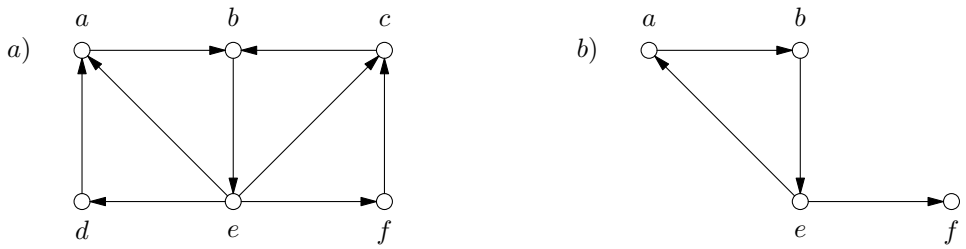

Abbildung 2.2: Ein Beispiel für einen induzierten Teilgraphen: a) Ein gerichteter Graph $G = (V, E)$. b) Der induzierte Teilgraph $G[V'] = (V', E[V'])$ für $V' = \{a, b, e, f\}$.

Um die Anzahl der Vorgänger und Nachfolger eines Knotens v zu erfassen, definieren wir *Eingangs-* und *Ausgangsgrad* von v.

Definition 2.2.4 (*Eingangsgrad, Ausgangsgrad*)
Für einen Knoten $v \in V$ eines gerichteten Graphen G heißt $d_G^-(v) := |N_G^-(v)|$ *Eingangsgrad* und $d_G^+(v) := |N_G^+(v)|$ *Ausgangsgrad* von v.

Wenn keine Verwechslungsgefahr besteht, schreiben wir anstelle von $d_G^-(v)$ und $d_G^+(v)$ auch $d^-(v)$ und $d^+(v)$.

Eine im Folgenden sehr häufig genutzte Notation ist $G[V']$ als Bezeichnung für den durch eine Knotenmenge V' *induzierten Teilgraphen* eines Graphen G.

Definition 2.2.5 (*Induzierter Teilgraph $G[V']$*)
Sei $G = (V, E)$ ein (gerichteter) Graph und $V' \subseteq V$ eine Teilmenge der Knoten. Dann heißt $G[V'] := (V', E[V'])$ mit der *induzierten Kantenmenge*

$$E[V'] := \{(u, v) \mid u, v \in V' \text{ und } (u, v) \in E\} \text{ (für gerichtete Graphen)}$$

bzw.

$$E[V'] := \{\{u, v\} \mid u, v \in V' \text{ und } \{u, v\} \in E\} \text{ (für ungerichtete Graphen)}$$

der durch V' *induzierte Teilgraph* von G. (Beispiel siehe Abbildung 2.2.)

Definition 2.2.6 (*Weg, Pfad*)
Sei $G = (V, E)$ ein gerichteter Graph. Eine Folge $v_1 v_2 \dots v_n$ von Knoten aus V heißt *Weg* von v_1 nach v_n, wenn für $1 \leq i \leq n-1$ gilt: $(v_i, v_{i+1}) \in E$.

Einen Weg, auf dem alle Knoten verschieden sind, nennen wir *Pfad*.

Wege und Pfade definieren wir analog auch für ungerichtete Graphen.

Wenn wir uns mit Feedback-Set-Problemen beschäftigen, ist der Begriff des *Kreises* essentiell.

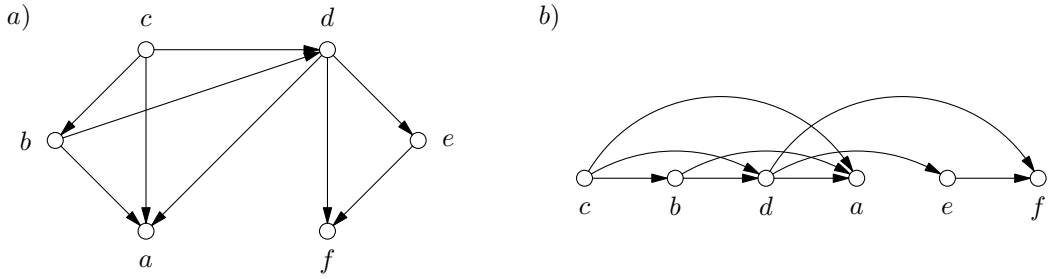

Abbildung 2.3: Ein Beispiel zu Definition 2.2.9: a) Ein gerichteter, kreisfreier Graph $G = (\{a, b, c, d, e, f\}, E)$. b) Die Folge c, b, d, a, e, f ist eine topologische Sortierung der Knoten des Graphen.

Definition 2.2.7 (*Kreis, Dreieck, Viereck*)
Einen Weg $v_1 v_2 \dots v_n$ in einem Graphen nennen wir *Kreis* der Länge $n - 1$, wenn $v_1 = v_n$ gilt und ansonsten alle Knoten des Weges voneinander verschieden sind.

Einen Kreis der Länge drei nennen wir auch *Dreieck*, einen Kreis der Länge vier *Viereck*.

Die Definition gerichteter Pfade ermöglicht uns den Begriff des *starken Zusammenhangs* von gerichteten Graphen.

Definition 2.2.8 (*Stark zusammenhängender gerichteter Graph, starke Zusammenhangskomponente*)
Ein gerichteter Graph heißt *stark zusammenhängend*, wenn in ihm von jedem Knoten zu jedem anderen Knoten ein gerichteter Pfad existiert.

Ein Teilgraph eines gerichteten Graphen G heißt *starke Zusammenhangskomponente*, wenn er ein maximaler stark zusammenhängender Teilgraph von G ist.

Um kreisfreie Graphen zu charakterisieren, ist die *topologische Sortierung* der Knoten von Interesse.

Definition 2.2.9 (*Topologische Sortierung*)
Sei $G = (V, E)$ ein gerichteter Graph. Eine Permutation $v_1, \dots, v_{|V|}$ der Knoten von G heißt *topologische Sortierung*, falls für alle $i, j \in \{1, \dots, |V|\}$ gilt: $(v_i, v_j) \in E \Rightarrow i < j$.

Ein Beispiel für eine topologische Sortierung ist in Abbildung 2.3 zu sehen. Es ist genau dann möglich, die Knoten eines gerichteten Graphen G topologisch zu sortieren, wenn G kreisfrei ist, da in einem Graphen, dessen

Knoten topologisch sortiert sind, alle Kanten „von links nach rechts“, also von Knoten mit kleinerem Index zu Knoten mit größerem Index in der Folge zeigen, sodass sich kein Kreis schließen kann, denn dafür müsste es eine Kante „von rechts nach links“ geben.

An dieser Stelle möchten wir einige Graphklassen definieren, für die in dieser Arbeit Ergebnisse zusammengefasst oder neu vorgestellt werden. Die wichtigsten Graphklassen sind hierbei der *Turniergraph* und der *bipartite Turniergraph*.

Definition 2.2.10 (*Turniergraph*)
Ein gerichteter Graph $T = (V, E)$ heißt *Turniergraph*, wenn für jeweils zwei Knoten $u, v \in V$, $u \neq v$ genau eine der Kanten (u, v) und (v, u) in E enthalten ist.

In einem kreisfreien Turniergraphen ist die topologische Sortierung der Knoten eindeutig, da die Position zweier beliebiger Knoten untereinander durch die Richtung der Kante zwischen diesen Knoten festgelegt ist. Wie man leicht sieht, haben die Knoten der topologisch sortierten Folge dann die Eingangsgrade $0, 1, \ldots, |V| - 1$. Umgekehrt ist auch jeder Turniergraph, in dem alle Eingangsknotengrade unterschiedlich sind, kreisfrei.

Während der Turniergraph in der Literatur häufig zu finden ist, ist der *bipartite Turniergraph* relativ unbekannt, er kann jedoch zum Beispiel bei der Modellierung paarweiser Vergleiche zwischen Mitgliedern zweier Gruppen entstehen (s. Kapitel 1) und wurde von Cai et al. [CDZ02] bereits im Zusammenhang mit Feedback Vertex Set betrachtet.

Definition 2.2.11 (*Bipartiter Turniergraph*)
Ein gerichteter Graph $T = (V, E)$ heißt *bipartiter Turniergraph*, wenn es eine Zerlegung von V in zwei disjunkte Mengen V_1, V_2 gibt, sodass die von diesen Mengen induzierten Teilgraphen $T[V_1]$ und $T[V_2]$ keine Kanten enthalten und für beliebige zwei Knoten $u \in V_1, v \in V_2$ gilt: Genau eine der Kanten (u, v) und (v, u) ist in E enthalten.

Eine viel untersuchte Graphklasse, die in dieser Arbeit eher am Rande vorkommt, ist der *planare Graph*.

Definition 2.2.12 (*Planarer Graph*)
Ein (gerichteter oder ungerichteter) Graph heißt *planar*, wenn es eine Möglichkeit gibt, ihn so in eine Ebene einzubetten, dass sich keine Kanten überschneiden.

Nach dieser kurzen Einführung in die Graphentheorie möchten wir diejenigen Probleme formal definieren, mit denen wir uns hier befassen.

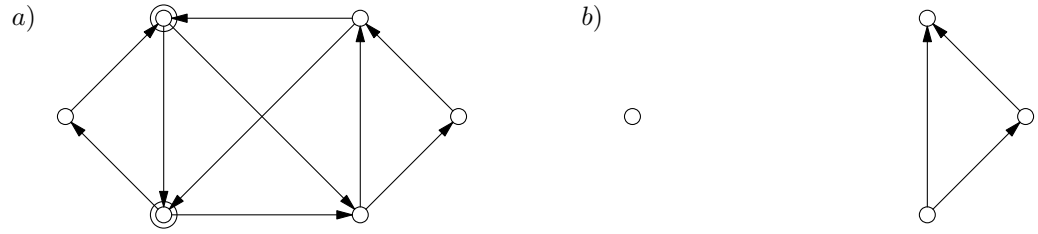

Abbildung 2.4: Ein Beispiel für Feedback Vertex Set: a) Ein gerichteter Graph. Die beiden markierten Knoten bilden ein kleinstes Feedback Vertex Set für den Graphen. b) Nach dem Entfernen der Knoten des Feedback Vertex Sets ist der Graph kreisfrei.

2.3 Entscheidungsprobleme

Die beiden wichtigsten Probleme, die wir in dieser Arbeit betrachten, sind die *Feedback-Set-Probleme* Feedback Vertex Set und Feedback Arc Set.

Definition 2.3.1 (Feedback Vertex Set (FVS))

Eingabe: Ein (gerichteter oder ungerichteter) Graph $G = (V, E)$ und eine natürliche Zahl $k \geq 0$.

Frage: Gibt es eine Menge $F \subseteq V$ mit $|F| \leq k$, sodass $G[V \setminus F]$ keine (gerichteten bzw. ungerichteten) Kreise besitzt?

Eine Menge F von Knoten, deren Entfernen G kreisfrei macht, nennen wir – unabhängig von ihrer Größe – ein *Feedback Vertex Set (FVS)* (Beispiel siehe Abbildung 2.4).

Das Feedback-Arc-Set-Problem ist Feedback Vertex Set sehr ähnlich, statt Knoten werden jedoch Kanten gelöscht.

Definition 2.3.2 (Feedback Arc Set (FAS))

Eingabe: Ein gerichteter Graph $G = (V, E)$ und eine natürliche Zahl $k \geq 0$.

Frage: Gibt es eine Menge $F \subseteq E$ mit $|F| \leq k$, sodass der Graph $(V, E \setminus F)$ keine gerichteten Kreise besitzt?

Eine Kantenmenge, deren Löschung G kreisfrei macht, nennen wir – unabhängig von ihrer Größe – ein *Feedback Arc Set (FAS)* (Beispiel siehe Abbildung 2.5).

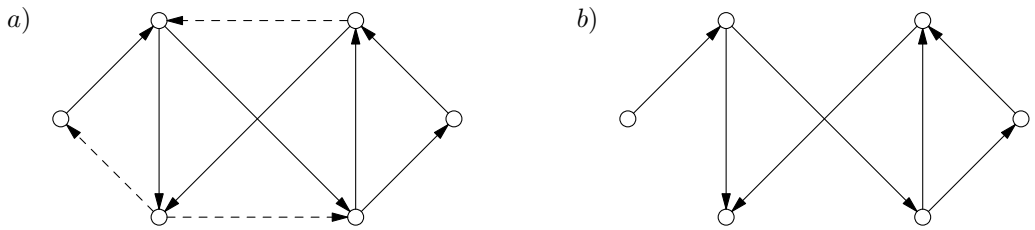

Abbildung 2.5: Ein Beispiel für Feedback Arc Set: a) Ein gerichteter Graph. Die gestrichelten Kanten markieren ein kleinstes Feedback Arc Set für den Graphen. b) Nach dem Entfernen der Kanten des Feedback Arc Sets existieren im Graphen keine gerichteten Kreise mehr.

Das analog zu Feedback Arc Set definierte Problem für ungerichtete Graphen nennen wir Feedback Edge Set.

Ein Problem, das auch im Zusammenhang mit Feedback-Set-Problemen interessant ist, weil sich Feedback Vertex Set einfach darauf reduzieren lässt, ist das Hitting-Set-Problem.

Definition 2.3.3 (Hitting Set (HS))

Eingabe: Eine Grundmenge S von Elementen, eine Menge $C \subseteq \mathfrak{P}(S)$ von Teilmengen von S und eine natürliche Zahl $k \geq 0$.

Frage: Gibt es eine Teilmenge von S, die mindestens ein Element aus jeder Menge in C enthält (ein sogenanntes *Hitting Set*) und höchstens eine Größe von k hat?

Besonders häufig begegnet uns in diesem Zusammenhang der Spezialfall, dass die Anzahl der Elemente in den Mengen aus C durch eine Konstante d beschränkt ist. Hier sprechen wir vom d-Hitting-Set-Problem.

Definition 2.3.4 (d-Hitting Set (d-HS))

Eingabe: Eine Grundmenge S von Elementen, eine Menge $C \subseteq \mathfrak{P}(S)$ von d-elementigen oder kleineren Teilmengen von S und eine natürliche Zahl $k \geq 0$.

Frage: Gibt es eine Teilmenge von S, die mindestens ein Element aus jeder Menge in C enthält und höchstens eine Größe von k hat?

Nachdem wir die für diese Arbeit wichtigsten graphentheoretischen Begriffe und Probleme eingeführt haben, wenden wir uns jetzt den Grundbegriffen der parametrisierten Komplexitätstheorie zu.

2.4 Parametrisierte Komplexitätstheorie

Um das grundlegende Konzept der parametrisierten Komplexitätstheorie zu erklären, ist vor allem der Begriff des *parametrisierten Problems* wichtig.

Definition 2.4.1 (*Parametrisiertes Problem, Parameter*)
Ein *parametrisiertes Problem* ist eine Sprache $L \subseteq \Sigma^* \times \Sigma^*$, wobei Σ ein endliches Alphabet ist. Die zweite Komponente des Problems heißt *Parameter*.

Für uns sind in dieser Arbeit nur natürliche Zahlen als Parameter interessant; alle parametrisierten Probleme, die hier vorkommen, sind also Sprachen $L \subseteq \Sigma^* \times \mathbb{N}$. Der Parameter ist eine Größe, die gewöhnlich direkt oder indirekt in der Eingabe enthalten ist. Im Folgenden betrachten wir ausschließlich Parametrisierungen, in denen der Parameter die Größe der Lösung, also in der Regel die Größe des Feedback Vertex Sets oder Feedback Arc Sets ist.

Wir definieren zunächst den Begriff des *parametrisierten Algorithmus*, der Grundlage der parametrisierten Komplexitätstheorie ist.

Definition 2.4.2 (*Parametrisierter Algorithmus*)
Sei X ein parametrisiertes Entscheidungsproblem. Dann ist ein *parametrisierter Algorithmus* für X ein Algorithmus, der für jede Probleminstanz (x, k) mit $|x| = n$ in einer Laufzeit von $f(k) \cdot n^c$ entscheidet, ob $(x, k) \in X$ gilt. Dabei ist c eine Konstante und f eine berechenbare Funktion, die nur von k abhängt.

Der Vorteil eines parametrisierten Algorithmus ist also, dass die Laufzeit nur polynomiell von der Größe der Eingabe abhängt und sich die Schwierigkeit des Problems im Wesentlichen im Vorfaktor $f(k)$ niederschlägt, der nur vom Parameter k abhängig ist. Dieser wird in der Regel so gewählt, dass er im Vergleich zur Eingabelänge klein ist.

Es ist nicht für alle NP-vollständigen Probleme möglich, einen parametrisierten Algorithmus mit jedem sinnvoll erscheinenden Parameter zu finden, es sei denn, dass P = NP. Ein Beispiel ist das k-Färbbarkeitsproblem, bei dem zu entscheiden ist, ob sich die Knoten eines gegebenen Graphen mit k Farben so färben lassen, dass keine zwei benachbarten Knoten dieselbe Farbe besitzen. Für dieses Problem mit der Färbungszahl k als Parameter gibt es unter der Voraussetzung P $\neq$ NP keinen parametrisierten Algorithmus, da ansonsten das NP-vollständige 3-Färbbarkeitsproblem in Polynomialzeit lösbar wäre. Der Begriff des parametrisierten Algorithmus ermöglicht also die Einführung einer neuen Komplexitätsklasse, die alle parametrisierten Probleme enthält, für die es einen solchen Algorithmus gibt.

Definition 2.4.3 (*Festparameter-handhabbar, FPT*)
Ein Problem nennen wir *festparameter-handhabbar* (englisch *fixed-parameter tractable*), wenn es einen parametrisierten Algorithmus für dieses Problem gibt. Die Menge aller Festparameter-handhabbaren Probleme heißt *FPT*.

Eine andere Möglichkeit, ein parametrisiertes Problem aus FPT zu lösen, ist, für die eingegebene Instanz einen *Problemkern* zu berechnen – eine kleinere äquivalente Probleminstanz, deren Größe nur vom Parameter abhängig ist.

Definition 2.4.4 (*Problemkern*)
Ein Entscheidungsproblem X besitzt einen *Problemkern*, wenn es einen Polynomialzeitalgorithmus gibt, der für eine Eingabe (I, k), wobei I eine Instanz von X und k ein Parameter ist, eine Ausgabe (I', k') berechnet, sodass gilt:

- $(I, k) \in X$ gdw. $(I', k') \in X$,
- $|I'| \leq f(k)$, wobei f eine Funktion ist, die nur von k abhängt, und
- $k' \leq k$.

Den Algorithmus, der für eine gegebene Probleminstanz einen Problemkern berechnet, nennen wir *Problemkernreduktion*.

Es ist eine bekannte Tatsache, dass es genau dann einen Problemkern für ein Problem X mit Parameter k gibt, wenn ein mit k parametrisierter Algorithmus für X existiert.

Da Problemkernreduktionen Probleminstanzen in neue Probleminstanzen überführen, deren Größe nur von einem – in der Regel kleinen – Parameter abhängen, können sie helfen, in der Praxis schwierige Probleme zu lösen. Denn wenn es gelingt, Probleminstanzen hinreichend zu verkleinern, kann auch bei NP-vollständigen Problemen ein exakter, nicht-parametrisierter Algorithmus eingesetzt werden.

Kapitel 3

Frühere Resultate im Überblick

Feedback Vertex Set und Feedback Arc Set auf allgemeinen gerichteten Graphen sind schon lange Zeit als schwierige Probleme bekannt, da ihre NP-Vollständigkeit bereits 1972 von Karp in seinem bedeutenden Artikel „Reducibility among combinatorial problems“ [Kar72] bewiesen wurde.

Diese Probleme sind seitdem weiter untersucht worden, und obwohl Feedback Vertex Set und Feedback Arc Set nicht so umfassend verstanden sind wie Vertex Cover und andere der von Karp betrachteten Entscheidungsprobleme, so gibt es doch zahlreiche interessante Ergebnisse, die die Komplexität und Approximierbarkeit dieser Probleme auf unterschiedlichen Graphklassen betreffen.

Viele Komplexitäts- und Approximationsergebnisse treffen auf Feedback Vertex Set und Feedback Arc Set auf allgemeinen gerichteten Graphen gleichermaßen zu, weil die kantengewichteten Versionen dieser Probleme aufeinander reduzierbar sind (siehe z. B. [ENSS98]), ohne dass sich die Größe der gesuchten Lösung ändert. Die charakterisierenden Eigenschaften der meisten spezielleren Graphklassen werden bei der Reduktion jedoch nicht erhalten, sodass Ergebnisse z. B. für Turniergraphen nicht übernommen werden können.

Wir möchten hier einen Überblick über die wichtigsten Komplexitätsergebnisse für gerichtete Graphen geben und anschließend einige Lemmata vorstellen, die wichtige Eigenschaften von Turniergraphen und gerichteten Graphen im Hinblick auf Feedback-Set-Probleme aufzeigen und die wir später in Beweisen benötigen.

Graphklasse	Literatur
Reduzierbare Flussgraphen	[Sha79]
Zyklisch reduzierbare Graphen	[WLS85]
fvs-einfache gerichtete Graphen	[Spe89]

Abbildung 3.1: Klassen gerichteter Graphen, auf denen Feedback Vertex Set in Polynomialzeit lösbar ist

3.1 Die Komplexität von Feedback Vertex Set auf gerichteten Graphen

Das Wissen um die NP-Vollständigkeit von Feedback Vertex Set auf allgemeinen gerichteten Graphen macht es besonders interessant, Klassen gerichteter Graphen zu finden, auf denen sich Feedback Vertex Set in Polynomialzeit lösen lässt.

Shamir veröffentlichte 1979 einen Linearzeitalgorithmus für sogenannte reduzierbare Flussgraphen [Sha79]. Die Klasse der reduzierbaren Flussgraphen war damit die erste nichttriviale Graphklasse mit einem Polynomialzeitalgorithmus für Feedback Vertex Set. Wenige Jahre später definierten Wang, Lloyd und Soffa die Klasse der zyklisch reduzierbaren Graphen und gaben einen Polynomialzeitalgorithmus für diese Graphen an [WLS85].

Speckenmeyer entwickelte eine Datenreduktionsprozedur mit polynomieller Laufzeit für Feedback-Vertex-Set-Instanzen [Spe89]. Diejenigen Graphen, für die man bereits mit dieser Prozedur ein kleinstes Feedback Vertex Set finden kann, nannte er fvs-einfache gerichtete Graphen. Er konnte beweisen, das diese Graphklasse sowohl die reduzierbaren Flussgraphen als auch die zyklisch reduzierbaren Graphen umfasst.

Dennoch gibt es nur wenige Graphklassen, für die bekannt ist, das sich Feedback Vertex Set in Polynomialzeit lösen lässt. Gerade für einige bekannte und praxisrelevante Graphklassen ist die NP-Vollständigkeit des Problems bewiesen. Einige Jahre nach Karps NP-Vollständigkeitsbeweis für Feedback Vertex Set auf allgemeinen gerichteten Graphen [Kar72] zeigte Yannakakis, dass das Problem auch auf planaren gerichteten Graphen NP-vollständig ist [Yan78]. Kurz darauf konnten Garey und Johnson beweisen, dass Feedback Vertex Set selbst dann NP-hart bleibt, wenn man Ein- und Ausgangsgrad bei allgemeinen gerichteten Graphen auf jeweils zwei und bei planaren Graphen auf jeweils drei beschränkt [GJ79]. Speckenmeyer bewies später die NP-Vollständigkeit des Problems auf Turniergraphen [Spe89], und Cai et al. konnten vor wenigen Jahren dasselbe für bipartite Turniergraphen zeigen [CDZ02].

Graphklasse	Literatur
Allgemeine gerichtete Graphen	[Kar72]
Turniergraphen	[Spe89]
Bipartite Turniergraphen	[CDZ02]
Gerichtete Graphen mit Ein- und Ausgangsgrad ≤ 2	[GJ79]
Planare gerichtete Graphen	[Yan78]
Planare gerichtete Graphen mit Ein- und Ausgangsgrad ≤ 3	[GJ79]

Abbildung 3.2: Klassen gerichteter Graphen, auf denen Feedback Vertex Set NP-vollständig ist

Graphklasse	Approximationsfaktor	Literatur
Ungerichtete Graphen	2	[BBF99]
Allgemeine ger. Graphen	$O(\log n \log \log n)$	[ENSS98]
Turniergraphen	2,5	[CDZ01]
Bipartite Turniergraphen	3,5	[CDZ02]
Planare gerichtete Graphen	2,25	[GW96]

Abbildung 3.3: Approximationsergebnisse für Feedback Vertex Set auf unterschiedlichen Graphklassen

Ein üblicher Ansatz, NP-vollständige Optimierungsprobleme in der Praxis anzugehen, ist die Approximation. Approximationsalgorithmen laufen in Polynomialzeit und geben eine Lösung aus, die nicht optimal ist, deren Größe aber innerhalb beweisbarer Schranken liegt.

Es ist bekannt, dass Feedback Vertex Set auf gerichteten Graphen APX-hart ist, d. h. dass es höchstwahrscheinlich nicht mit beliebig nah an eins liegendem konstantem Faktor in Polynomialzeit approximiert werden kann [Kan92]. Die beste bisher bekannte Approximation für Feedback Vertex Set auf allgemeinen gerichteten Graphen besitzt einen Approximationsfaktor von $O(\log n \log \log n)$ [ENSS98] bzw. der Länge des längsten einfachen Kreises im Graphen [DF03], während für Feedback Vertex Set auf ungerichteten Graphen eine Approximation mit konstantem Approximationsfaktor zwei existiert [BBF99]. Es gibt jedoch auch einige Klassen gerichteter Graphen, für die Approximationsalgorithmen mit konstantem Approximationsfaktor bekannt sind. Dazu gehören zum einen planare Graphen [GW96], zum anderen die für diese Arbeit zentralen Graphklassen, die Turniergraphen und bipartiten Turniergraphen, für die Cai et al. Approximationsfaktoren von 2,5 und 3,5 [CDZ01, CDZ02] erzielen konnten. Für Feedback Vertex Set auf Turniergraphen kennen wir zudem auch eine Untergrenze für die Approximierbarkeit. Da Speckenmeyer für den Beweis der NP-Härte [Spe89]

Graphklasse	Literatur
Planare gerichtete Graphen	[Luc76, GLS88]
Reduzierbare Flussgraphen	[Ram88]

Abbildung 3.4: Klassen gerichteter Graphen, auf denen Feedback Arc Set in Polynomialzeit lösbar ist

das Vertex-Cover-Problem mit einer sogenannten *L-Reduktion* [PY91] auf Feedback Vertex Set auf Turniergraphen reduzierte, können wir alle Resultate zur Nicht-Approximierbarkeit von Vertex Cover übertragen. Daher kann es unter der Voraussetzung, dass $P \neq NP$ gilt, keinen besseren Approximationsfaktor als 1,36 [DS05] für Feedback Vertex Set auf Turniergraphen geben.

Ob Feedback Vertex Set mit der Lösungsgröße als Parameter in FPT ist, ist ein lange offenes Problem. Auf Turniergraphen und bipartiten Turniergraphen ist Feedback Vertex Set in FPT, da es sich einfach auf 3- bzw. 4-Hitting Set reduzieren lässt und parametrisierte Algorithmen für d-Hitting Set bekannt sind [Fer04, Fer05]. Die Reduktion von Feedback Vertex Set auf Turniergraphen auf 3-Hitting Set ist in Kapitel 5.1 beschrieben, die Reduktion von Feedback Vertex Set auf bipartiten Turniergraphen auf 4-Hitting Set erfolgt analog. Der beste Algorithmus auf bipartiten Turniergraphen ist ein 4-Hitting-Set-Algorithmus von Fernau mit einer Laufzeit von $O(3{,}12^k+n)$ [Fer05]. Auf Turniergraphen erreichen wir mit unserem Algorithmus in Kapitel 5.1 eine Laufzeit von $O(2^k \cdot n^2(\lg n+k))$, während der schnellste 3-Hitting-Set-Algorithmus, der bisher schnellster parametrisierter Algorithmus für Feedback Vertex Set auf Turniergraphen war, eine Laufzeit von $O(2{,}18^k + n)$ besitzt [Fer04].

3.2 Die Komplexität von Feedback Arc Set

Für Feedback Arc Set sind weniger Ergebnisse bekannt als für Feedback Vertex Set, obwohl dieses Problem früher definiert und – besonders auf Turniergraphen – schon Anfang der sechziger Jahre untersucht wurde [Sla61].

Das analoge Problem Feedback Edge Set auf ungerichteten Graphen ist einfach in Linearzeit lösbar, da die Menge der Kanten, die in einem *minimalen Spannbaum* für einen ungerichteten Graphen nicht vorkommen, ein kleinstes Feedback Edge Set bilden. Für Feedback Arc Set sind jedoch kaum Graphklassen bekannt, auf denen das Problem in Polynomialzeit lösbar ist. Lucchesi zeigte, dass Feedback Arc Set auf planaren Graphen in P

Graphklasse	Literatur
Allgemeine gerichtete Graphen	[Kar72]
Turniergraphen	[Alo05, CTY05, Con05]
Gerichtete Graphen mit Ein-/Ausgangsgrad ≤ 3	[Gav77]

Abbildung 3.5: Klassen gerichteter Graphen, auf denen Feedback Arc Set NP-vollständig ist

Graphklasse	Approximationsfaktor	Literatur
Allgemeine gerichtete Graphen	$O(\log n \log \log n)$	[ENSS98]
Turniergraphen	2	[vZ05]

Abbildung 3.6: Approximationsergebnisse für Feedback Arc Set

ist [Luc76], und Ramachandran bewies die effiziente Lösbarkeit des Problems auf reduzierbaren Flussgraphen [Ram88].

Auch wenn seit schon lange bekannt ist, dass Feedback Arc Set NP-vollständig ist [Kar72], sogar wenn Ein- und Ausgangsgrad nicht größer als drei sind [Gav77], ist lange Zeit offen gewesen, ob Feedback Arc Set auf Turniergraphen NP-vollständig ist. Diese Vermutung von Bang-Jensen und Thomassen [BT92] wurde erst kürzlich von Ailon et al. [ACN05] aufgegriffen, die bewiesen, dass Feedback Arc Set auf Turniergraphen unter randomisierter Reduktion NP-vollständig ist. Kurz darauf zeigten Noga Alon [Alo05] und Charbit et al. [CTY05] unabhängig voneinander die NP-Vollständigkeit des Problems, indem sie einen ähnlichen Ansatz wie den von Ailon et al. verwendeten und derandomisierten. Zusätzlich untersuchte Conitzer das Problem unter dem Namen *Slater Ranking* und konnte ebenfalls seine NP-Vollständigkeit beweisen, indem er MAXSAT darauf reduzierte [Con05].

Wie Feedback Vertex Set ist auch Feedback Arc Set auf gerichteten Graphen APX-vollständig, und es ist bisher keine Approximation mit einem besseren Faktor als $\log n \log \log n$ [ENSS98] bekannt. Vor kurzem präsentierten Ailon et al. eine randomisierte 3-Approximation für Feedback Arc Set auf Turniergraphen, die van Zuylen derandomisieren konnte [vZ05]. Für Turniergraphen mit gewichteten Kanten, deren Gewichte die Dreiecksungleichung erfüllen, also insbesondere auch für ungewichtete Turniergraphen, erreichte sie dabei einen Approximationsfaktor von zwei.

Die Frage, ob Feedback Arc Set mit der Lösungsgröße als Parameter in FPT ist, ist gleichbedeutend mit der Frage, ob Feedback Vertex Set in FPT ist und daher bislang ebenfalls offen. Die Festparameter-Handhabbarkeit des Problems auf Turniergraphen zeigten Raman und Saurabh, die außerdem einen parametrisierten Algorithmus mit einer Laufzeit

von $O(2{,}415^k \cdot n^\omega)$ angeben konnten [RS04]. Dass Feedback Arc Set auf bipartiten Turniergraphen in FPT ist, ist analog dazu einfach zu erkennen – in Kapitel 5.2 gehen wir genauer darauf ein. Dort geben wir zunächst einen einfachen parametrisierten Algorithmus mit einer Laufzeit von $O(4^k \cdot n^{2,38})$ an und stellen im Anschluss daran einen verbesserten Algorithmus mit Laufzeit $O(3{,}38^k \cdot n^6)$ vor.

Raman und Saurabh konnten zudem zeigen, dass Feedback Arc Set auf dichten gerichteten Graphen, d. h. Graphen, denen gegenüber einem Turniergraphen nur $O(n^{1+o(1)})$ Kanten fehlen, ebenfalls festparameter-handhabbar ist [RS04].

3.3 Aussagen über gerichtete Graphen

Dieser Abschnitt enthält einige grundlegende und bekannte Hilfssätze, die wichtige Eigenschaften von Turniergraphen, bipartiten Turniergraphen und Feedback Arc Sets von gerichteten Graphen charakterisieren und die wir später in Beweisen verwenden werden.

Zunächst zeigen wir, dass in einem Turniergraphen jeder Knoten, der zu einem beliebigen Kreis gehört, auch zu einem Dreieck gehört.

Lemma 3.3.1 *In einem Turniergraphen $T = (V, E)$ liegt jeder Knoten, der auf einem gerichteten Kreis liegt, auch auf einem gerichteten Kreis der Länge drei.*

Beweis: Angenommen, es gäbe einen Knoten $v_1 \in V$, der nur auf gerichteten Kreisen einer Länge größer als drei liegt. Sei der Kreis $v_1 v_2 \cdots v_n v_1$, $n > 3$, ein kürzester Kreis durch v_1. Weil T ein Turniergraph ist, muss entweder (v_1, v_3) oder (v_3, v_1) in E enthalten sein.

Fall 1: $(v_1, v_3) \in E$

In diesem Fall liegt v_1 auf dem Kreis $v_1 v_3 \cdots v_n v_1$. Dies steht im Widerspruch zur Annahme, dass $v_1 v_2 \cdots v_n v_1$ der kürzeste Kreis durch v_1 ist.

Fall 2: $(v_3, v_1) \in E$

Dann liegt v_1 auf dem gerichteten Dreieck $v_1 v_2 v_3 v_1$, ebenfalls im Widerspruch zur Annahme. □

Die folgende Aussage ist eine unmittelbare Folgerung aus Lemma 3.3.1.

Korollar 3.3.1 *Ein Turniergraph T ist genau dann kreisfrei, wenn er keine Kreise der Länge drei enthält.* □

Analog zu Lemma 3.3.1 gilt:

Lemma 3.3.2 *In einem bipartiten Turniergraphen $T = (V, E)$ liegt jeder Knoten, der auf einem gerichteten Kreis liegt, auch auf einem gerichteten Kreis der Länge vier.*

Beweis: Sei v_1 ein beliebiger Knoten aus V, der auf einem gerichteten Kreis liegt. Angenommen, ein kürzester solcher Kreis wäre der Pfad $v_1v_2v_3 \cdots v_nv_1$ mit geradem $n > 4$.

Weil T ein bipartiter Turniergraph ist, gibt es jeweils eine Kante zwischen einem Knoten v_i mit geradem Index i und einem Knoten v_j mit ungeradem Index j. Also muss es in E entweder die Kante (v_1, v_4) oder die Kante (v_4, v_1) geben.

Fall 1: Es gibt die Kante (v_1, v_4).

Dann liegt v_1 auf dem kürzeren Kreis $v_1v_4 \cdots v_nv_1$. Dies ist jedoch ein Widerspruch zur Annahme, dass $v_1v_2v_3 \cdots v_nv_1$ bereits ein kürzester Kreis mit v_1 ist.

Fall 2: Es gibt die Kante (v_4, v_1).

Dann liegt v_1 auf dem gerichteten Viereck $v_1v_2v_3v_4v_1$, was im Widerspruch zur Annahme steht. □

Aus Lemma 3.3.2 folgt direkt:

Korollar 3.3.2 *Ein bipartiter Turniergraph T ist genau dann kreisfrei, wenn er keine Kreise der Länge vier enthält.* □

Die Korollare 3.3.1 und 3.3.2 ermöglichen triviale parametrisierte Algorithmen für Feedback Vertex Set und Feedback Arc Set auf (bipartiten) Turniergraphen und Approximationsalgorithmen für Feedback Vertex Set auf diesen Graphen. Zum Beispiel bekommen wir einen parametrisierten Algorithmus der Laufzeit $O(3^k \cdot n^{O(1)})$ für Feedback Vertex Set auf Turniergraphen, indem wir im eingegebenen Graphen einen Kreis der Länge drei suchen und rekursiv in alle drei Möglichkeiten verzweigen, einen Knoten dieses Kreises zu löschen. Eine triviale 3-Approximation erhalten wir, indem wir wiederholt einen Kreis der Länge drei suchen, alle drei Knoten des Kreises löschen und diese in das Feedback Vertex Set aufnehmen.

Raman und Saurabh beweisen in [RS04] eine interessante Eigenschaft von minimalen Feedback Arc Sets für gerichtete Graphen, die uns später ermöglichen wird, in Feedback-Arc-Set-Algorithmen die Turniergraphen-Eigenschaft der Eingabegraphen zu erhalten, indem wir die Kanten der Graphen umdrehen statt sie zu löschen. Wir geben einen eigenen Beweis des Lemmas an, der einfacher als der von Raman und Saurabh beschriebene ist.

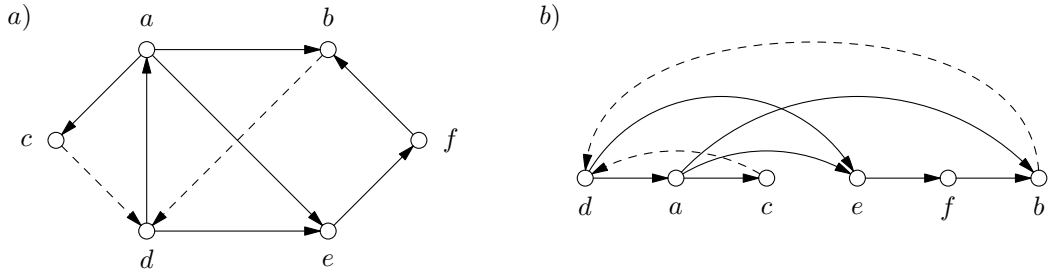

Abbildung 3.7: Illustration zu Lemma 3.3.3: a) Ein gerichteter Graph, für den die gestrichelten Kanten ein minimales Feedback Arc Set markieren. b) Derselbe Graph; die Knoten sind wie im Lemma beschrieben sortiert.

Lemma 3.3.3 [RS04] *Sei $G = (V, E)$ ein gerichteter Graph und F ein minimales Feedback Arc Set von G. Des Weiteren sei G' der Graph, der entsteht, wenn man die Kanten aus F in G umdreht. Dann ist G' kreisfrei.*

Beweis: Sei $\tilde{G} = (V, E \setminus F)$ der Graph, der entsteht, wenn man die Kanten des Feedback Arc Set F aus G entfernt.

Da $\tilde{G}$ kreisfrei ist, kann man V topologisch so sortieren, dass eine Folge $v_1, v_2, \ldots v_{|V|}$ mit $(v_i, v_j) \in (E \setminus F) \Rightarrow i < j$ entsteht.

Ordnen wir die Knoten aus V in dieser Reihenfolge von links nach rechts räumlich an und verbinden sie mit den Kanten aus $E \setminus F$ (Beispiel siehe Abbildung 3.7), so zeigen alle Kanten von links nach rechts.

Wenn wir jetzt die Kanten aus F einfügen, sind sie „Rückwärtskanten“, d. h. sie zeigen von rechts nach links: Gäbe es eine Kante $e \in F$, die keine Rückwärtskante ist, so hätten wir eine topologische Sortierung für den Graphen, der aus G entsteht, wenn die Kantenmenge $F \setminus \{e\}$ gelöscht wird, also wäre dieser Graph kreisfrei. Dann wäre $F \setminus \{e\}$ ein Feedback Arc Set für G, was im Widerspruch zur Minimalität von F steht.

Drehen wir in G jetzt alle Kanten aus F um, so erhalten wir den Graphen G'. Da wir genau die Rückwärtskanten umgedreht haben, zeigen jetzt alle Kanten von links nach rechts, d. h. die Knoten von G' lassen sich topologisch sortieren. Dann ist G' kreisfrei. □

Kapitel 4

Eine Problemkernreduktion für Feedback Arc Set auf Turniergraphen

In diesem Kapitel stellen wir eine Möglichkeit vor, für eine Feedback-Arc-Set-Instanz (T, k) auf Turniergraphen einen Problemkern zu finden.

Der Problemkern ist eine kleinere Instanz (T', k') des Problems, die genau dann lösbar ist, wenn (T, k) lösbar ist. Zudem ist die Größe von T' nur von k abhängig, und es gilt $k' \leq k$. Mit Hilfe einer Problemkernreduktion lässt sich auf einfache Weise ein parametrisierter Algorithmus für das Problem finden, indem die Probleminstanz mit der Reduktion in Polynomialzeit auf eine Größe, die nur von k abhängt, reduziert wird und im Anschluss mit einem beliebigen genauen Algorithmus gelöst wird.

Im weiteren Verlauf dieses Kapitels werden wir zunächst mit Hilfe zweier einfacher Reduktionsregeln beschreiben, wie wir eine Probleminstanz verkleinern, und im Anschluss daran beweisen, dass diese Regeln einen Problemkern der Größe $O(k^2)$ berechnen. Zum Ende des Kapitels zeigen wir dann, dass die Problemkernreduktion für einen Turniergraphen mit n Knoten und den Parameter k eine Laufzeit von $O(k \cdot n^3)$ hat.

4.1 Die Reduktionsregeln

Wir geben als Erstes die zwei Reduktionsregeln an, deren Anwendung auf eine Feedback-Arc-Set-Instanz (T, k), die aus einem Turniergraph $T = (V, E)$ und der Lösungsgröße k besteht, einen Problemkern erzeugt.

Regel 1: Wenn T eine Kante (u, v) besitzt, die zu mehr als k gerichteten Kreisen der Länge drei gehört, dann ersetze diese Kante in T

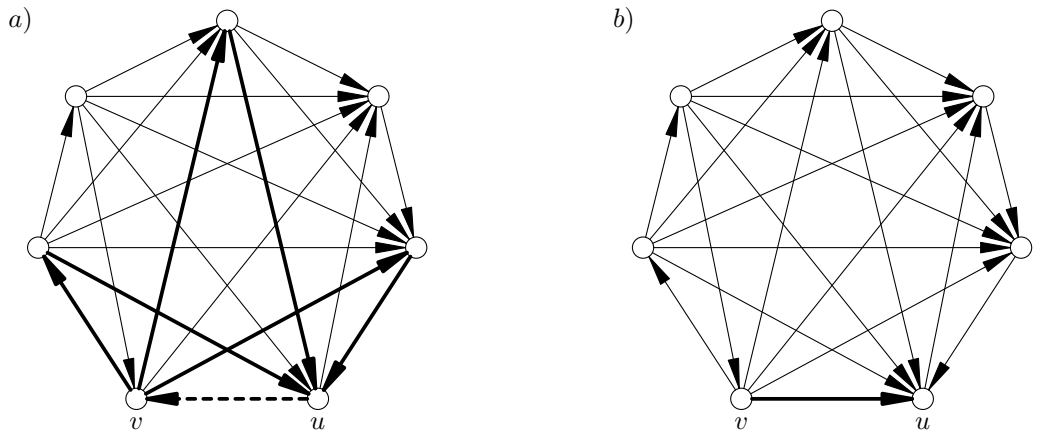

Abbildung 4.1: Beispiel für die Anwendung von Regel 1 bei $k = 2$: *a*) Vor der Reduktion: Die Kante (u, v) gehört zu drei Kreisen der Länge drei, hier mit fett gedruckten Pfeilen markiert. *b*) Nach der Reduktion: Für (u, v) wurde (v, u) in den Graphen eingefügt, und es gilt $k = 1$.

durch (v, u) und verringere k um eins.

Regel 2: Lösche alle Knoten aus V, die nicht auf Kreisen liegen.

Regel 2 führen wir durch, indem wir alle starken Zusammenhangskomponenten des Graphen bestimmen und die der Größe eins löschen.

Ein Beispiel für die Anwendung der Regel 1 ist in Abbildung 4.1, ein Beispiel für Regel 2 in Abbildung 4.2 zu sehen.

Diese Reduktionsregeln werden angewendet, bis $k = 0$ oder bis keine Regel mehr angewendet werden kann. Es genügt, Regel 2 erst dann zu benutzen, wenn Regel 1 nicht mehr anwendbar ist.

Damit ist die grundlegende Funktionsweise der Reduktion beschrieben, und wir können uns der genaueren Untersuchung zuwenden, ob und in welcher Laufzeit diese Reduktion einen Problemkern erzeugen kann.

4.2 Der Problemkern

Nachdem wir im vorigen Abschnitt den Algorithmus zur Problemkernreduktion kennen gelernt haben, zeigen wir im folgenden Satz, dass die Reduktionsregeln tatsächlich geeignet sind, in polynomieller Laufzeit einen Problemkern zu erzeugen.

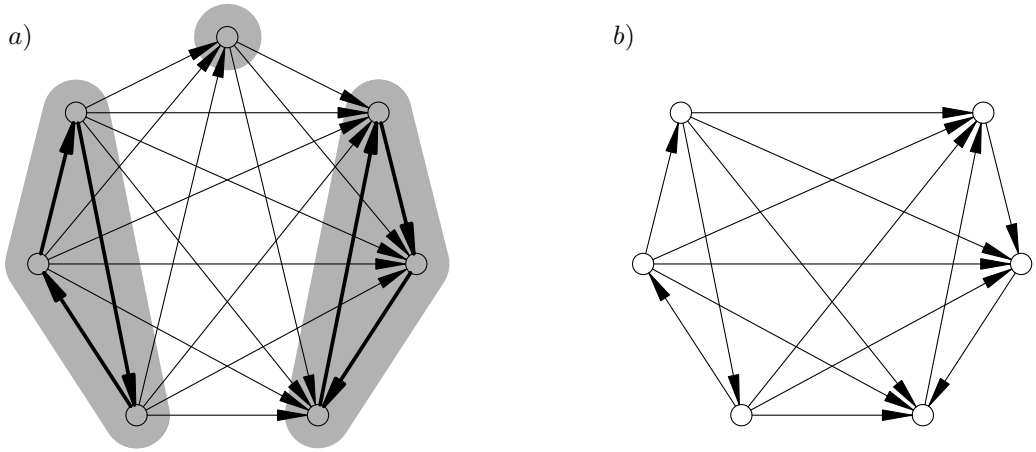

Abbildung 4.2: Beispiel für die Anwendung von Regel 2: Knoten, die sich nicht in Kreisen befinden, werden gelöscht, indem man alle starken Zusammenhangskomponenten der Größe eins löscht. *a*) Vor der Reduktion: Die starken Zusammenhangskomponenten sind grau unterlegt. *b*) Nach der Reduktion: Die Komponente der Größe eins wurde gelöscht.

Satz 4.2.1 *Für* Feedback Arc Set *auf Turniergraphen gibt es einen Problemkern mit* $O(k^2)$ *Knoten, der mit Hilfe der oben angegebenen Reduktionsregeln in* $O(k \cdot n^3)$ *Zeit zu bestimmen ist.*

Beweis: Um für eine Instanz des Feedback-Arc-Set-Problems einen Problemkern zu finden, wenden wir die obigen Reduktionsregeln auf den gegebenen Turniergraphen $T = (V, E)$ und die maximal erlaubte Lösungsgröße k an.

Sei T' der durch Anwendung der Reduktionsregeln 1 und 2 aus T entstandene Graph und k' der zugehörige Parameter.

Jetzt ist zu zeigen, dass (T', k') ein Problemkern für (T, k) ist. In Lemma 4.2.1 beweisen wir, dass (T, k) genau dann lösbar ist, wenn (T', k') lösbar ist. Danach ist noch zu zeigen, dass T' nicht mehr als $O(k^2)$ Knoten besitzt. Dass $k' \leq k$ gilt, ist offensichtlich, da keine der Reduktionsregeln k erhöht.

Um zu zeigen, dass die Knotenzahl von T' durch $O(k^2)$ begrenzt ist, genügt es zu zeigen, dass dies der Fall ist, wenn T' ein Feedback Arc Set der Größe k' besitzt. Daher beweisen wir in Lemma 4.2.2, dass der Graph T' höchstens $k'^2 + 2k'$ Knoten besitzt, wenn es ein Feedback Arc Set F der Größe k' für T' gibt. Dann wissen wir, dass für T' kein Feedback Arc Set der Größe k' existiert, es also auch T kein Feedback Arc Set der Größe k gibt, wenn T' mehr als $k^2 + 2k$ Knoten besitzt.

Abschließend betrachten wir den Zeitaufwand der Reduktionsregeln. In Lemma 4.2.3 zeigen wir, dass der Algorithmus mit einer Laufzeit von $O(k \cdot n^3)$ auskommt. □

Es sind noch drei Lemmata zu beweisen, deren Aussagen wir in Satz 4.2.1 verwendet haben. Zunächst widmen wir uns Lemma 4.2.1, das die Äquivalenz von Eingabe (T, k) und reduzierter Probleminstanz (T', k') behauptet.

Lemma 4.2.1 *Die durch Anwendung der Reduktionsregeln 1 und 2 aus einer* Feedback-Arc-Set-*Instanz abgeleitete Instanz* (T', k') *hat die Eigenschaft, dass der Graph* T' *genau dann ein Feedback Arc Set der Größe* k' *besitzt, wenn für* T *ein Feedback Arc Set der Größe* k *existiert.*

Beweis: Wir wissen aus Lemma 3.3.3, dass ein gerichteter Graph, in dem man die Kanten eines minimalen Feedback Arc Sets umdreht, kreisfrei ist. Daher bleibt die Lösbarkeit einer Feedback-Arc-Set-Instanz, bestehend aus T und k, erhalten, wenn man eine Kante, die in jedem Feedback Arc Set mit höchstens k Kanten enthalten sein muss, umdreht und k entsprechend um eins senkt.

Regel 1 entspricht genau diesem Vorgehen: Wenn eine Kante (u, v) in mehr als k gerichteten Dreiecken vorkommt, ist die einzige Möglichkeit, diese Dreiecke mit dem Umdrehen von höchstens k Kanten zu zerstören, (u, v) in das Feedback Arc Set aufzunehmen. Dadurch, dass wir solche Kanten umdrehen anstatt sie zu löschen, bleibt der eingegebene Graph ein Turniergraph. Regel 2 ist korrekt, da durch das Löschen von Knoten, die nicht auf Kreisen liegen, weder Kreise zerstört noch erzeugt werden. Der für eine lösbare Probleminstanz berechnete Problemkern ist also ebenfalls eine lösbare Instanz.

Umgekehrt konstruieren wir aus einem Feedback Arc Set der Größe k' für T' ein Feedback Arc Set der Größe k für T, indem wir die Menge derjenigen Kanten, die im Laufe einer Problemkernreduktion umgedreht werden, mit dem Feedback Arc Set für T' vereinigen. □

Eine weitere Problemkerneigenschaft, die in Satz 4.2.1 zu zeigen war, ist die Größe der reduzierten Probleminstanz. Wir zeigen in Lemma 4.2.2, dass T' höchstens $k'^2 + 2k'$ Knoten besitzt.

Lemma 4.2.2 *Wenn es ein Feedback Arc Set* F *der Größe* k' *für* T' *gibt, besitzt der Graph* T' *höchstens* $k'^2 + 2k'$ *Knoten.*

Beweis: Sei F ein Feedback Arc Set der Größe k' für T'. Für jede der bis zu k' Kanten (u, v) aus F gilt: (u, v) gehört zu maximal k' gerichteten Dreiecken, die insgesamt bis zu $k' + 2$ Knoten besitzen, nämlich u, v und k' „dritte“ Knoten. Da F mindestens eine Kante jedes gerichteten Kreises enthalten

muss, gibt es insgesamt nicht mehr als diese $k'(k'+2)$ Knoten in Kreisen der Länge drei.

Andererseits gibt es in T' auch keinen Knoten, der nicht zu einem gerichteten Dreieck gehört – Knoten, die überhaupt nicht auf Kreisen liegen, wurden mit Hilfe von Regel 2 gelöscht, und jeder Knoten, der auf einem Kreis liegt, ist nach Lemma 3.3.1 auch Teil eines gerichteten Dreiecks.

Somit besitzt der reduzierte Graph T' höchstens $k'^2 + 2k'$ Knoten, falls T' ein Feedback Arc Set der Größe k besitzt. □

Das dritte und letzte der in Satz 4.2.1 verwendeten Lemmata gibt die Laufzeit der Reduktion an. Mit dem Beweis seiner Aussage beschließen wir dieses Kapitel.

Lemma 4.2.3 *Die Problemkernreduktion ist in einer Laufzeit von $O(k \cdot n^3)$ durchführbar.*

Beweis: Betrachten wir zunächst die Laufzeit von Reduktionsregel 1. Um eine Kante zu finden, die in mehr als k Kreisen der Länge drei vorkommt, benötigt man $O(n^3)$ Zeit. Das Umdrehen der entsprechenden Kante und das Senken von k sind in konstanter Zeit möglich. Regel 1 hat damit eine Laufzeit von $O(n^3)$.

In Regel 2 bestimmen wir alle Knoten, die nicht auf gerichteten Kreisen liegen. Dafür zerlegen wir T in seine starken Zusammenhangskomponenten und löschen diejenigen der Größe eins aus dem Graphen. Dass unser Vorgehen korrekt ist, auf diese Weise also genau diejenigen Knoten, die nicht auf Kreisen liegen, gelöscht werden, ist einfach zu sehen: Wenn ein Knoten der einzige Knoten in einer starken Zusammenhangskomponente ist, liegt er auf keinem Kreis, weil alle Knoten eines Kreises zu derselben starken Zusammenhangskomponente gehören. Wenn andererseits eine starke Zusammenhangskomponente mehrere Knoten enthält, so liegen diese alle auf Kreisen: Wenn ein Knoten u mit einem anderen Knoten v in derselben starken Zusammenhangskomponente liegt, gibt es einen Pfad von u nach v und einen Pfad von v nach u. Sind diese Pfade knotendisjunkt, so bilden sie einen Kreis durch u und v. Sind sie nicht knotendisjunkt, so liegt u auf einem Kreis durch den ersten gemeinsamen Knoten der beiden Pfade.

Die maximalen starken Zusammenhangskomponenten von T zu finden, benötigt eine Laufzeit von $O(|V| + |E|) = O(n^2)$ [Tar72]. Weil man die zu löschenden Knoten jeweils in konstanter Zeit, insgesamt somit in $O(n)$ Zeit, aus dem Graphen entfernen kann, ergibt sich damit für Regel 2 eine Laufzeit von $O(n^2)$.

Da Regel 1 höchstens k-mal anwendet wird (bis $k = 0$) und es ausreicht, Regel 2 nur einmal – sobald Regel 1 nicht mehr anwendbar ist – auszuführen,

erhalten wir eine Gesamtlaufzeit von $O(k \cdot n^3)$. □

Kapitel 5

Parametrisierte Algorithmen

Wir haben bereits in Kapitel 1 das Konzept der parametrisierten Algorithmen beschrieben. Hier stellen wir nun zwei neue parametrisierte Algorithmen vor. In Kapitel 5.1 zeigen wir zunächst einen Algorithmus mit einer Laufzeit von $O(2^k \cdot n^2(\log n + k))$ für Feedback Vertex Set auf Turniergraphen, in Kapitel 5.2 einen Algorithmus für Feedback Arc Set auf bipartiten Turniergraphen, der eine Laufzeit von $O(3{,}38^k \cdot n^6)$ benötigt.

5.1 Iterative Kompression für Feedback Vertex Set auf Turniergraphen

Im diesem Abschnitt präsentieren wir unseren parametrisierten Algorithmus FVST für Feedback Vertex Set auf Turniergraphen, d. h. gerichteten Graphen, in denen es zwischen je zwei Knoten u, v stets entweder die Kante (u, v) oder die Kante (v, u) gibt. Als Parameter verwenden wir die Größe k des gesuchten Feedback Vertex Sets.

Die Laufzeit von FVST beträgt insgesamt $O(2^k \cdot n^2(\log n + k))$. Damit verbessern wir die exponentielle Basis der Laufzeit gegenüber dem bisher schnellsten parametrisierten Algorithmus von 2,18 auf 2.

Der zuvor beste bekannte parametrisierte Algorithmus für Feedback Vertex Set auf Turniergraphen ist ein Algorithmus für 3-Hitting Set von Fernau [Fer05]. Fernaus Algorithmus läuft in einer Zeit von $O(2{,}18^k + n)$ und ist auf Feedback Vertex Set mittels einer einfachen Reduktion anwendbar:

Für eine Instanz von Feedback Vertex Set, die aus einem Turniergraphen $T = (V, E)$ und der Lösungsgröße $k \geq 0$ besteht, ist (C, V, k) eine

äquivalente 3-Hitting-Set-Instanz, wobei

$$C = \{\{v_1, v_2, v_3\} \mid v_1, v_2, v_3 \in V \text{ und } v_1v_2v_3v_1 \text{ ist ein Dreieck in } T\}.$$

Unser Algorithmus FVST basiert auf der Methode der *iterativen Kompression*, die erstmals von Reed et al. [RSV04] verwendet wurde. Bei der iterativen Kompression verwenden wir einen *Kompressionsalgorithmus*, der für einen gegebenen Graphen und eine bekannte Lösung F der Größe $k+1$ eine um eins kleinere Lösung berechnet, falls eine solche existiert. Um eine Lösung für einen Graphen $T = (V, E)$ zu berechnen, wenden wir diesen Algorithmus, beginnend bei einem Knoten, schrittweise auf immer größere induzierte Teilgraphen von T an.

Das grundlegende Prinzip, das wir hier für die Kompression nutzen, ist dem der parametrisierten Algorithmen für Feedback Vertex Set auf ungerichteten Graphen von Guo et al. [GGH+05] und Dehne et al. [DFL+05] ähnlich. Es besteht darin, alle Möglichkeiten zu betrachten, die bekannte Lösung F in zwei Mengen S und X zu zerlegen. Dabei ist X jeweils die Menge der Knoten aus F, die wir in das kleinere Feedback Vertex Set übernehmen möchten, und S die Menge der Knoten aus F, die wir nicht in die kleinere Lösung übernehmen. Wir überprüfen für jede dieser Zerlegungen, ob es in dem Graphen T', der entsteht, wenn die Knotenmenge X aus T gelöscht wird, ein Feedback Vertex Set mit weniger als $|S|$ Knoten gibt, das keine Knoten aus S enthält. Zusammen mit X ergeben diese Knoten dann ein Feedback Vertex Set mit maximaler Größe k für den Graphen T. Gelingt es mit dieser Methode für keine Zerlegung, ein Feedback Vertex Set mit höchstens k Knoten für T zu finden, so gibt es keines.

Um für eine gegebene Zerlegung ein Feedback Vertex Set mit weniger als $|S|$ Knoten ohne Knoten aus S zu finden, führen wir zunächst einfache Datenreduktionen durch und zeigen dann, dass sich das Problem auf dem verbleibenden Graphen als eine effizient durchzuführende Berechnung der längsten gemeinsamen Teilsequenz zweier Folgen darstellt. Auf diese Art und Weise gelingt es uns, jede Zerlegung in Polynomialzeit zu überprüfen.

Eine genaue Beschreibung unseres Kompressionsalgorithmus ist in Abbildung 5.1 zu sehen.

Als nächstes möchten wir Korrektheit und Laufzeit des Kompressionsalgorithmus untersuchen. Zuerst zeigen wir in Lemma 5.1.1 seine Korrektheit, anschließend analysieren wir in Lemma 5.1.5 seine Laufzeit.

5.1.1 Die Korrektheit des Kompressionsalgorithmus

Für den Korrektheitsbeweis vollziehen wir die Arbeitsweise und den Zweck jedes einzelnen Schrittes des Algorithmus nach.

compress(T,F)
Eingabe: Ein Turniergraph $T = (V, E)$ und ein Feedback Vertex Set F der Größe $k + 1$ für T.
Ausgabe: „JA“ und ein Feedback Vertex Set für T mit maximal k Knoten, falls ein solches existiert;
„NEIN“, wenn es kein Feedback Vertex Set mit höchstens k Knoten für T gibt.

1 Bilde alle möglichen Zerlegungen von F in zwei Teilmengen X und S.
Für jede dieser Zerlegungen führe die folgenden Schritte aus:

2 Induziert S Kreise in T?
Nein: Gehe zu Schritt 3.
Ja: Verwirf die aktuelle Zerlegung von F in X und S und überprüfe die nächste.
Falls bereits alle Zerlegungen überprüft sind, gib „NEIN“ zurück und brich den Algorithmus ab.

3 Sei $V' := V \setminus X$ und $F' := X$.
Bilde den induzierten Teilgraphen $T' := T[V']$ von T.

4 Sortiere die Knoten aus $T[S]$ *topologisch* in eine Reihenfolge $\tilde{S} = s_1, s_2 \dots s_{|S|}$.

5 Bestimme alle gerichteten Dreiecke in T' mit zwei Knoten aus S und einem aus $V' \setminus S$.
Lösche jeweils den Knoten aus $V' \setminus S$ aus T' und füge ihn in F' ein.

6 Bestimme für jeden Knoten $v \in V' \setminus S$ den Wert

$$p(v) := \begin{cases} \min\{i \mid (v, s_i) \in E[V']\}, & \exists i \in \{1, \dots, |S|\} : (v, s_i) \in E[V'] \\ |S| + 1, & \text{sonst} \end{cases}$$

7 Sortiere $V' \setminus S$ topologisch in eine Folge $\tilde{V} = v_1, v_2, \dots, v_{|V' \setminus S|}$.

8 Sortiere $V' \setminus S$ aufsteigend nach $p(v)$ in eine Folge P. Ordne dabei Knoten mit gleichem $p(v)$ gemäß ihrer Reihenfolge in $\tilde{V}$ an.

9 Bestimme die längste gemeinsame Teilsequenz von $\tilde{V}$ und P.
Sei Y die Menge der Knoten dieser Sequenz.
$F' := F' \cup (V' \setminus (Y \cup S))$

10 Gilt $|F'| \leq k$?
Ja: Gib „JA“ und F' zurück und brich ab.
Nein: Verwirf die aktuelle Zerlegung von F in S und X und überprüfe die nächste.
Falls bereits alle Zerlegungen überprüft sind, gib „NEIN“ zurück und brich den Algorithmus ab.

Abbildung 5.1: Kompressionsalgorithmus für Feedback Vertex Set auf Turniergraphen

Lemma 5.1.1 *Bei Eingabe eines Turniergraphen T und eines Feedback Vertex Sets der Größe $k+1$ für T gibt der in Abbildung 5.1 beschriebene Algorithmus* compress *„JA“ und ein Feedback Vertex Set der Größe k für T aus, falls ein solches existiert. Ansonsten gibt der Algorithmus „NEIN“ aus.*

Beweis: Bei einem Aufruf von compress(T, F) werden in Schritt 1 zunächst alle möglichen Zerlegungen des bestehenden Feedback Vertex Sets F in zwei Teilmengen S und X gebildet. Im weiteren Verlauf des Algorithmus wird für jede dieser Zerlegungen überprüft, ob es ein Feedback Vertex Set mit höchstens $k = |F| - 1$ Knoten für T gibt, das alle Knoten aus X und keine Knoten aus S enthält. Wenn es ein Feedback Vertex Set $\hat{F}$ mit höchstens k Knoten für T gibt, wird der Algorithmus mindestens für die Zerlegung von F in $S = F \setminus \hat{F}$ und $X = F \cap \hat{F}$ erfolgreich sein und in Schritt 10 ein Feedback Vertex Set dieser Größe ausgeben. Ansonsten wird er für keine Zerlegung erfolgreich sein und mit Antwort „NEIN“ abbrechen.

Um zu zeigen, dass genau dann für eine Zerlegung ein Feedback Vertex Set mit maximal k Knoten ausgegeben wird, wenn es ein solches Feedback Vertex Set gibt, betrachten wir die Arbeitsweise der einzelnen Schritte des Algorithmus. Sei also $F = S \cup X$ eine Zerlegung von F.

Schritt 2: In Schritt 2 wird überprüft, ob S in T gerichtete Kreise induziert. Wenn ja, kann es kein Feedback Vertex Set für T ohne Knoten aus S geben; die Überprüfung der Zerlegung kann daher abgebrochen werden.

Schritt 3: Anschließend wird in Schritt 3 der Graph T' gebildet, der entsteht, wenn man die Knoten aus X, also diejenigen Knoten, die mit Sicherheit zu einem neuen Feedback Vertex Set gehören sollen, löscht. Gleichzeitig fügen wir diese Knoten in eine neu definierte Menge F' ein, die nach und nach zu einem Feedback Vertex Set für T aufgebaut wird, indem die Knoten eines Feedback Vertex Sets für T' in F' eingefügt werden.

Schritt 4 betrachten wir später zusammen mit Schritt 7.

Schritt 5: In Schritt 5 werden alle Kreise der Länge drei in T' gefunden, die einen Knoten aus $V' \setminus S$ und zwei aus S enthalten. Da wir für das Feedback Vertex Set für T keine Knoten aus S verwenden, muss für jeden solchen Kreis der jeweilige Knoten aus $V' \setminus S$ Teil dieses Feedback Vertex Sets sein. Daher können wir die entsprechenden Knoten aus T' löschen und in F' einfügen.

Schritt 6: Nachdem wir Schritt 5 ausgeführt und für jeden Knoten $v \in V' \setminus S$ den Wert $p(v)$ berechnet haben, gilt für alle $i \in \{1, \ldots |S|\}$:

$$(s_i, v) \in E[V'] \text{ wenn } i < p(v) \text{ und } (v, s_i) \in E[V'] \text{ wenn } i \geq p(v)$$

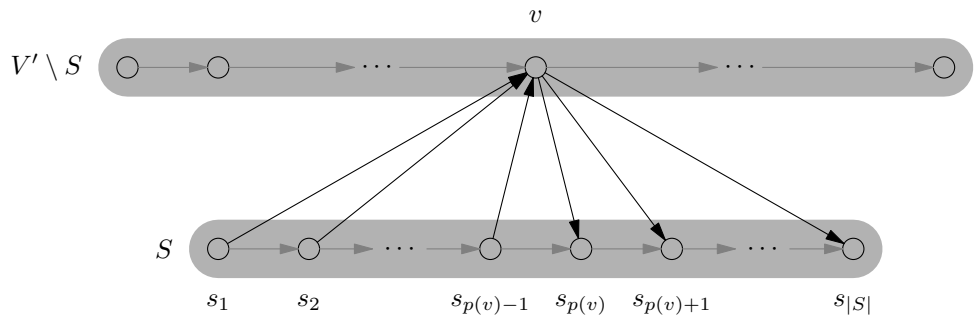

Abbildung 5.2: Darstellung der Eigenschaften von $p(v)$. Zur Vereinfachung sind nur einige der Kanten zwischen Knoten in S bzw. $V \setminus S$ (grau) eingezeichnet.

Eine Illustration dieser Eigenschaft findet sich in Abbildung 5.2. Wir beweisen sie später in Lemma 5.1.2.

Schritte 4 und 7: In den Schritten 4 und 7 werden die Knoten aus S und $V' \setminus S$ jeweils topologisch sortiert, also in eine Reihenfolge gebracht, in der es in $E[V']$ von jedem Knoten Kanten zu allen Knoten gibt, die in der Reihenfolge hinter ihm stehen. Eine topologische Sortierung der Knoten eines gerichteten Graphen ist genau dann möglich, wenn der Graph kreisfrei ist. Dass $T'[S]$ kreisfrei ist, wurde in Schritt 2 sichergestellt, und $T'[V' \setminus S]$ ist kreisfrei, weil S ein Feedback Vertex Set für T' ist.

In Turniergraphen ist die topologische Sortierung eindeutig, da es zwischen zwei Knoten immer eine Kante geben muss und beliebige Knoten somit vergleichbar sind: Existiert in der Kantenmenge eine Kante (u, v), so steht u in der topologisch sortierten Folge vor v.

Schritt 8: Die Folge P entsteht durch das Sortieren der Knoten aus $V' \setminus S$ nach ihrem jeweiligen Wert p. Die Reihenfolge wird dadurch eindeutig, dass Knoten mit demselben Wert p gemäß ihrer Reihenfolge in $\tilde{V}$ sortiert werden.

Schritt 9: In diesem Schritt berechnen wir die Menge Y der Knoten der längsten gemeinsamen Teilsequenz von V' und P. Wie wir in Lemma 5.1.3 beweisen werden, ist $T'[Y \cup S]$ der größte kreisfreie induzierte Teilgraph von T', der alle Knoten aus S enthält. Ein Beispiel dafür ist in Abbildung 5.3 zu sehen.

Dann sind diejenigen Knoten, die aus T' entfernt werden müssen, um den kreisfreien Graphen $T'[Y \cup S]$ zu erhalten, das kleinste Feedback

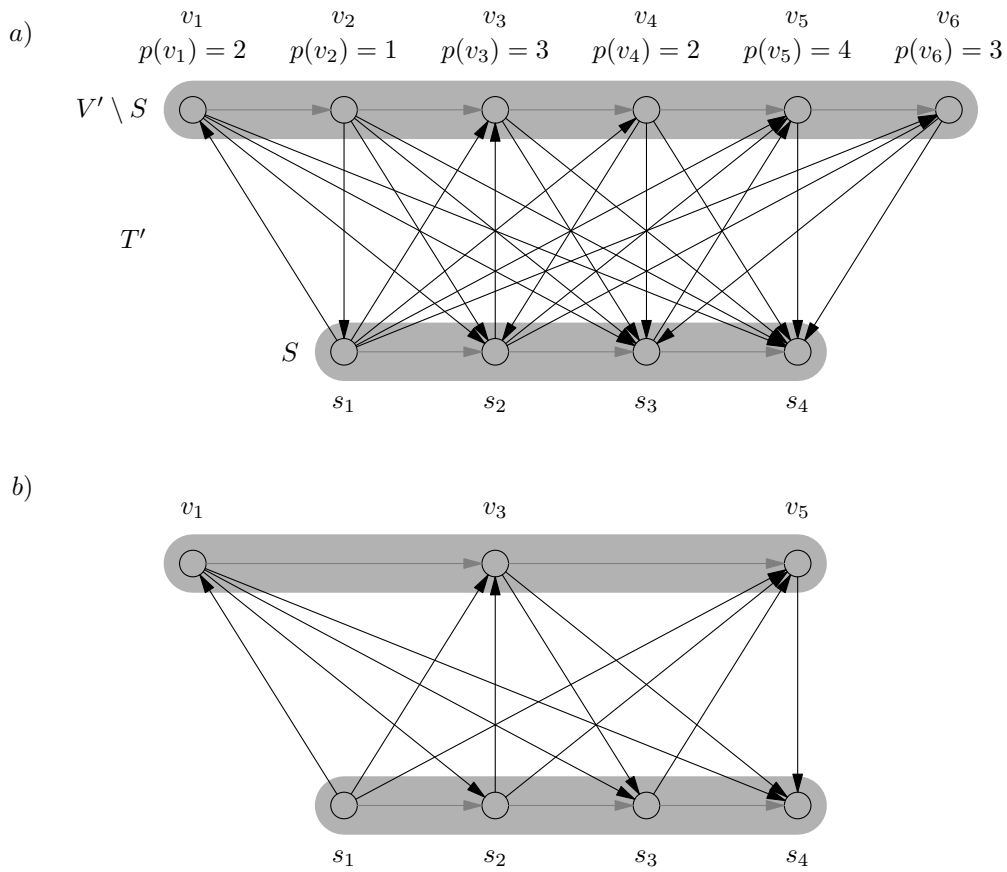

Abbildung 5.3: Ein Beispiel für die Ausführung des 9. Schrittes des compress-Algorithmus. Aus Gründen der Übersichtlichkeit sind nur einige der vielen Kanten, die alle Knoten der Mengen S und $V' \setminus S$ mit allen jeweils rechts von ihnen gelegenen Knoten derselben Menge verbinden, grau angedeutet. a) Der Turniergraph T'. Es gilt: $\tilde{V} = v_1, v_2, v_3, v_4, v_5, v_6$ und $P = v_2, v_1, v_4, v_3, v_6, v_5$. Die längste gemeinsame Teilsequenz v_1, v_3, v_5 hat die Länge 3. Damit gilt $Y = \{v_1, v_3, v_5\}$. b) Der kreisfreie Teilgraph $T'[Y \cup S]$.

Vertex Set für T' ohne Knoten aus S. Dies ist die Menge $V' \setminus (Y \cup S)$, deren Elemente wir in F' einfügen.

Schritt 10: Da wir in Schritt 2 die Knoten aus X in F' eingefügt und danach nur noch den entsprechend reduzierten Graphen T' betrachtet haben, im Schritt 5 eine korrekte Datenreduktion durchgeführt und in Schritt 9 ein kleinstes Feedback Vertex Set ohne Knoten aus S für den Restgraphen T' in F' eingefügt haben, ist F' nach Abschluss von Schritt 9 ein kleinstes Feedback Vertex Set für T ohne Knoten aus S und mit allen Knoten aus X. In Schritt 10 überprüfen wir daher, ob dieses Feedback Vertex Set höchstens k Knoten enthält. Da es kein kleineres Feedback Vertex Set als F' gibt, das alle Knoten aus X und keinen aus S enthält, müssen wir die Zerlegung $F = S \cup X$ verwerfen, falls $|F'| > k$. □

Jetzt sind noch zwei Lemmata zu beweisen, deren Aussagen wir im obigen Beweis benutzt haben. Wir beginnen mit Lemma 5.1.2, das den in Schritt 6 definierten Wert $p(v)$ maßgeblich charakterisiert.

Lemma 5.1.2 *Nach Durchführung von Schritt 6 des Algorithmus gilt für jeden Knoten $v \in V' \setminus S$ und jedes $i \in \{1, \ldots, |S|\}$*

$$(v, s_i) \in E[V'] \text{ gdw. } i \geq p(v)$$

wobei s_i der i-te Knoten der in Schritt 4 definierten, topologisch sortierten Knotenfolge $\tilde{S}$ ist (siehe Abbildung 5.2).

Beweis: Die Definition von $p(v)$ lautet:

$$p(v) = \begin{cases} \min\{i \mid (v, s_i) \in E[V']\} & \text{falls } \exists i \in \{1, \ldots, |S|\} : (v, s_i) \in E[V'] \\ |S| + 1 & \text{sonst} \end{cases}$$

Zuerst zeigen wir, dass $(v, s_i) \in E[V']$, wenn $i \geq p(v)$.

Angenommen, es gäbe einen Knoten $v \in V' \setminus S$ und ein $i \geq p(v)$ sodass es die Kante (v, s_i) nicht gibt. Dann muss es stattdessen die Kante (s_i, v) geben. Da nach Definition von $p(v)$ die Kante $(v, s_{p(v)})$ und nach Definition von $\tilde{S}$ die Kante $(s_{p(v)}, s_i)$ existiert, wäre $v s_{p(v)} s_i v$ dann ein gerichtetes Dreieck mit genau einem Knoten aus $V \setminus S$. Solche Kreise gibt es jedoch nach Ausführung von Schritt 5 nicht mehr.

Jetzt ist noch die umgekehrte Richtung zu beweisen. Wir zeigen, dass für alle $i < p(v)$ gilt: $(v, s_i) \notin E[V']$. Diese Aussage folgt direkt aus der Definition von $p(v)$, da $p(v)$ die kleinste Zahl i ist, für die $(v, s_i) \in E[V']$. □

Als nächstes beweisen wir das zweite im Beweis von Lemma 5.1.1 verwendete Lemma. Seine Aussage ist für den Korrektheitsbeweis zentral.

Lemma 5.1.3 *Nach dem Ausführen von Schritt 9 des Algorithmus* compress *ist $T'[Y \cup S]$ der größte kreisfreie induzierte Teilgraph von T', der alle Knoten aus S enthält.*

Beweis: In Lemma 5.1.4 beweisen wir, dass nach der Ausführung von Schritt 5 genau dann Kreise in T' existieren, wenn es zwei Knoten u, v aus $V' \setminus S$ mit $(u, v) \in E[V']$ und $p(u) > p(v)$ gibt. Dies bedeutet, dass T' genau dann kreisfrei ist, wenn für alle Knotenpaare $u, v \in V' \setminus S$ mit $(u, v) \in E[V']$ jeweils $p(u) \leq p(v)$ gilt.

Eine direkte Folgerung hieraus ist die folgende wichtige Beobachtung: T' ist genau dann azyklisch, wenn die in den Schritten 7 und 8 definierten Folgen $\tilde{V}$ und P identisch sind, denn genau in diesem Fall gilt für jeweils zwei Knoten u, v, die mit einer Kante (u, v) verbunden sind, $p(u) \leq p(v)$. Dies gilt mit derselben Begründung auch für alle induzierten Teilgraphen von T', denn beim Entfernen eines Knotens aus T' ändern sich die zugehörigen Folgen $\tilde{V}$ und P nur insofern, als dass der entsprechende Knoten nicht mehr vorkommt, die Reihenfolge der restlichen Knoten bleibt jedoch gleich.

Da die Gleichheit der Folgen notwendige und hinreichende Bedingung für die Kreisfreiheit eines Teilgraphen von T' mit allen Knoten aus S ist, können wir den größten solchen Teilgraphen bestimmen, indem wir die größte Menge Y von Knoten aus $V' \setminus S$ bestimmen, für die die topologisch sortierte Folge und die nach p sortierte Folge gleich sind.

Die Menge Y ist also die längste gemeinsame Teilsequenz von P und $\tilde{V}$. Zusammen mit den Knoten aus S induziert Y dann den größten induzierten Teilgraphen von T', in dem alle Knoten aus S erhalten bleiben. □

Um den Beweis von Lemma 5.1.3 zu vervollständigen, benötigen wir jetzt noch Lemma 5.1.4, dessen Aussage wir vorhin verwendet haben.

Lemma 5.1.4 *Nach Durchführung von Schritt 5 des Kompressionsalgorithmus existieren genau dann Kreise in T', wenn es zwei Knoten u, v aus $V' \setminus S$ gibt, für die gilt: $(u, v) \in E[V']$ und $p(u) > p(v)$.*

Beweis: Wenn in T' Kreise existieren, dann existieren nach Korollar 3.3.1 auch solche der Länge drei. In Schritt 5 des Kompressionsalgorithmus haben wir alle Kreise der Länge drei mit zwei Knoten aus S zerstört, ohne die Turniergrapheneigenschaft zu zerstören. Des Weiteren sind $T[S]$ und $T[V' \setminus S]$ kreisfrei. Also müssen alle Dreiecke in T' jeweils zwei Knoten aus $V' \setminus S$ und einen Knoten aus S besitzen.

Seien u und v die beiden Knoten aus $V' \setminus S$ eines Dreiecks in T', und o. B. d. A. sei (u, v) in $E[V']$. Weiter sei $s_i \in S$ der dritte Knoten des gerichteten Dreiecks, wobei i die Position des Knotens in der Folge $\tilde{S}$ ist. Dann

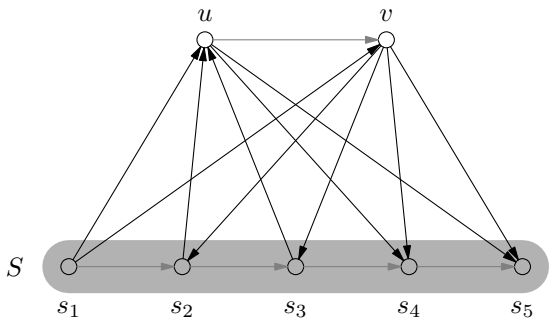

Abbildung 5.4: Ein Beispiel für Lemma 5.1.4: Da $p(u) = 4$ und $p(v) = 2$, gibt es in T' die gerichteten Dreiecke $T'[\{u, v, s_2\}]$ und $T'[\{u, v, s_3\}]$. (Aus Gründen der Übersichtlichkeit sind nur einige der Kanten (grau) dargestellt, die jeden Knoten aus S mit allen anderen Knoten aus S, die rechts von ihm liegen, verbinden.)

existieren in $E[V']$ die Kanten (v, s_i) und (s_i, u). Daraus folgt nach Lemma 5.1.2, dass $p(v) \leq i$ und $p(u) > i$, also $p(u) > p(v)$.

Umgekehrt seien nun u und v zwei Knoten aus $V' \setminus S$, die durch eine Kante $(u, v) \in E[V']$ verbunden sind, und für die $p(u) > p(v)$ gilt (Beispiel siehe Abbildung 5.4). Dann existieren nach Lemma 5.1.2 die Kanten $(v, s_{p(v)})$ und $(s_{p(v)}, u)$ in $E[V']$. Damit ist $uvs_{p(v)}u$ ein gerichtetes Dreieck in T. □

Mit Lemma 5.1.4 ist der Korrektheitsbeweis für den Kompressionsschritt abgeschlossen. Jetzt untersuchen wir wie angekündigt die Laufzeit des Kompressionsschrittes.

5.1.2 Die Laufzeit des Kompressionsalgorithmus

Im folgenden Lemma 5.1.5 gehen wir die Schritte des compress-Algorithmus einzeln durch und analysieren die jeweilige Laufzeit.

Lemma 5.1.5 *Der in Abbildung 5.1 beschriebene Kompressionsalgorithmus* compress *läuft in einer Zeit von* $O(2^k \cdot n(\log n + k))$.

Beweis: In Schritt 1 des Algorithmus werden alle möglichen Zerlegungen von F in zwei Teilmengen gebildet. Davon gibt es 2^{k+1}, also $O(2^k)$ Stück. Für jede dieser Zerlegungen werden dann die weiteren Schritte des Algorithmus durchgeführt, die jeweils eine maximale Laufzeit von $O(n \cdot k)$ bzw. $O(n \cdot \log n)$ haben, sodass sich für den Kompressionsalgorithmus eine Gesamtlaufzeit

von $O(2^k \cdot n(\log n + k))$ ergibt. Im Folgenden betrachten wir die Laufzeiten der einzelnen Schritte des Algorithmus.

Schritt 2: In diesem Schritt ist zu überprüfen, ob $T[S]$ kreisfrei ist. Dies können wir in Linearzeit tun, indem wir kontrollieren, ob die Eingangsknotengrade der Knoten in $T[S]$ genau $0, 1, \ldots, |S| - 1$ betragen (siehe Anmerkung zu Definition 2.2.10 auf Seite 18).

Schritt 4: Die topologische Sortierung von S in Schritt 4 können wir ebenfalls in Linearzeit ausführen, da wir wissen, dass ein Knoten mit Eingangsknotengrad i in der topologisch sortierten Folge an $(i + 1)$-ter Stelle steht. Denn eine topologische Sortierung der Knoten von $T[S]$ besitzt die Eingangsknotengrade $0, 1, \ldots, |S| - 1$ in genau dieser Reihenfolge, weil jeder Knoten mit allen weiteren Knoten, die in der Reihenfolge hinter ihm stehen, durch eine gerichtete Kante verbunden ist.

Schritt 5: In Schritt 5 suchen wir gerichtete Kreise der Länge drei in T', die aus zwei Knoten aus S und einem aus $V' \setminus S$ bestehen. Dies ist in $O(n \cdot k)$ Zeit möglich: Wir gehen für jeden Knoten $v \in V' \setminus S$ einmal die topologisch sortierte Knotenfolge $\tilde{S}$ durch. Wenn in dieser Folge ein Knoten $s_i \in N^+(v)$ vor einem Knoten $s_{i+1} \in N^-(v)$ steht, existiert ein gerichtetes Dreieck $vs_is_{i+1}v$.

Schritt 6: Um in Schritt 6 für einen Knoten aus $V' \setminus S$ den Wert $p(v)$ zu bestimmen, suchen wir in der topologisch sortierten Folge $\tilde{S}$ nach dem ersten Knoten s_i mit $(v, s_i) \in E[V']$. Dafür benötigen wir pro Knoten mit binärer Suche $O(\log k)$ Zeit, für alle Knoten aus $V' \setminus S$ dementsprechend $O(n \cdot \log k)$ Zeit.

Schritte 7 und 8: In den Schritten 7 und 8 wird $V' \setminus S$ auf zweierlei Arten sortiert. Für die topologische Sortierung der Knoten ist analog zu Schritt 4 eine Laufzeit von $O(n)$ nötig, für die Sortierung nach p mit einem Standardsortieralgorithmus wie *merge sort* [CLRS01] eine Laufzeit von $O(n \cdot \log n)$.

Schritt 9: In diesem Schritt wird die längste gemeinsame Teilsequenz von $\tilde{V}$ und P bestimmt. Da $\tilde{V}$ eine Permutation von P ist, ist die längste gemeinsame Teilsequenz identisch mit der längsten Teilsequenz von $\tilde{V}$, die aufsteigend nach p sortiert ist. Diese Sequenz kann mit dem Algorithmus zur Bestimmung längster aufsteigender Teilsequenzen von Fredman [Fre75] in einer Laufzeit von $O(n \cdot \log n)$ bestimmt werden. □

FVST(T)
Eingabe: Ein Turniergraph $T = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$
Ausgabe: Ein kleinstes Feedback Vertex Set F für T
1 $T_1 := T[\{v_1\}]$
2 $F_1 := \emptyset$
3 $i := 1$
4 **Wiederhole**
5 $i := i + 1$
6 $T_i := T[\{v_1, \ldots, v_i\}]$
7 $F_i := F_{i-1} \cup \{v_i\}$
8 **Wenn** compress(T_i, F_i) „JA“ und
ein Feedback Vertex Set F zurück liefert,
9 **dann** $F_i := F$
10 **bis** $i = n$
11 Gib F_n aus.

Abbildung 5.5: Algorithmus FVST

Mit Lemma 5.1.5 ist unsere Analyse des Kompressionsschrittes vollständig. Im letzten Teil dieses Abschnitts wenden wir uns nunmehr dem Gesamtalgorithmus FVST für Feedback Vertex Set auf Turniergraphen zu, dessen Herz der Kompressionsalgorithmus ist.

5.1.3 Der Algorithmus FVST

Der Algorithmus FVST, den wir hier zeigen, berechnet für einen gegebenen Turniergraphen ein kleinstes Feedback Vertex Set. Wir bilden wie in den Algorithmen für Feedback Vertex Set auf ungerichteten Graphen [GGH$^+$05, DFL$^+$05] iterativ immer größere Teilgraphen des eingegebenen Graphen und versuchen in jedem Schritt, das Feedback Vertex Set mit Hilfe des Kompressionsalgorithmus zu verkleinern.

Der genaue Algorithmus ist in Abbildung 5.5 zu sehen. Er liefert uns Satz 5.1.1, der die Hauptaussage dieses Abschnitts enthält.

Satz 5.1.1 *Für einen Turniergraphen T lässt sich ein kleinstes Feedback Vertex Set F in einer Laufzeit von $O(2^k \cdot n^2(\log n + k))$ bestimmen, wobei k die Größe von F ist.*

Beweis: Im ersten Schritt (Zeilen 1–2 in Abbildung 5.5) betrachten wir den induzierten Graphen $T_1 = T[\{v_1\}]$. Da dieser nur aus einem einzelnen Knoten

besteht, ist das optimale Feedback Vertex Set F_1 für T_1 leer.

Im $(i+1)$-ten Schritt (in der Programmschleife in den Zeilen 4–10) betrachten wir $T_{i+1} = T[\{v_1, \dots v_{i+1}\}]$. Wir kennen bereits ein kleinstes Feedback Vertex Set F_i für T_i. Dann ist $F_{i+1} := F_i \cup \{v_{i+1}\}$ ein Feedback Vertex Set für T_{i+1}. Mit Hilfe des Kompressionsalgorithmus compress können wir dann in $O(2^k \cdot n(\log n + k))$ Zeit berechnen, ob es ein um ein Element kleineres Feedback Vertex Set für T_{i+1} gibt, oder ob F_{i+1} bereits optimal ist. Auf diese Weise erhalten wir im n-ten Schritt ein kleinstes Feedback Vertex Set $F = F_n$ für $T_n = T$ bei einer Gesamtlaufzeit von $O(2^k \cdot n^2(\log n + k))$. □

Der Algorithmus FVST berechnet immer das kleinste Feedback Vertex Set für einen gegebenen Graphen. Durch eine kleine Abwandlung des Algorithmus können wir für einen gegebenen Turniergraphen T und ein $k \geq 0$ in einer Laufzeit von $O(2^k \cdot n^2(\log n + k))$ stattdessen entscheiden, ob es ein Feedback Vertex Set der Größe k für T gibt.

Dazu brechen wir die Programmschleife ab, wenn das optimale Feedback Vertex Set F_i für einen Teilgraphen T_i von T größer als k wird und geben aus, dass es kein Feedback Vertex Set der Größe k für T gibt.

5.2 Ein Suchbaumalgorithmus für Feedback Arc Set auf bipartiten Turniergraphen

Wir stellen in diesem Abschnitt unseren parametrisierten Algorithmus für Feedback Arc Set auf bipartiten Turniergraphen vor. Als Parameter verwenden wir die Größe des gesuchten Feedback Arc Sets. Der Algorithmus nutzt die Tatsache, dass Feedback Arc Set auf einem bipartiten Turniergraphen, der keinen der in Abbildung 5.6 abgebildeten Graphen als Teilgraphen enthält, in Polynomialzeit lösbar ist. Diese Tatsache beweisen wir später, indem wir zeigen, dass ein bipartiter Turniergraph, in dem sich zwei Kreise überschneiden, stets einen der beiden Graphen F_1 oder F_2 als Teilgraphen besitzt.

Unser Algorithmus funktioniert ähnlich wie der Algorithmus für Feedback Arc Set auf Turniergraphen von Raman und Saurabh [RS04]. Er kann mit geringfügigen Änderungen auch zur Lösung des kantengewichteten Feedback-Arc-Set-Problems auf bipartiten Turniergraphen verwendet werden.

Mit unserem Algorithmus FASbT geben wir einen parametrisierten Algorithmus für ein Problem an, für das, soweit wir wissen, zuvor kein solcher Algorithmus bekannt war. Dass Feedback Arc Set mit der Lösungsgröße

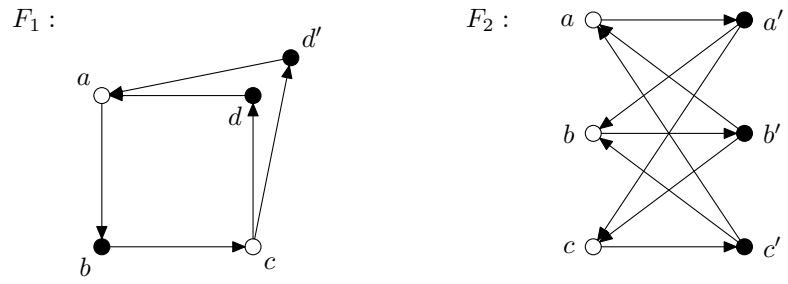

Abbildung 5.6: Die „verbotenen Teilgraphen“ F_1 und F_2. Schwarze und weiße Knoten verdeutlichen die Bipartition des Graphen.

als Parameter auf bipartiten Turniergraphen in FPT ist, ist einfach zu sehen, indem wir einen unkomplizierten Algorithmus simple-FASbT der Laufzeit $O(4^k \cdot n^\omega)$ betrachten, wobei $\omega \approx 2{,}376$ der Exponent der Laufzeit des schnellsten Matrixmultiplikationsalgorithmus [CW90] ist.

5.2.1 Der einfache Algorithmus simple-FASbT

Wir nutzen die Aussage von Korollar 3.3.2, dass jeder bipartite Turniergraph, der nicht kreisfrei ist, einen Kreis der Länge vier besitzt, und die Aussage von Lemma 3.3.3, dass das Umdrehen der Kanten eines minimalen Feedback Arc Set eines gerichteten Graphen diesen Graphen kreisfrei macht.

Der Algorithmus simple-FASbT funktioniert wie folgt: Wir suchen im gegebenen bipartiten Turniergraphen T nach einem Kreis der Länge vier, indem wir die Adjazenzmatrix A mit sich selbst multiplizieren und in der entstehenden Matrix A' einen Eintrag $a'_{ij} \geq 1$ suchen, sodass auch $a'_{ji} \geq 1$ gilt. Die Knoten mit den Indizes i und j sind dann durch Pfade der Länge zwei in beide Richtungen miteinander verbunden, liegen sich also in einem Kreis der Länge vier gegenüber. Die anderen beiden Knoten des Kreises lassen sich mit Hilfe der Adjazenzmatrix A bestimmen. Um einen Kreis zu finden oder die Kreisfreiheit des Graphen festzustellen, benötigen wir also $O(n^\omega)$ Zeit.

Gibt es keinen Kreis und gilt $k \geq 0$, so ist die Probleminstanz lösbar, da keine weiteren Kreise zu zerstören sind, und es kann zur Bestätigung „JA“ ausgegeben werden. Ansonsten führen wir vier Unterprogrammaufrufe aus, jeweils einen für jede der vier Kanten des Kreises in T. In jedem dieser Aufrufe wird derjenige Graph übergeben, der durch das Umdrehen der jeweiligen Kante entsteht, dazu der verringerte Parameter $k-1$ und die um die entsprechende Kante erweiterte Kantenmenge F. Auf diese Art und Weise probieren

wir jede Möglichkeit aus, einen Kreis der Länge vier zu unterbrechen, denn das Umdrehen eines minimalen Feedback Arc Set in einem gerichteten Graphen ist äquivalent zum Löschen der Kanten (siehe Lemma 3.3.3).

simple-FASbT(T, k, F)

Eingabe: Ein bipartiter Turniergraph $T = (V, E)$, die maximale Größe k der gesuchten Lösung und eine Kantenmenge F (zu Beginn gilt $F = \emptyset$ und $k \geq 0$).

Ausgabe: „JA“ und ein Feedback Arc Set F für T mit höchstens k Kanten, falls ein solches existiert, „NEIN“, wenn es für T kein solches Feedback Arc Set gibt.

Schritt 1: Suche in T nach einem Kreis der Länge vier. Falls keiner existiert und $k \geq 0$, so brich den Gesamtalgorithmus ab und gib „JA“ und F aus.

Schritt 2: Falls $k < 0$, brich den Programmaufruf ab und gib „NEIN“ aus.

Schritt 3: Sei *abcda* der in Schritt 1 gefundene Kreis. Verzweige in

1. simple-FASbT(T', $k-1$, $F \cup \{(a,b)\}$)
2. simple-FASbT(T', $k-1$, $F \cup \{(b,c)\}$)
3. simple-FASbT(T', $k-1$, $F \cup \{(c,d)\}$)
4. simple-FASbT(T', $k-1$, $F \cup \{(d,a)\}$)

wobei T' aus T jeweils durch Umdrehen der in F einzufügenden Kante entsteht.

Wenn alle Unterprogrammaufrufe mit Antwort „NEIN“ abgebrochen werden, so brich den Programmaufruf ab und gib „NEIN“ aus.

Gibt einer der Programmaufrufe „JA“ und eine Menge F' zurück, so gib ebenfalls „JA“ und F' aus.

Der Algorithmus verzweigt sich bei jedem Aufruf in vier Fälle. Da die Rekursion in Schritt 2 abgebrochen wird, wenn $k \leq 0$ und sich k bei jedem Unteraufruf des Algorithmus um eins verringert, erhalten wir einen Suchbaum mit Verzweigungsfaktor vier bei einer Suchtiefe von k. In jedem Knoten des Suchbaumes benötigen wir eine Laufzeit von $O(n^\omega)$, um zu überprüfen, ob der Graph Kreise enthält und gegebenenfalls einen Kreis zu finden, sodass simple-FASbT eine Gesamtlaufzeit von $O(4^k \cdot n^\omega)$ hat.

Der Algorithmus FASbT, den wir im Folgenden vorstellen, hat eine ähnliche rekursive Grundstruktur, aber wir führen eine feinere Fallunterscheidung durch, die uns eine bessere asymptotische Laufzeit ermöglicht.

5.2.2 Der Algorithmus FASbT

Wir verfahren analog zum parametrisierten Algorithmus für Feedback Arc Set auf Turniergraphen von Raman und Saurabh [RS04]. Zunächst beobachten wir, dass jeder bipartite Turniergraph, in dem es Kreise der Länge vier mit gemeinsamen Knoten gibt, einen der zwei Teilgraphen F_1 und F_2 aus Abbildung 5.6 enthält. Wir testen für einen Eingabegraphen mit Parameter k, ob dieser einen der beiden „verbotenen Teilgraphen“ enthält. Wenn ja, betrachten wir in einem Suchbaum alle Möglichkeiten, sämtliche Kreise in dem entsprechenden Teilgraphen durch Umdrehen einer minimalen Kantenmenge zu zerstören. Wenn die Probleminstanz lösbar ist, muss mindestens eine dieser Kantenmengen Teil eines kleinsten Feedback Arc Sets sein. Dasselbe führen wir rekursiv für alle dabei entstehenden Graphen durch, bis es im jeweiligen Graphen keinen „verbotenen Teilgraphen“ mehr gibt oder das Feedback Arc Set mehr als k Kanten enthält und verworfen werden kann.

Wenn der Graph keine Teilgraphen F_1 und F_2 mehr enthält, existieren nur noch knotendisjunkte Kreise der Länge vier und keine Kreise größerer Länge. Aus jedem Kreis nehmen wir jeweils eine beliebige Kante in unser Feedback Arc Set auf. Wenn das Feedback Arc Set dann höchstens k Kanten enthält, haben wir eine Lösung für die ursprüngliche Eingabeinstanz gefunden.

Hierbei erhalten wir einen Suchbaum der Tiefe k mit einem Verzweigungsfaktor von 3,38 und einem Zeitaufwand von $O(n^6)$ pro Knoten des Suchbaums für das Finden der verbotenen Teilgraphen, sodass wir eine Gesamtlaufzeit von $O(3{,}38^k \cdot n^6)$ erhalten.

Jetzt geben wir eine detailliertere Beschreibung des Algorithmus FASbT.

FASbT(T, k, F)

Eingabe: Ein bipartiter Turniergraph $T = (V, E)$, die Lösungsgröße k und eine Menge F von gerichteten Kanten (zu Beginn gilt $F = \emptyset$ und $k \geq 0$).

Ausgabe: „JA“ und ein Feedback Arc Set F für T mit höchstens k Kanten, falls ein solches existiert, „NEIN“, wenn es für T kein solches Feedback Arc Set gibt.

Schritt 1: Falls T keine Kreise enthält und $k \geq 0$, brich ab und gib „JA“ und F aus.

Schritt 2: Falls $k < 0$, brich ab und gib „NEIN“ zurück.

Schritt 3: Überprüfe, ob T einen Teilgraphen F_1 (siehe Abbildung 5.6) enthält. Falls ja, verzweige in

1. FASbT(T', $k-1$, $F \cup \{(a,b)\}$)
2. FASbT(T', $k-1$, $F \cup \{(b,c)\}$)
3. FASbT(T', $k-2$, $F \cup \{(c,d),(c,d')\}$)
4. FASbT(T', $k-2$, $F \cup \{(c,d),(d',a)\}$)
5. FASbT(T', $k-2$, $F \cup \{(d,a),(c,d')\}$)
6. FASbT(T', $k-2$, $F \cup \{(d,a),(d',a)\}$),

wobei T' aus T jeweils durch das Umdrehen der in F einzufügenden Kanten entsteht. Liefert einer der Aufrufe „JA“, dann gib „JA“ und das zurückgelieferte F aus, ansonsten gib „NEIN“ aus.

Schritt 4: Überprüfe, ob T einen Teilgraphen F_2 (siehe Abbildung 5.6) enthält. Falls ja, verzweige in

1. FASbT(T', $k-2$, $F \cup \{(a,a'),(b,b')\}$)
2. FASbT(T', $k-2$, $F \cup \{(a,a'),(c,c')\}$)
3. FASbT(T', $k-2$, $F \cup \{(b,b'),(c,c')\}$)
4. FASbT(T', $k-2$, $F \cup \{(a,a'),(b',c)\}$)
5. FASbT(T', $k-2$, $F \cup \{(a,a'),(c',b)\}$)
6. FASbT(T', $k-2$, $F \cup \{(b,b'),(a',c)\}$)
7. FASbT(T', $k-2$, $F \cup \{(b,b'),(c',a)\}$)
8. FASbT(T', $k-2$, $F \cup \{(c,c'),(a',b)\}$)
9. FASbT(T', $k-2$, $F \cup \{(c,c'),(b',a)\}$)
10. FASbT(T', $k-3$, $F \cup \{(a',b),(a',c),(b',c)\}$)
11. FASbT(T', $k-3$, $F \cup \{(a',b),(a',c),(c',b)\}$)
12. FASbT(T', $k-3$, $F \cup \{(a',b),(c',a),(b',c)\}$)
13. FASbT(T', $k-3$, $F \cup \{(a',b),(c',a),(c',b)\}$)
14. FASbT(T', $k-3$, $F \cup \{(b',a),(a',c),(b',c)\}$)
15. FASbT(T', $k-3$, $F \cup \{(b',a),(a',c),(c',b)\}$)
16. FASbT(T', $k-3$, $F \cup \{(b',a),(c',a),(b',c)\}$)
17. FASbT(T', $k-3$, $F \cup \{(b',a),(c',a),(c',b)\}$),

wobei T' aus T jeweils wie in Schritt 3 entsteht. Liefert einer der Aufrufe „JA“, dann gib „JA“ und das zurückgelieferte F aus, ansonsten gib „NEIN“ aus.

Schritt 5: Löse das Problem auf dem verbliebenen Graphen T in Polynomialzeit: Nimm aus jedem gerichteten Viereck eine beliebige Kante, füge sie zu F hinzu und senke k jeweils um eins. Wenn dann $k \geq 0$, antworte „JA“ und gib F aus, ansonsten antworte „NEIN“.

Nach dieser Beschreibung des Algorithmus ist jetzt eine Analyse der Korrektheit und der Laufzeit erforderlich. Es gilt:

Satz 5.2.1 Feedback Arc Set *auf bipartiten Turniergraphen ist mit dem Algorithmus* FASbT *in Laufzeit* $O(3{,}38^k \cdot n^6)$ *lösbar.*

Der Beweis dieses Satzes setzt sich aus verschiedenen Teilen zusammen. Korollar 5.2.1, das eine direkte Folgerung aus Lemma 5.2.1 ist, wird uns im nächsten Abschnitt die Korrektheit des Algorithmus liefern.

In einem späteren Abschnitt beweisen wir dann mit Lemma 5.2.5, dass der Algorithmus in einer Laufzeit von $O(3{,}38^k \cdot n^6)$ arbeitet.

5.2.3 Die Korrektheit des Algorithmus

Wir zeigen jetzt Lemma 5.2.1, aus dem sich direkt die Aussage ableiten lässt, dass der FASbT-Algorithmus das Feedback-Arc-Set-Problem löst.

Lemma 5.2.1 *Wenn für einen Turniergraphen T ein Feedback Arc Set der Größe k existiert, so liefert der Programmaufruf* FASbT(T, k, F) *„JA“ und eine Menge $F \cup F'$ zurück, wobei F' ein Feedback Arc Set mit höchstens k Kanten für T ist. Gibt es kein solches Feedback Arc Set, so gibt der Aufruf „NEIN“ zurück.*

Beweis: Wir beweisen induktiv, dass der Algorithmus korrekt ist. Als Induktionsanfang betrachten wir zum einen den Fall, dass der eingegebene Turniergraph T kreisfrei ist und zum anderen den Fall, dass $k < 0$. Danach nehmen wir im Induktionsschritt an, dass alle Unterprogrammaufrufe korrekt sind und beweisen, dass dann auch der aufrufende Algorithmus korrekt arbeitet.

Induktionsanfang Sei also zunächst $k < 0$. Dann kann es kein Feedback Arc Set der Größe k für den eingegebenen Graphen geben. Der Algorithmus bricht in einem solchen Fall in Schritt zwei mit der Ausgabe „NEIN“ ab, arbeitet also korrekt.

Wenn der Eingabegraph T kreisfrei ist und $k \geq 0$ gilt, so ist $F' := \emptyset$ ein Feedback Arc Set für T. In diesem Fall gibt der Algorithmus direkt im ersten Schritt „JA“ und die Eingabemenge $F = F \cup F'$ aus.

Induktionsannahme Für einen Programmaufruf FASbT(T, k, F) sind alle Unterprogrammaufrufe FASbT(T', k', $F \cup A$) korrekt, d. h. wenn es ein Feedback Arc Set der Größe k' für T' gibt, liefert der Unterprogrammaufruf „JA“ und eine Menge $(F \cup A) \cup F'$ zurück, wobei F' ein Feedback Arc Set mit höchstens k' Kanten für T' ist. Ansonsten gibt der Aufruf „NEIN“ aus.

Induktionsschritt Sei jetzt (T, k, F) die Eingabe des Algorithmus, wobei T nicht kreisfrei ist und $k \geq 0$ gilt. Wir haben oben angenommen, dass alle Unteraufrufe von FASbT korrekt sind und zeigen jetzt, dass dann die richtige Antwort ausgegeben wird.

Da T nicht kreisfrei ist und $k \geq 0$, geschieht in den Schritten 1 und 2 des Algorithmus nichts. Wir unterscheiden jetzt die drei Fälle, dass T einen Teilgraphen F_1 besitzt, dass T keinen Teilgraphen F_1, aber einen Teilgraphen F_2 besitzt und dass T keinen der beiden Teilgraphen F_1 und F_2 besitzt.

Fall 1: T besitzt einen Teilgraphen F_1. Dann muss jedes Feedback Arc Set für T Kanten enthalten, deren Entfernen die beiden Kreise $abcda$ und $abcd'a$ in F_1 zerstört. Betrachten wir nun, welche Kanten dafür in Frage kommen. Wir können F_1 mit dem Löschen nur einer Kante kreisfrei machen, indem wir (a, b) oder (b, c) entfernen. Es ist jedoch möglich, dass es wegen der Gesamtstruktur des Graphen T besser ist, andere Kanten zu löschen, um ein insgesamt kleineres Feedback Arc Set zu erhalten. Um den Kreis $abcda$ zu unterbrechen, brauchen wir dann eine der Kanten (c, d) und (d, a) und für den Kreis $abcd'a$ eine der Kanten (c, d') und (d', a).

Jedes Feedback Arc Set für T muss darum (a, b) oder (b, c) oder eine der Kanten (c, d) und (d, a) sowie eine der Kanten (c, d') und (d', a) enthalten.

Wird FASbT mit einem bipartiten Turniergraphen T aufgerufen, der einen Teilgraphen F_1 enthält, verzweigt sich der Algorithmus in Schritt 3 in sechs Unterprogrammaufrufe. In jedem dieser Unteraufrufe von FASbT wird eine der gerade beschriebenen Möglichkeiten ausprobiert, die Kreise in F_1 zu zerstören. Die entsprechenden Kanten werden in T umgedreht, in F eingefügt und k pro umgedrehter Kante um eins gesenkt.

Wenn die Probleminstanz lösbar ist, führt auf diese Weise mindestens ein Zweig unseres Suchbaums zum Erfolg. Denn gibt es ein Feedback Arc Set $\hat{F}$ mit höchstens k Kanten für T, so gibt es in

Schritt 3 mindestens einen Programmaufruf FASbT(T', k', $F \cup A$) mit $A \subseteq \hat{F}$.

Dann ist $\hat{F} \setminus A$ ein Feedback Arc Set mit höchstens $k' = k - |A|$ Kanten für T'. Nach der Induktionsannahme liefert der Unterprogrammaufruf und damit auch der Gesamtalgorithmus in diesem Fall „JA“ und eine Menge $(F \cup A) \cup \tilde{F}$ zurück, wobei $\tilde{F}$ ein Feedback Arc Set mit höchstens k' Kanten für T' ist.

Dies ist die gewünschte Ausgabe $F \cup F'$ für FASbT(T, k, F), da $F' := A \cup \tilde{F}$ ein Feedback Arc Set mit höchstens k Kanten für T ist; der Algorithmus arbeitet also korrekt.

Gibt es hingegen kein Feedback Arc Set der Größe k für T, lässt sich T also nicht durch das Umdrehen von k Kanten kreisfrei machen, gibt der Algorithmus ebenfalls die richtige Antwort zurück. Denn dann gibt es auch kein Feedback Arc Set der Größe $k - j$ für einen Graphen T', der aus T durch das Umdrehen von j Kanten entsteht. Deswegen kann keiner der Unterprogrammrufe in Schritt 3 erfolgreich sein. Da die Unterprogrammaufrufe nach Induktionsannahme eine korrekte Ausgabe liefern, geben sie alle „NEIN“ zurück, sodass insgesamt „NEIN“ ausgegeben wird.

Fall 2: T besitzt keinen Teilgraphen F_1, aber einen Teilgraphen F_2. In diesem Fall geschieht in Schritt 3 des Algorithmus nichts. In Schritt 4 werden wie in Fall 1 alle Möglichkeiten ausprobiert, die drei Kreise $aa'bb'a$, $aa'cc'a$ und $bb'cc'b$ in F_2 durch das Umdrehen eines minimalen Feedback Arc Sets zu zerstören. Es gibt neun solche Feedback Arc Sets mit nur zwei Kanten. Sie bestehen jeweils aus einer der Kanten (a, a'), (b, b') und (c, c'), da diese zu jeweils zwei Kreisen in F_2 gehören, und einer Kante des dritten Kreises. Des Weiteren gibt es acht minimale Feedback Arc Sets der Größe drei für F_2, die aus jedem Kreis eine der zwei Kanten enthalten, die zu keinem anderen Kreis gehören.

Analog zu Fall 1 können wir sehen, dass FASbT(T, k, F) für einen Turniergraphen T, der einen Teilgraphen F_2, nicht aber einen Teilgraphen F_1 enthält, genau dann „JA“ und $F \cup F'$ zurückgibt, wobei F' ein Feedback Arc Set mit höchstens k Kanten für T ist, wenn ein solches Feedback Arc Set existiert.

Fall 3: T besitzt weder einen Teilgraphen F_1 noch einen Teilgraphen F_2. Dann sind, wie wir in Lemma 5.2.2 zeigen werden, alle Kreise der Länge vier knotendisjunkt. In Lemma 5.2.3 zeigen wir zudem, dass es in diesem Fall auch keine größeren Kreise gibt. Alle Kreise in T sind also knotendisjunkt und haben die Länge vier. Dann ist jede

Menge, die aus je einer beliebigen Kante jedes Kreises besteht, ein kleinstes Feedback Arc Set für T. Ist die Größe eines solchen Feedback Arc Sets höchstens k, so ist die Eingabeinstanz lösbar, ansonsten nicht.

Rufen wir den Algorithmus also mit einem Turniergraphen T ohne Teilgraphen F_1 und F_2 auf, so geschieht in den Schritten drei und vier nichts. In Schritt 5 fügt der Algorithmus ein kleinstes Feedback Arc Set, das wie oben beschrieben gebildet wird, zur eingegebenen Menge F hinzu. Für jede eingefügte Kante wird k um eins gesenkt, sodass k genau dann kleiner als Null ist, wenn es kein Feedback Arc Set mit höchstens k Kanten für T gibt. Genau in diesem Fall wird „NEIN“ ausgegeben, ansonsten „JA“ und die um ein kleinstes Feedback Arc Set für T erweiterte Menge F. □

Eine direkte Folgerung aus Lemma 5.2.1 ist das folgende Korollar.

Korollar 5.2.1 *Für den Anfangsaufruf* FASbT(T, k, $\emptyset$) *gibt der Algorithmus genau dann „JA“ und ein Feedback Arc Set der Größe k für T zurück, falls ein solches Feedback Arc Set existiert.*

In Fall 3 des Beweises von Lemma 5.2.1 haben wir die Aussagen zweier Lemmata benutzt, die wir im Folgenden beweisen. Lemma 5.2.2 enthält die für den gesamten Algorithmus zentrale Aussage, dass jeder bipartite Turniergraph, in dem es zwei Kreise der Länge vier mit gemeinsamen Knoten gibt, einen der Graphen F_1 und F_2 als Teilgraph besitzt. Lemma 5.2.3, das wir im Anschluss beweisen werden, besagt, dass es in einem bipartiten Turniergraphen ohne Teilgraphen F_1 und F_2 keine Kreise einer größeren Länge als vier gibt.

Lemma 5.2.2 *Besitzt ein bipartiter Turniergraphen $T = (V_1 \cup V_2, E)$ keinen der Graphen F_1 und F_2 (siehe Abbildung 5.6) als Teilgraph, so sind alle Kreise der Länge vier in T knotendisjunkt.*

Beweis: Wir betrachten alle Möglichkeiten, wie sich zwei Kreise der Länge vier in einem bipartiten Turniergraphen T überschneiden können und zeigen, dass in jedem Fall einer der „verbotenen Teilgraphen“ F_1 und F_2 aus Abbildung 5.6 vorkommt.

Fall 1: Die Kreise besitzen zwei gemeinsame Kanten. Dann müssen sie genau drei gemeinsame Knoten besitzen. In diesem Fall enthält T den Teilgraphen F_1 (siehe Abbildung 5.6).

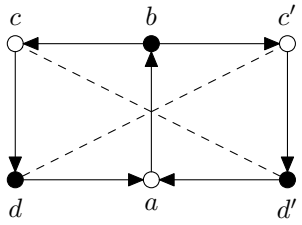

Abbildung 5.7: Darstellung von Fall 2 im Beweis von Lemma 5.2.2. Zwei Kreise der Länge vier besitzen genau eine gemeinsame Kante (a, b).

Fall 2: Die Kreise besitzen genau eine gemeinsame Kante (a, b) und damit, da T ein bipartiter Turniergraph ist, genau zwei gemeinsame Knoten. Dann besitzt T den Teilgraphen in Abbildung 5.7, wobei $abcda$ und $abc'd'a$ die beiden Kreise mit der gemeinsamen Kante (a, b) sind.

Da der Graph T ein bipartiter Turniergraph ist, muss es noch jeweils eine Kante zwischen c und d' und zwischen c' und d geben. Besitzt T die Kante (c', d), so gibt es in T die Kreise $dabcd$ und $dabc'd$, T enthält dann also einen Teilgraphen F_1. Analog verhält es sich mit der Kante (c, d'). Gibt es in T die Kanten (d, c') und (d', c), so ist der Teilgraph isomorph zu F_2 (siehe Abbildung 5.6).

Fall 3: Die Kreise besitzen keine gemeinsame Kante, aber einen oder zwei gemeinsame Knoten.

Fall 3a: Es gibt zwei gemeinsame Knoten. Diese müssen entweder beide aus der Knotenmenge V_1 oder aus der dazu disjunkten Knotenmenge V_2 kommen. Dann enthält T einen zu F_1 isomorphen Teilgraphen. Die gemeinsamen Knoten entsprechen den Knoten a und c von F_1 in Abbildung 5.6.

Fall 3b: Es gibt genau einen gemeinsamen Knoten. Dann enthält T den Teilgraphen in Abbildung 5.8. Gibt es in T die Kante (b', c), so existieren die Kreise $abcda$ und $ab'cda$. Analog findet man auch mit der Kante (c, d') einen Teilgraphen F_1. Enthält T hingegen keine dieser Kanten, so existieren die Kanten (c, b') und (d', c). Dann gibt es die Kreise $ab'c'd'a$ und $cb'c'd'c$ und damit wiederum einen Teilgraphen F_1. □

Für Lemma 5.2.1 benötigen wir des Weiteren die Aussage von Lemma 5.2.3, das wir jetzt beweisen.

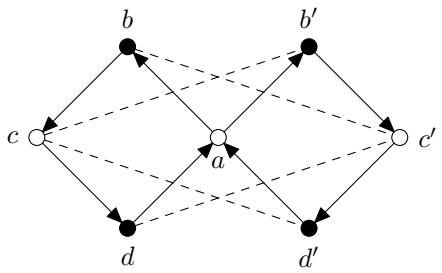

Abbildung 5.8: Darstellung von Fall 3b im Beweis von Lemma 5.2.2. Zwei Kreise der Länge vier besitzen genau einen gemeinsamen Knoten a.

Lemma 5.2.3 *Wenn in einem bipartiten Turniergraphen alle gerichteten Kreise der Länge vier knotendisjunkt sind, existieren keine Kreise größerer Länge.*

Beweis: Angenommen, es gäbe in einem bipartiten Turniergraphen T, in dem alle Kreise der Länge vier knotendisjunkt sind, Kreise der Länge l mit $l \geq 6$. Sei C ein solcher Kreis. Nach Lemma 5.2.4, das wir im Anschluss an diesen Beweis zeigen, liegt jeder Knoten von C auf einem Kreis der Länge vier, der nur aus Knoten von C besteht. Da in T alle Kreise der Länge vier knotendisjunkt sind, bedeutet dies, dass die Länge von C durch vier teilbar sein muss. Sei C entsprechend $v_1v_2 \cdots v_lv_1$, wobei $l \geq 6$ und l durch vier teilbar ist.

Da T ein bipartiter Turniergraph ist, muss sowohl eine der Kanten (v_1, v_4) und (v_4, v_1) als auch eine der Kanten (v_2, v_5) und (v_5, v_2) in E enthalten sein.

Fall 1: $(v_1, v_4) \in E$.

Dann ist $v_1v_4v_5 \cdots v_lv_1$ ein gerichteter Kreis der Länge $l-2$. Da $l-2$ nicht durch vier teilbar ist, ist dies ein Widerspruch zur Annahme.

Fall 2: $(v_2, v_5) \in E$.

In diesem Fall ist $v_1v_2v_5 \cdots v_lv_1$ ein Kreis der Länge $l-2$ in T, wiederum im Widerspruch zur Annahme.

Fall 3: $(v_4, v_1) \in E$ und $(v_5, v_2) \in E$.

Dann gibt es in T die gerichteten Vierecke $v_1v_2v_3v_4v_1$ und $v_2v_3v_4v_5v_2$, die im Widerspruch zur Voraussetzung nicht knotendisjunkt sind. □

Zum Ende dieses Korrektheitsbeweises zeigen wir jetzt noch Lemma 5.2.4, auf dessen Aussage wir in Lemma 5.2.3 vorgegriffen haben.

Lemma 5.2.4 *In einem bipartiten Turniergraphen $T = (V, E)$ liegt jeder Knoten, der auf einem Kreis C einer Länge größer als vier liegt, auch auf einem Kreis der Länge vier, der nur aus Knoten auf C besteht.*

Beweis: Sei $v_1 \in V$ ein Knoten, der auf einem Kreis C der Länge l mit $l > 4$ liegt. Wir zeigen induktiv, dass v_1 auf einem Kreis der Länge vier liegt, der nur aus Knoten auf Kreis C besteht.

Induktionsanfang Wir betrachten den Fall $l = 6$. Der Kreis C besteht dann aus einem geschlossenen Pfad $v_1v_2v_3v_4v_5v_6v_1$. Weil T ein bipartiter Turniergraph ist, muss es in E genau eine der Kanten (v_1, v_4) und (v_4, v_1) geben.

Fall 1: $(v_4, v_1) \in E$.

Dann ist der Pfad $v_1v_2v_3v_4v_1$ ein gerichtetes Viereck in T.

Fall 2: $(v_1, v_4) \in E$.

Dann ist $v_4v_5v_6v_1v_4$ ein gerichtetes Viereck in T.

Induktionsannahme Für jeden Knoten $v \in V$, der in T auf einem Kreis C der Länge l mit $l \leq m$ liegt, gibt es in T einen Kreis der Länge vier, der nur aus Knoten auf C besteht.

Induktionsschritt Sei nun $l = m + 2$, und entspreche C dem geschlossenen Pfad $v_1v_2v_3v_4 \cdots v_lv_1$. Es gilt wiederum, dass es in E eine der Kanten (v_1, v_4) und (v_4, v_1) geben muss.

Fall 1: $(v_4, v_1) \in E$.

Dann ist der Pfad $v_1v_2v_3v_4v_1$ ein gerichtetes Viereck in T.

Fall 2: $(v_1, v_4) \in E$.

In diesem Fall ist $C' = v_4 \cdots v_lv_1v_4$ ein Kreis der Länge $l - 2$. Nach Induktionsannahme existiert dann ein Kreis der Länge vier durch v_1, der nur aus Knoten von C', also auch von C, besteht. □

Mit dem vorigen Lemma ist der Beweis der Korrektheit von FASbT abgeschlossen, und wir können mit der Laufzeitanalyse beginnen.

5.2.4 Die Laufzeit des Algorithmus

In diesem letzten Abschnitt des Kapitels unterziehen wir den FASbT-Algorithmus einer genauen Analyse und zeigen, dass er für einen Turniergraphen mit n Knoten in einer Zeit von $O(3{,}38^k \cdot n^6)$ läuft.

Lemma 5.2.5 *Der angegebene Algorithmus für* Feedback Arc Set *auf bipartiten Turniergraphen läuft in einer Zeit von* $O(3{,}38^k \cdot n^6)$.

Beweis: Um einen Teilgraphen F_1 zu finden oder festzustellen, dass ein Graph keinen Teilgraphen F_1 besitzt, benötigt man eine Laufzeit von $O(n^\omega)$, wie wir gleich in Lemma 5.2.6 beweisen. Dabei ist n^ω mit $\omega \approx 2{,}376$ wieder die Laufzeit des schnellsten Matrixmultiplikationsalgorithmus [CW90]. Einen Teilgraphen F_2 kann man in Laufzeit $O(n^6)$ finden.

Die Verzweigung in die sechs Fälle in Schritt 3 ergibt einen Verzweigungsfaktor von 3,24, die Verzweigung in die siebzehn Fälle in Schritt 4 einen Verzweigungsfaktor von 3,38. Bei einer Verzweigungstiefe von k erhalten wir damit eine Gesamtlaufzeit von $O(3{,}38^k \cdot n^6)$. □

Jetzt zeigen wir nur noch, dass wir einen Teilgraphen F_1 in $O(n^\omega)$ Zeit finden können. Dieser Beweis ist zwar für die Abschätzung der Gesamtlaufzeit des Algorithmus nicht relevant, da jedoch im Algorithmus immer erst nach einem Teilgraphen F_1 gesucht wird und nur dann nach die kompliziertere Suche nach dem Teilgraphen F_2 erfolgt, wenn kein Teilgraph F_1 gefunden wird, ist es praktisch relevant, diese häufiger durchgeführte Suche schnell erledigen zu können.

Lemma 5.2.6 *Ob ein gegebener bipartiter Turniergraph* $T = (V, E)$ *mit* n *Knoten den Teilgraphen* F_1 *enthält, lässt sich in einer Laufzeit von* $O(n^\omega)$ *bestimmen.*

Beweis: Sei $V = \{v_1, v_2, \ldots, v_n\}$. Wir bilden die $(n \times n)$-Adjazenzmatrix A von T mit $a_{ij} = 1$, wenn $(v_i, v_j) \in E$ und $a_{ij} = 0$, wenn $(v_i, v_j) \notin E$.

Sei $A' := A^2$ die Matrix, die entsteht, wenn wir A mit sich selbst multiplizieren (Laufzeit $O(n^\omega)$). Da jeder Eintrag a'_{ij} von A' nach der Formel

$$a'_{ij} := a_{i1} \cdot a_{1j} + a_{i2} \cdot a_{2j} + \cdots + a_{in} \cdot a_{nj}$$

gebildet wird, gilt: $a'_{ij} = l$ genau dann, wenn es l gerichtete Wege der Länge zwei von i nach j gibt.

Wir suchen in A' nach einem Eintrag $a'_{ij} \geq 2$. Wenn zusätzlich $a'_{ji} \geq 1$, dann existiert ein Teilgraph F_1, da es in T zwei Pfade der Länge zwei von v_i nach v_j und einen von v_j nach v_i gibt. Diese drei gerichteten Pfade sind alle knotendisjunkt, da T ein Turniergraph ist und daher zwischen zwei Knoten nur eine Kante besitzt. Somit entspricht v_i dem Knoten c von F_1 und v_j dessen Knoten a in Abbildung 5.6. Da es zwischen v_i und v_j zwei gerichtete Wege gibt, können wir mit Hilfe der Adjazenzmatrix A in $O(n)$ Zeit die zwei Knoten finden, über die v_i und v_j verbunden sind. Diese zwei Knoten

entsprechen dann den Knoten d und d' des Graphen F_1. Derjenige Knoten, der auf dem Pfad von v_j nach v_i liegt, entspricht dem Knoten b von F_1.

Wenn kein Eintrag $a_{ij} \geq 2$ mit $a_{ji} \geq 1$ in A existiert, dann gibt es keinen Teilgraphen F_1 in T.

Für die Matrixmultiplikation benötigen wir eine Laufzeit von $O(n^\omega)$, für das Durchsuchen von A' eine Laufzeit von $O(n^2)$, insgesamt also $O(n^\omega)$ Zeit. □

Kapitel 6

Zusammenfassung und Ausblick

Wir haben in dieser Arbeit neue Algorithmen für festparameter-handhabbare Feedback-Set-Probleme auf gerichteten Graphen vorgestellt und zudem aufgezeigt, in welchem Umfang parametrisierte Algorithmen und Komplexitätsergebnisse für diese Probleme bisher bekannt sind. Die aktuellen Ergebnisse (siehe Abbildung 1.2 und 1.3) machen parametrisierte Algorithmen für Feedback-Set-Probleme auf Turniergraphen und bipartiten Turniergraphen zu einer interessanten Alternative zur Approximation.

Problemkernreduktionen können es ermöglichen, Probleminstanzen beweisbar so stark zu verkleinern, dass sie mit herkömmlichen, nicht-parametrisierten Algorithmen gelöst werden können. Wir haben eine Problemkernreduktion für Feedback Arc Set auf Turniergraphen angegeben, die eingegebene Probleminstanzen auf eine Größe von $O(k^2)$ verkleinert. Weitere nichttriviale Problemkerne für Feedback-Set-Probleme sind bislang nicht bekannt.

Mit Hilfe der Methode der iterativen Kompression konnten wir einen Algorithmus für Feedback Vertex Set angeben, der mit einer Laufzeit von $O(2^k \cdot n^2(\log n + k))$ eine niedrige Basis für den exponentiellen Vorfaktor aufweist. Damit haben wir gezeigt, dass die iterative Kompression auch für Probleme auf gerichteten Graphen ein vielversprechender Ansatz ist. In der nächsten Zeit gilt es, die Möglichkeiten dieser Methode weiter auszuleuchten.

Unser Algorithmus für Feedback Arc Set auf bipartiten Turniergraphen ist der erste nichttriviale parametrisierte Algorithmus für dieses Problem. Wir konnten gegenüber dem trivialen Algorithmus eine Verkleinerung des exponentiellen Vorfaktors der Laufzeit von 4^k auf $3{,}38^k$ erreichen. Feedback-Arc-Set-Algorithmen auf bipartiten Turniergraphen benötigen bislang größere Laufzeiten als diejenigen auf Turniergraphen, ein Beweis der NP-Härte von Feedback Arc Set auf bipartiten Turniergraphen steht jedoch noch aus.

Anknüpfend an unsere Arbeit stellen sich folgende Fragen:

- Können wir mit der iterativen Kompression auch parametrisierte Algorithmen für Feedback Vertex Set auf weiteren Klassen gerichteter Graphen finden?
- Eignet sich die Methode der iterativen Kompression für Feedback Arc Set auf gerichteten Graphklassen?
- Lässt sich die Laufzeit unserer Algorithmen durch weitere Verfeinerungen noch verbessern, zum Beispiel mittels automatischer Suchbaumerstellung [GGHN04]?
- Ist Feedback Arc Set auf bipartiten Turniergraphen NP-vollständig? Der Beweis steht bislang noch aus, könnte aber mit den neuen Ergebnissen zur NP-Vollständigkeit des Problems auf Turniergraphen [Alo05, CTY05, Con05] in greifbare Nähe gerückt sein.
- Welche Größe haben Problemkerne für weitere Festparameter-handhabbare Feedback-Set-Probleme, insbesondere auch auf bipartiten Turniergraphen?

Eine Beantwortung dieser Fragen würde zu einem weitaus tieferen Verständnis von Feedback-Set-Problemen auf gerichteten Graphen führen und könnte Anhaltspunkte in der weiterhin offenen Frage, ob Feeback Vertex Set und Feeback Arc Set auf allgemeinen gerichteten Graphen in FPT sind, geben.

Danksagung

Ich danke Rolf Niedermeier, Michael Dom, Jiong Guo und Falk Hüffner für die Betreuung meiner Diplomarbeit. Sie alle haben mich für parametrisierte Algorithmen begeistert, mir kontinuierlich und mit viel Geduld für Fragen und Diskussionen zu Verfügung gestanden, Anregungen und Rückmeldung gegeben und für ein sehr angenehmes Arbeitsklima gesorgt.

Außerdem möchte ich meinen Eltern und meinem Bruder danken, die mich in meinem Studium stets unterstützt haben.

Literaturverzeichnis

[ACN05] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. In *Proceedings of the thirty-seventh Annual ACM Symposium on Theory of Computing STOC'05*, pages 684–693. ACM Press, 2005. 8, 27

[Alo05] N. Alon. Ranking tournaments. *SIAM Journal on Discrete Mathematics*, 2005. To appear. 10, 27, 64

[BBF99] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999. 25

[BT92] J. Bang-Jensen and C. Thomassen. A polynomial algorithm for the 2-path problem for semicomplete digraphs. *SIAM Journal on Discrete Mathematics*, 5:366–376, 1992. 27

[CDZ01] M.-C. Cai, X. Deng, and W. Zang. An approximation algorithm for feedback vertex sets in tournaments. *SIAM Journal on Computing*, 30(6):1993–2007, 2001. 25

[CDZ02] M.-C. Cai, X. Deng, and W. Zang. A min-max theorem on feedback vertex sets. *Mathematics on Operations Research*, 27(2):361–371, 2002. 10, 18, 24, 25

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill, 2001. 46

[Con05] V. Conitzer. Computing slater rankings using similarities among candidates. Technical Report RC23748, IBM, 2005. 10, 27, 64

[CTY05] P. Charbit, S. Thomassé, and A. Yeo. The mimimum feedback arc set problem is NP-hard for tournaments. *Combinatorics, Probability and Computing*, 2005. To appear. 10, 27, 64

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. 49, 60

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. 9

[DF03] C. Demetrescu and I. Finocchi. Combinatorial algorithms for feedback problems in directed graphs. *Information Processing Letters*, 86(3):129–136, 2003. 25

[DFL+05] F. Dehne, M. R. Fellows, M. A. Langston, F. A. Rosamond, and K. Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. In *Proceedings of the 11th Annual International Conference on Computing and Combinatorics (COCOON'05)*, volume 3595 of *Lecture Notes in Computer Science*, pages 859–869. Springer, 2005. 38, 47

[DS05] I. Dinur and S. Safra. On the hardness of approximating minimum vertex-cover. *Annals of Mathematics*, 162(1):439–486, 2005. 26

[ENSS98] G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998. 23, 25, 27

[Fer04] H. Fernau. A top-down approach to search-trees: Improved algorithmics for 3-hitting set. Technical report, Electronic Colloqium on Computational Complexity, 2004. 10, 26

[Fer05] H. Fernau. *Parameterized Algorithmics: A Graph-Theoretic Approach*. Habilitationsschrift, Universität Tübingen, Germany, April 2005. 10, 26, 37

[FL99] R. Focardi and F. L. Luccio. Minimum feedback vertex set in k-dimensional hypercubes. Technical Report CS-99-21, Technical Report Series in Computer Science, Università ca' Foscari di Venezia, Italy, 1999.

[FPR99] P. Festa, P. M. Pardalos, and M. G. C. Resende. Feedback set problems. In D. Z. Du and P. M. Resende, editors, *Handbook of Combinatorial Optimization*, pages 209–258. Kluwer Academic Publishers, Boston, 1999.

[Fre75] M. L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975. 46

[Gav77] F. Gavril. Some NP-complete problems on graphs. In *Proceedings of the 11th Conference on Information Science and Systems*, pages 91–95, John Hopkins University, Baltimore, 1977. 27

[GGH+05] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, and S. Wernicke. Improved fixed-parameter algorithms for two feedback set problems. In *Proceedings of the 9th Workshop on Algorithms and Data Structures WADS'05*, volume 3608 of *Lecture Notes in Computer Science*, pages 158–168. Springer, 2005. 38, 47

[GGHN04] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004. 64

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Comp., 1979. 24, 25

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. 26

[GW96] M. X. Goemans and D. P. Williamson. Primal-dual approximation algorithms for feedback problems. In *Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization (IPCO'96)*, volume 1084 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 1996. 25

[Kan92] V. Kann. *On the Approximability of NP-complete Optimization Problems*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 1992. 25

[Kar72] R. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. 7, 10, 23, 24, 25, 27

[KP97] V. Kann and A. Panconesi. Hardness of approximation. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 13–30. J. Wiley & Sons, 1997.

[Kul99] O. Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223:1–72, 1999.

[Luc76] C. L. Lucchesi. *A Minmax Equality for Directed Graphs*. PhD thesis, University of Waterloo, Ontario, Canada, 1976. 26, 27

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Clarendon Press, Oxford, 2006. To appear. 9

[PY91] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991. 26

[Ram88] V. Ramachandran. Finding a minimum feedback arc set in reducible flow graphs. *Journal of Algorithms*, 9(3):299–313, 1988. 26, 27

[RG00] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill Higher Education, 2000. 7

[Ros82] B. K. Rosen. Robust linear algorithms for cutsets. *Journal of Algorithms*, 3(3):205–217, 1982. 7

[RS04] V. Raman and S. Saurabh. Improved parameterized algorithms for feedback set problems in weighted tournaments. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC'04)*, volume 3162 of *Lecture Notes in Computer Science*, pages 260–270. Springer, 2004. To appear in *Theoretical Computer Science*. 10, 28, 29, 30, 48, 51

[RSV04] B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004. 38

[Sha79] A. Shamir. A linear time algorithm for finding minimum cutsets in reduced graphs. *SIAM Journal on Computing*, 8(4):645–655, 1979. 24

[Sla61] P. Slater. Inconsistencies in a schedule of paired comparisons. *Biometrika*, 48:303–312, 1961. 26

[Spe89] E. Speckenmeyer. On feedback problems in digraphs. In *Proceedings of the 15th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'89)*, volume 411 of *Lecture Notes in Computer Science*, pages 218–231. Springer, 1989. 10, 24, 25

[Tar72] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972. 35

[vZ05] A. van Zuylen. Deterministic approximation algorithms for ranking and clustering problems. Technical Report 1431, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, USA, 2005. 27

[WHT05] F.-H. Wang, C.-J. Hsu, and J.-C. Tsai. Minimal feedback vertex sets in directed split-stars. *Networks*, 45(4):218–223, 2005.

[WLS85] C.-C. Wang, E. L. Lloyd, and M. L. Soffa. Feedback vertex sets and cyclically reducible graphs. *Journal of the ACM*, 32(2):296–313, 1985. 24

[Yan78] M. Yannakakis. Node- and edge-deletion NP-complete problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC'78)*, pages 253–264. ACM Press, 1978. 24, 25

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Jena, den 23. Dezember 2005

Anke Truß