Studienarbeit

Das "Feedback Vertex Set"-Problem

Ronald Kriemann

17. Oktober 1997

Inhaltsverzeichnis

1. Einführung

Diese Studienarbeit soll einen Überblick über verschiedene Algorithmen zum Lösen von "Feedback Vertex Set"-Problemen bieten. Ein Feedback Vertex Set (FVS) eines Graphen ist dabei eine Menge von Knoten, deren Entfernen den Graphen azyklisch macht. Das Problem selbst besteht darin, eine möglichst kleine Menge von Knoten zu finden, die ein FVS bilden.

Dieses Problem tritt in Theorie und Praxis recht häufig auf. Anhand der folgenden drei Beispiele soll dies verdeutlicht werden.

Ein Beispiel aus der Programmverifikation. Programme bestehen meist aus einer Reihe von Schleifen. Somit können Berechnungsfolgen beliebig lang werden. Ein FVS im Flußdiagramm eines solchen Programmes zerlegt diese in beschränkte Teilfolgen. Falls man die Korrektheit dieser Teilfolgen zeigen kann, erfolgt ein Korrektheitsbeweis des gesamten Programmes über eine Induktion der einzelnen Teile. Ähnliches trifft auch für das Testen von Schaltkreisen zu.

Ein anderes Beispiel betrifft das Auflösen von Verklemmungszuständen. In einem solchen System hat man mehrere wechselwirkende Prozesse. In einer Graphendarstellung des Systems stellen die Knoten die einzelnen Prozesse dar. Eine Kante zwischen zwei Prozessen gibt an, daß ein Prozeß nicht fortfahren kann, bevor sich der andere Prozeß nicht beendet hat. Falls es einen Zyklus in einem solchen Graphen gibt, befindet sich das System in einem Verklemmungszustand: alle Prozesse warten auf das Beenden eines anderen Prozesses. Hier wird ein minimales FVS benötigt, um die Anzahl der zu terminierenden Prozesse möglichst klein zu halten.

Das letzte Beispiel stammt aus der numerischen Mathematik. Dort kann man das Lösen von Konvektions-Diffusions-Gleichungen mit einem starken Konvektionsanteil beschleunigen, wenn man die Unbekannten so anordnet, daß ihre Richtung dem Konvektionsfluss entspricht. Im Idealfall hat die Systemmatrix einer solchen Gleichung eine Dreiecksgestalt und ist in einem Schritt lösbar. In der Graphendarstellung der Matrix entspricht der Idealfall einem azyklischen Graphen. Den Graphen der Matrix erhält man, indem die Unbekannten der Gleichung die Knotenmenge bilden und eine Kante zwischen den Knoten i und j eingefügt wird, falls das Matrixelement in Zeile i und Spalte j einen Wert ungleich 0 hat. Falls Zykel in diesem Graph existieren, sucht man nach einer kleinen Menge von Knoten die ein FVS bilden. Eine Umordnung der Unbekannten bezüglich dieses FVS ergibt eine Matrix mit Blockstruktur. Die FVS-Knoten bilden dabei einen relativ kleinen Block, während der Rest wieder Dreiecksgestalt hat.

Leider lassen sich aber minimale FVS i.allg. nur mit großem Aufwand berechnen. Allerdings wird ein solches FVS nur in den wenigsten Fällen benötigt. Es genügt auch ein FVS, welches "fast"- oder quasi-minimal ist, dessen Berechnung aber schnell erfolgen kann. Eine Beschreibung solcher Algorithmen stellt den Hauptteil dieser Studienarbeit dar.

Zunächst werden in Abschnitt 2 die verwendeten Begriffe aus der Graphen- und Komplexitätstheorie definiert und die einzelnen FVS-Probleme vorgestellt. In Abschnitt 3 geht es um Algorithmen zum Bestimmen von FVSs in gerichteten Graphen. Algorithmen für ungerichteten Graphen werden in Abschnitt 4 behandelt.

2. Definitionen

2.1. Graphentheoretische Voraussetzungen

Def 2.1 (Graph) Ein (ungerichteter) *Graph* $G = (V(G), E(G))$ besteht aus einer nichtleeren *Knotenmenge* $V = V(G)$ und einer *Kantenmenge* $E = E(G) \subseteq V \times V$. Eine Kante $e \in E$ entspricht somit einem ungeordnetem Paar $e = (u, v)$ von Knoten $u, v \in V$.

Zwei Knoten heißen zueinander *adjazent*, falls sie durch eine Kante verbunden sind. Die entsprechende Kante ist *inzident* zu den beiden Knoten. Die Anzahl der zu einem Knoten v inzidenten Kanten bezeichnet man als *Grad* von v bzw. $d(v)$ (von engl: degree).

$G' = (V', E')$ wird als *Untergraph* von $G = (V, E)$ bezeichnet, falls $V' \subseteq V$ und $E' \subseteq (V' \times V') \cap E$. Sei $S \subseteq V$. Der durch S induzierte Untergraph von G ist $G(S) = (S, E_S)$, wobei $E_S = (S \times S) \cap E$. Die Reduktion $G - S$ von G um S ist der Untergraph $G(V - S)$. Analog läßt sich die Reduktion um einen Untergraphen G' von G definieren als $G - G' = G - V(G')$ und die Reduktion von G um $K \subseteq E$ als den Untergraphen $G(K) = (V, E - K)$.

Eine endliche Folge $p = v_0 v_1 v_2 \dots v_{k-1} v_k$ mit $(v_i, v_{i+1}) \in E$, bezeichnet man als *Pfad*. Ein Pfad heißt *geschlossen*, falls $v_0 = v_k$ und *offen*, falls $v_0 \neq v_k$. Desweiteren bezeichnet die Länge $|p|$ eines Pfades p die Anzahl der Kanten entlang dieses Pfades. Ein Pfad heißt *Weg*, falls alle Knoten verschieden sind und *Kantenzug*, wenn alle Kanten verschieden sind. Ein geschlossener Kantenzug $\zeta = v_1 v_2 \dots v_n v_1$ heißt *Zyklus*, wenn $v_1 \dots v_n$ ein Weg ist. Einen Graphen ohne Zykel nennt man *azyklisch*.

Zwei Knoten $u, v \in V$ heißen *zusammenhängend*, falls es einen Weg von u nach v gibt. Ein Graph heißt zusammenhängend, wenn alle Knotenpaare zusammenhängend sind.

Def 2.2 (Digraph) Ein *Digraph* (von engl: directed graph) $G = (V, E)$ ist ein Graph mit Knotenmenge V und Kantenmenge E, wobei die Knotenpaare in E geordnet sind. Für $(u, v) \in E$ (kurz: $u \to v$) heißt u *Anfangsknoten* und v *Endknoten*.

Eine Kante, deren Endknoten (Anfangsknoten) v ist, bezeichnet man als *einfallende* (*ausgehende*) Kante von v. Der *Eingangsgrad* $id(v)$ (von engl: inward degree) des Knoten v bezeichnet die Anzahl der einfallenden Kanten und der *Ausgangsgrad* $od(v)$ (von engl: outward degree) die Anzahl der ausgehenden Kanten. Sei $E_{in}(v)$ die Menge der einfallenden Kanten von v und $E_{out}(v)$ die Menge der ausgehenden Kanten von v. Die Menge der zu v inzidenten Kanten sei $E(v) = E_{in}(v) \cup E_{out}(v)$.

Pfad, Weg, Kantenzug und Zyklus sind analog zum ungerichteten Fall definiert.

Als *unterliegender Graph* eines Digraphen $G = (V, E)$ wird der ungerichtete Graph $G' = (V, E)$ bezeichnet. Offensichtlich gehen dabei die Richtungen der Kanten verloren. Ein Digraph ist *schwach zusammenhängend*, wenn sein unterliegender Graph zusammenhängend ist. Eine alternative Definitionen von schwach zusammenhängend, die ebenfalls in der Literatur vorkommt, ist: Für alle $u, v \in V$ gibt es einen Pfad von u nach v oder von v nach u. Ein Digraph heißt *stark zusammenhängend*, falls es für jedes Knotenpaar $u, v \in V$ einen gerichteten Weg von u nach v gibt.

Im folgenden wird auf die Unterscheidung von Graph und Digraph verzichtet, falls aus dem Kontext hervorgeht, was gemeint ist.

Def 2.3 (Netzwerk) Ein *Netzwerk* $N = (V, E, c)$ ist ein schwach zusammenhängender Digraph, mit einer *Kapazitätsfunktion* $c : E \to \mathbb{R}_+$. Der Wert $c(e)$ einer Kante $e \in E$ wird als *Kapazität* von e bezeichnet.

Ein Knoten $a \in V$ heißt *Quelle*, wenn er Eingangsgrad 0 hat. Ein Knoten $b \in V$ heißt *Senke*, wenn er Ausgangsgrad 0 hat.

Def 2.4 (Fluß) Ein *Fluß* in einem Netzwerk $N = (V, E, c)$ von Quelle a zur Senke b ist eine Funktion $f : E \to \mathbb{R}_+$ mit

1. für alle $e \in E$ gilt: $f(e) \leq c(e)$,
2. $\sum_{e \in E_{out}(a)} f(e) = \sum_{e \in E_{in}(b)} f(e)$ und
3. für alle $v \in V - \{a, b\}$ gilt: $\sum_{e \in E_{in}(v)} f(e) = \sum_{e \in E_{out}(v)} f(e)$.

Eine Verallgemeinerung dieser Definition ist ein Fluß in einem Netzwerk mit k Quelle-Senke-Paaren $\{(a_i, b_i)\}_{i=1}^{k}$. Bedingung (2) gilt dann für jedes Paar (a_i, b_i) separat.

Ein *Schnitt* in einem Netzwerk N zwischen der Quelle a und der Senke b ist eine Menge $C \subseteq E(N)$, so daß $N - C$ in mindestens zwei nicht zusammenhängende Untergraphen N_1 und N_2 zerfällt, wobei $a \in N_1$ und $b \in N_2$ gilt. Ein *minimaler Schnitt* ist ein Schnitt zwischen a und b mit minimaler Kapazität. Für Algorithmen zu Berechnung eines minimalen Schnittes siehe [BK96] und [HO92].

Für eine detailierte Einführung in Graphen und Netzwerke siehe [CH94].

2.2. Feedback Vertex Set Probleme

Def 2.5 (Feedback Vertex Set) Sei $G = (V, E)$ ein Graph. Ein Feedback Vertex Set (FVS) von G ist eine Teilmenge $F \subseteq V$, so daß F aus jedem Zyklus von G mindestens einen Knoten enthält. Es gilt also: $G - F$ ist azyklisch.

Im folgenden werden verschiedene Problemstellung bezüglich FVS vorgestellt.

FVS-Problem

Das klassische FVS-Problem lautet:

Eingabe:	Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$
Ausgabe:	*Ja*, falls: G hat FVS F mit $\|F\| = k$
	Nein, sonst.

Eine alternative Definition ist:

Eingabe:	Graph $G = (V, E)$
Ausgabe:	FVS F mit $\|F\| = \min\{\|F'\| : F'$ ist ein FVS von $G\}$

Die Aufgabe besteht hier darin, ein FVS mit minimaler Kardinalität zu finden. Dies ist i.allg. nicht eindeutig bestimmt (z.B., wenn G ein Zyklus mit Länge ≥ 2 ist).

SUBSET-FVS-Problem

Dieses Problem stellt eine Verallgemeinerung des FVS-Problems dar. Hier wird nur eine Teilmenge der Zykel des Graphen betrachtet. Diese Teilmenge ist definiert durch eine Menge $S \subseteq V$, den Sonderknoten.

Eingabe:	Graph $G = (V, E)$ und $S \subseteq V$.
	Seien $Z = \{\zeta : \zeta$ ist ein Zyklus und $\zeta \cap S \neq \emptyset\}$ und $G_Z = G(Z)$.
Ausgabe:	FVS F von G_Z mit $\lvert F\rvert = \min\{\lvert F'\rvert : F'$ ist ein FVS von $G_Z\}$

Gewichtetes FVS-Problem

Die hier betrachtete Graph ist mit einer Gewichtsfunktion versehen, die das gesuchte FVS definiert:

Eingabe:	Graph $G = (V, E, w)$, mit $w : V \to \mathbb{R}_+$
Ausgabe:	FVS F mit $\sum_{v \in F} w(v) = \min\{\sum_{v \in F'} w(v) : F'$ ist FVS von $G\}$

Gesucht ist ein FVS mit minimalem Gewicht. Falls w eine konstante Funktion ist, entspricht das Problem dem normalen FVS-Problem.

Im folgenden bezeichne $F_{opt}(G)$ ein minimales FVS des Graphen G. Die Minimalität bezieht sich dabei auf das jeweils betrachtete FVS-Problem.

2.3. NP-Vollständigkeit

Für die Lösung der in Kapitel 2.2 genannten Probleme können verschiedene Algorithmen verwendet werden. Interessant ist in der Praxis, wie sich diese Algorithmen im Hinblick auf Platz- und Zeitverbrauch für unterschiedliche Eingabelängen z.B. für unterschiedliche Graphen verhalten.

Der Algorithmus wird hierzu in einer Form kodiert, die einen objektiven Vergleich mit anderen Algorithmen ermöglicht. Diese Form sind die *Turing-Maschinen* (kurz: TM). Eine TM besteht aus einem unendlichen Arbeitsband, auf dem zu Beginn die Eingabe steht, und einer endlichen Zustandskontrolle. Diese Kontrolle, verbunden mit einem Arbeitsfeld auf dem Band, definiert die jeweils nächste Aktion der TM.

Formell ist eine TM A ein Sechs-Tupel $A = (Q, \Sigma, \Gamma, q_0, \delta, F)$, wobei Q die Zustandsmenge, $\Sigma \subseteq \Gamma$ das Eingabealphabet, Γ das Arbeitsalphabet, q_0 der Anfangszustand, δ die Transitionsrelation und F die Endzustände sind. Eine Transition $(q, a, a', \{l, n, r\}, q')$ bedeutet dabei: A befindet sich im Zustand q, auf dem Arbeitsfeld steht a und A wechselt in Zustand q', schreibt a' auf das Arbeitsfeld und bewegt sich um ein Feld auf dem Band nach links oder rechts oder bleibt auf dem alten Feld. A hört auf, falls ein Endzustand erreicht ist. Die *Sprache* einer TM A (kurz: $L(A)$) ist definiert als die Menge aller Eingaben $w \in \Sigma^*$, für die A stoppt.

Man unterscheidet zwei Arten von TM. Zum einen die *deterministischen TM* (kurz: DTM), bei denen es zu jedem Paar von Zustand und Buchstabe auf dem Arbeitsfeld nur eine Transition gibt. Zum anderen die *nichtdeterministischen TM* (kurz: NTM), bei denen es zu jedem solchen Zustand-Buchstabe-Paar mehrere Transition gibt.

Die Probleme, die eine TM A lösen soll, werden über dem Eingabealphabet Σ in geeigneter Weise kodiert. So besteht das FVS-Problem mit $k \in \mathbb{N}$ aus allen Graphen die ein FVS mit Kardinalität k haben. Für ein Problem P stoppt A für alle $w \in P$. Es gilt also: $L(A) = P$. Probleme können somit auch als Sprachen angesehen werden.

Def 2.6 Sei $f : \mathbb{N} \to \mathbb{R}_+$. Eine TM A heißt $f(n)$*-zeitbeschränkt* (*platzbeschränkt*), falls für alle Berechnungen mit Eingabelänge n gilt : A benötigt $\leq f(n)$ Zeit (Platz). f wird dann als Zeit- (Platz-) *Komplexitätsfunktion* von A bezeichnet. Desweiteren gilt: $f(n) \in O(g(n))$, falls es eine Konstante c gibt, so daß $|f(n)| \leq c|g(n)|$, für alle $n \geq 0$.

Mit Hilfe der Komplexitätsfunktion kann man zwei Klassen von Problemen definieren:

$$\begin{aligned} \mathbf{P} &= \{P : \text{ ex. DTM } A \text{ mit } L(A) = P \text{ und } A \text{ arbeitet in Polynomzeit}\} \\ \mathbf{NP} &= \{P : \text{ ex. NTM } A \text{ mit } L(A) = P \text{ und } A \text{ arbeitet in Polynomzeit}\} \end{aligned}$$

Wie man sofort sieht, gilt $\mathbf{P} \subseteq \mathbf{NP}$. Offen bleibt aber ob $\mathbf{P} \subset \mathbf{NP}$. Im folgenden gehe ich stets von der Annahme aus, daß $\mathbf{P} \neq \mathbf{NP}$.

Ein wichtiger Begriff, der sich mit den Klassen **P** und **NP** verbindet, ist der Begriff der **NP**-*Vollständigkeit*. Ein Problem P heißt **NP**-*vollständig*, wenn $P \in \mathbf{NP}$ und alle anderen Probleme aus **NP** auf P "*reduziert*" (siehe Kap. 2.3.1) werden können. Anders ausgedrückt bedeutet dies: falls man P in Polynomzeit lösen kann, kann man alle Probleme aus **NP** in Polynomzeit lösen, und somit wäre $\mathbf{P} = \mathbf{NP}$.

Die Probleme aus Abschnitt 2.2 sind **NP**-vollständig (siehe [Kar72], [Bra87]). Daraus folgt, daß für wachsende Eingabelängen der Berechnungsaufwand zum Lösen der Probleme exponentiell steigt. Um dies zu vermeiden, geht man über zu Approximationsalgorithmen (siehe 2.3.2), deren Lösungen im allgemeinen ausreichen, auch wenn sie nicht optimal sind.

Für eine detailiertere Einführung in TM und **NP**-Vollständigkeit siehe [GJ79] oder [HU88].

2.3.1. Reduktionen und gleich schwere Probleme

Def 2.7 Seien $P_1, P_2 \subseteq \Sigma^*$ zwei Probleme. P_1 heißt *reduzierbar* auf P_2 (kurz: $P_1 \preceq P_2$), falls es eine berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ gibt mit: für alle $w \in \Sigma^* : w \in P_1 \Leftrightarrow f(w) \in P_2$.

Man kann also jede Instanz eines Problems aus P_1 darstellen als eine Instanz eines Problems in P_2.

Im folgenden werden zwei Beispiele für Reduktionen aufgezeigt. Als erstes wird dabei das Feedback-Edge-Set-Problem (FES) betrachtet. Das FES-Problem entspricht dem FVS-Problem. Anstatt einer minimalen Knotenmenge wird allerdings eine minimale Kantenmenge gesucht, deren Entfernen den Graphen azyklisch macht.

FES $\preceq$ FVS

Sei $G = (V, E)$ ein gerichteter Graph. Konstruiere den Digraphen $G' = (V', E')$ mit $V' = E$ und $((v_1, v_2), (v_3, v_4)) \in E'$ gdw. $v_2 = v_3$. Es gilt dann: $M \subseteq E$ ist ein FES in G gdw. M ist ein FVS in G'. Durch diese Reduktion kann auch das SUBSET-FES-Problem auf das SUBSET-FVS-Problem reduziert werden.

FVS $\preceq$ FES

Sei $G = (V, E)$ ein Digraph. Konstruiere den Digraphen G' folgendermaßen: spalte jeden Knoten $v \in V$ in zwei neue Knoten v_1 und v_2. Alle einfallenden Kanten von v werden zu einfallenden Kanten von v_1 und alle ausgehenden Kanten von v werden zu ausgehenden Kanten von v_2. Zusätzlich wird eine Kante von v_1 nach v_2 eingefügt. Offensichtlich gilt: jedes FVS F in G wird durch diese Transformation zu einem FES in G'.

Da sowohl FES $\preceq$ FVS als auch FVS $\preceq$ FES gilt, bezeichnet man beide Probleme als *gleich schwer*.

Ein erster Beweis für die **NP**-Vollständigkeit des FVS-Problems erfolgte durch Karp in [Kar72]. Dort wurde eine Reduktion auf das Vertex-Cover-Problem durchgeführt. Eine Menge $X \subseteq V$ heißt *Vertex-Cover* (kurz: VC), falls jede Kante des Graphen zu einem Knoten in X inzident ist. Das VC-Problem besteht darin, ein VC minimaler Kardinalität zu finden.

2.3.2. Approximationsalgorithmen

Bei den folgenden Definitionen wird von einem Optimierungsproblem (Minimierungsproblem) ausgegangen, da man bei diesen Problemen ein Maß für die Güte der Lösung angeben kann. Dieses Maß (oder Wert) ist für jedes Problem geeignet zu definieren (z.B. beim FVS-Problem als Größe des FVS). Alle in Kapitel 2.2 genannten Probleme sind Optimierungsprobleme.

Def 2.8 Sei P ein Optimierungsproblem, und bezeichne $W_{opt}(I)$ den Wert einer optimalen Lösung für die Eingabe I von P. Sei A ein Algorithmus, der zu jeder Eingabe I von P eine zulässige Lösung[1] mit dem Wert $W(A, I)$ berechnet. Die *Güte* von A für die Eingabe I ist definiert durch:

$$r(A, I) = \frac{W(A, I)}{W_{opt}(I)}.$$

Es werden nun die Approximationsalgorithmen, wie sie in den folgenden Abschnitten vorkommen, definiert.

Def 2.9 Sei $\varepsilon > 0$. Ein Algorithmus A für ein Optimierungsproblem P heißt polynomieller ε-Approximationsalgorithmus, wenn er für jede Eingabe I von P eine zulässige Lösung mit $r(A, I) \leq 1 + \varepsilon$ in polynomieller Zeit bezüglich $|I|$ berechnet.

[1] Eine zulässige Lösung ist eine nicht unbedingt optimale Lösung des Problems, z.B. anstatt einem minimalen FVS ein FVS mit gößerer Kardinalität.

Def 2.10 Sei $F : \mathbb{N} \to \mathbb{R}_+$ eine Funktion. Ein Algorithmus A für ein Optimierungsproblem P heißt polynomieller $F(n)$-Approximationsalgorithmus, wenn er für jede Eingabe I von P eine zulässige Lösung mit $r(A, I) \leq 1 + F(|I|)$ in polynomieller Zeit bezüglich $|I|$ berechnet.

Probleme, für die man lediglich $F(n)$-Approximationsalgorithmen angeben kann, sind offensichtlich hartnäckiger als Probleme mit ε-Approximationsalgorithmen.

3. Algorithmen für gerichtete Graphen

3.1. Polynomzeit lösbare Probleme

In diesem Kapitel werden verschiedene Klassen von Graphen vorgestellt, bei denen das FVS-Problem in polynomieller Zeit exakt lösbar ist. Als erstes wurde dies für reduzierbare Fluß-Graphen durch Shamir in [Sha79] gezeigt. Später konnten dann durch geeignete Reduzierbarkeits-Eigenschaften weitere Klassen aufgestellt werden, die insgesamt eine Hierarchie bilden (siehe [WLS85] und [LSW88]).

3.1.1. Reduzierbare Flußgraphen

Reduzierbare Flußgraphen (Def. siehe unten) findet man oft in Verbindung mit Fluß-Diagrammen von Programmen. Eine exakte Lösung ist somit von großem Interesse für die Programmverifikation (siehe Abschnitt 1).

Def 3.1 (Flußgraph) Ein Graph $G = (V, E)$ heißt *Flußgraph*, falls G keine Mehrfachkanten enthält und es einen Knoten $v \in V$ gibt, so daß es für alle $u \in V$ einen Pfad von v nach u gibt.

Im folgenden wird ein Algorithmus auf Flußgraphen benutzt, der *Depth-First-Search-Algorithmus* (DFS) (siehe dazu auch [Tar72]). Der DFS-Algorithmus stellt eine Art dar, wie man einen (Fluß-) Graphen durchlaufen kann. Gestartet wird von einem Knoten in G. Man wählt dann eine beliebige, noch nicht besuchte Kante zu einem Nachfolger. Falls keine nichtbesuchten Kanten vorhanden sind, geht man zurück zu dem Knoten, den man zuletzt besucht hat und der noch nichtbesuchte Kanten hat. Dabei werden die einzelnen Knoten in einem Stack gespeichert. Daraus ergibt sich eine Einteilung der Kanten in *Rückwärtskanten* und *DAG-Kanten* (DAG von engl: directed acyclic graph). Die Rückwärtskanten sind Kanten $(v, v') \in E(G)$, mit: v' befindet sich bereits auf dem Stack, wenn v auf den Stack gelegt wird. Die DAG-Kanten sind alle anderen. Diese Einteilung ist nicht eindeutig, sondern hängt vom DFS-Durchlauf des Graphen ab.

Für einen Flußgraphen $G = (V, E)$ und einen DFS-Durchlauf α sei der DAG von G bezüglich α, der Graph $G_d^\alpha = (V, E_d^\alpha)$, wobei E_d^α die Menge der DAG-Kanten ist. Dabei gilt: falls eine Kante aus $E - E_d^\alpha$ zu G_d^α hinzugefügt wird, ensteht ein Zyklus.

Def 3.2 (Reduzierbarer Flußgraph) Ein Flußgraph G ist *reduzierbar*, falls für alle DFS-Durchläufe α der Graph G_d^α gleich ist.

Die Eigenschaft bestimmter Knoten, die in der folgenden Definition und dem nachfolgenden Theorem dargestellt wird, sorgt für die exakte Lösung des FVS-Problems in reduzierbaren Flußgraphen.

Def 3.3 Sei $G = (V, E)$ ein reduzierbarer Flußgraph und $S \subseteq V$. Ein Knoten v heißt *aktiv* im Untergraphen $G - S$, falls es in $G - S$ einen DAG-Pfad[2] von v zu einem Knoten v' gibt, so daß $(v', v) \in E$ eine Rückwärtskante ist. Ein aktiver Knoten heißt *maximal*, falls keiner seiner DAG-Nachfolger ein aktiver Knoten ist.

Theorem 3.1 Sei $G = (V, E)$ ein reduzierbarer Flußgraph und $S \subseteq V$ eine Teilmenge eines minimalen FVS von G. Falls $v \in V$ ein maximaler Knoten in G ist, dann ist auch $S \cup \{v\}$ eine Teilmenge eines minimalen FVS von G.

Zum Beweis siehe [Sha79, Theorem 3]. Daraus ergibt sich ein Algorithmus, der nach allen maximalen Knoten sucht, und sie zu dem FVS hinzufügt. Algorithmus 3.1 stammt aus [Sha79] und arbeitet in $O(|V|+|E|)$. Er bricht allerdings ab, falls der Graph kein reduzierbarer Flußgraph ist. In [Ros82] ist eine erweiterte Version dieses Algorithmus beschrieben, die für alle Graphen ein FVS bestimmt. Dies ist dann aber, außer für reduzierbare Flußgraphen, nicht notwendigerweise minimal.

Algorithmus 3.1 benutzt einen Stack mit den üblichen Operationen: push ($\leftarrow$) und pop ($\rightarrow$). Das oberste Element des Stacks wird mit *top* bezeichnet. l und n sind zusätzliche Markierungen für die Knoten.

$G = (V, E)$; $r \in V$;
$F := \emptyset$; $c := 1$; stack $:= \emptyset$; stack $\leftarrow r$;
1: $n(top) := c$; $c := c + 1$; $l(top) := 0$;
2: **if** es gibt eine nicht-markierte Kante $e = (top, v) \in E$ **then** markiere e
else goto 3;
if v wurde noch nicht besucht **then** stack $\leftarrow v$; goto 1; **end if**;
if (top, v) ist eine Rückwärtskante **then** $l(top) := \max\{l(top), n(v)\}$; goto 2; **end if**;
$l(top) := \max\{l(top), l(v)\}$; goto 2;
3: **if** $l(top) = n(top)$ **then** $F := F \cup \{top\}$; $l(top) := 0$; **end if**;
if $l(top) > n(top)$ **then** Abbruch {Graph ist kein reduzierbarer Flußgraph};
$v' := top$; stack $\rightarrow top$;
if stack $= \emptyset$ **then** goto 4;
else $l(top) := \max\{l(top), l(v')\}$; goto 2; **end if**;
4: FVS $:= F$;

Algorithmus 3.1: FVS-Algorithmus für reduzierbare Flußgraphen

[2] Die Bezeichnung DAG gibt im folgenden an, daß ein Element oder Teil eines azyklischen, gerichteten Graphen gemeint ist, z.B. DAG-Pfad als Pfad in einem DAG

3.1.2. Zwei-Wege-reduzierbare Graphen

Sei $G = (V, E)$, ein gerichteter Graph ohne Mehrfachkanten. Die Charakterisierung der zwei-Wege-reduzierbaren Graphen basiert auf Transformationen. Sei lauten:

- t_1 – Sei v ein Knoten mit einer Schlinge. Setze $G = G - \{v\}$.
- t_2 – Sei v ein Knoten mit Ausgangsgrad 0. Setze $G = G - \{v\}$.
- t_3 – Sei v ein Knoten mit Eingangsgrad 0. Setze $G = G - \{v\}$.
- t_4 – Sei v ein Knoten mit Ausgangsgrad 1. Verbinde v und seinen Nachfolger y zu einem einzelnen Knoten, mit Namen y.
- t_5 – Sei v ein Knoten mit Eingangsgrad 1. Verbinde v und seinen Vorgänger y zu einem einzelnen Knoten, mit Namen y.

$T(G, v)$ bezeichne den Graphen, den man nach Anwendung der Transformation T auf Knoten v im Graphen G erhält.

Def 3.4 (T-Sequenz) Sei G ein Graph. Eine *T-Sequenz* von G ist eine Folge $(T_1, v_1), \ldots, (T_k, v_k)$ von Transformationen und Knoten, so daß $v_i \in V(G_i)$ und Tranformation T_i auf Knoten v_i anwendbar ist. Dabei sei $G_0 = G$ und $G_{i+1} = T_i(G_i, v_i)$. Eine T-Sequenz heißt *vollständig*, falls $|V(G_k)| = 0$.

Ein Graph G heißt *zwei-Wege-reduzierbar*, wenn es eine vollständige T-Sequenz für G gibt. Es lassen sich desweiteren folgende Unterklassen angeben: G heißt *vorwärts-reduzierbar*, falls es eine vollständige T-Sequenz für G gibt, die nur die Transformationen t_1, t_2 und t_4 enthält, und G heißt *rückwärts-reduzierbar*, falls es eine vollständige T-Sequenz für G gibt, die nur die Transformationen t_1, t_3 und t_5 enthält.

In [LSW88] werden die folgenden beiden Theoreme bewiesen, die die Grundlage des Algorithmus 3.2 darstellen.

Theorem 3.2 Sei G ein Graph. Falls G zwei-Wege-reduzierbar ist und τ eine vollständige T-Sequenz von G ist, ist $\{y_1, \ldots, y_k\}$ ein minimales FVS von G, wobei $y_1, \ldots, y_k$ die Knoten von G sind, auf die die Transformation t_1 in τ angewandt wurde.

Theorem 3.3 Sei G ein Graph. Falls G zwei-Wege-reduzierbar ist und Transformation T_i auf Knoten v von G anwendbar ist, existiert eine vollständige T-Sequenz von G mit (T_i, v) als ihrem erstem Element.

In [LSW88, Abschnitt 5] wird ebenfalls bewiesen, daß die Klasse der reduzierbaren Flußgraphen aus Abschnitt 3.1.1 eine Unterklasse der rückwärts-reduzierbaren Graphen ist. Die vorwärts-reduzierbaren Graphen sind dagegen identisch mit einer anderen Graphenklasse, den *zyklisch-reduzierbaren* Graphen, die in [WLS85] beschrieben sind.

$i := 0$; $G_0 := G$; $F := \emptyset$;
while $V(G_i) \neq \emptyset$ **do**
    wähle $v \in V$ und $T \in \{t_1, \ldots, t_5\}$, so daß T auf v anwendbar ist;
    if "keine Transformation auf Knoten anwendbar" **then**
        Abbruch; { G ist nicht reduzierbar }
    else
        $G_{i+1} = T(G_i, v)$;
        if $T = t_1$ **then** $F := F \cup \{v\}$;
        $i := i + 1$;
end if; **end while**;
FVS $= F$;

Algorithmus 3.2: FVS-Algorithmus für zwei-Wege-reduzierbare Graphen

3.1.3. Smith/Walford Graphen

Die Klassen, die in diesem Abschnitt vorgestellt werden, leiten sich ab aus einem Algorithmus zur Bestimmung eines minimalen FVS für einen beliebigen Graphen. Da dieses Problem aber **NP**-vollständig ist (siehe Abschnitt 2.3), kann der Aufwand exponentiell groß werden. Beschrieben wurde der Algorithmus erstmals in [SW75]. Die hier wiedergegebene Form geht zurück auf [LSW88].

Def 3.5 Sei $G = (V, E)$ ein Graph und $H \subseteq V$. Der *Smith/Walford-assoziierte* (kurz: SW-assoziierte) Graph von H sei der Graph $A(G, H) = (V_H, E_H)$ mit $V_H = H \cup \{v : v \in V$ und $v \notin \zeta$ für alle Zykel ζ in $G - H\}$ und $E_H = \{(v, v') : (v, v') \in E$ und $v, v' \in V_H\}$. H heißt *wesentlich* in G, falls für alle $D \subseteq V(V_H)$ mit $|D| = |H| - 1$ in $A(G, H) - D$ mindestens ein Zyklus existiert. Falls H wesentlich ist und $|H| \leq i$, $i \in \mathbb{N}$, nennt man H eine *W_i-Menge* in G.

Mit dieser Definition erhält man Algorithmus 3.3. Er entfernt iterativ die jeweils kleinstmögliche wesentliche Menge aus G, bis kein Zyklus mehr existiert. Dabei ist zu beachten, daß die Kardinalität der wesentlichen Menge auf dem i-ten Schritt nicht notwendigerweise i ist.

Ein Graph G gehört zur Klasse der *Smith/Walford i-reduzierbaren* Graphen (kurz: SW_i-reduzierbar), falls der Smith/Walford Algorithmus nach $\leq i$ Schritten fertig ist. Die Laufzeit des Algorithmus 3.3 liegt für einen SW_i-reduzierbaren Graph in $O(i \cdot |V|^{2i} \cdot |E|)$, z.B. für SW_1-reduzierbare Graphen in $O(|V|^2 \cdot |E|)$.

Die Korrektheit des Algorithmus wird in [LSW88, Kapitel 4] gezeigt.

3.2. Planare Graphen

In diesem Abschnitt wird ein 1-approximativer Algorithmus (siehe Abschnitt 2.3.2) zur Bestimmung eines FVS für planare Graphen vorgestellt. An diese Graphen wird zusätzlich eine Bedingung über die Verteilung der einfallenden und ausgehenden Kanten gestellt. Solche

$F = \emptyset$; $i = \emptyset$;
while G enthält einen Zyklus **do**
    $i = i + 1$;
    while es gibt eine W_i-Menge in G **do**
        $S = W_i$-Menge mit minimaler Kardinalität;
        $F = F \cup S$;
        $G = G - A(G, S)$;
end while; **end while**;
FVS $= F$;

Algorithmus 3.3: FVS-Algorithmus für Smith/Walford Graphen

Graphen erhält man z.B. bei der Diskretisierung von partiellen Differentialgleichungen (siehe auch Abschnitt 1 und [Hac97, Kapitel 1]).

Def 3.6 (Planarer Graph) Ein Graph heißt *planar*, falls er in der Ebene ($\mathbb{R}^2$) auf eine Art und Weise neugezeichnet werden kann, so daß sich jedes Paar von Kanten höchstens an seinen Endknoten trifft.

One-Flow-Direction Bedingung

Da G ein planarer Graph ist, lassen sich die Kanten eines Knoten $v \in V(G)$ nach dem Winkel ordnen, den sie zu einer festen Achse einnehmen (siehe Abb. 3.4(i)). Sei $(e_1, e_2, \ldots, e_k)$ eine solche (geordnete) Kantenliste. Es gilt die *One-Flow-Direction* Bedingung (kurz: OFD), falls es für alle Knoten $v \in V(G)$ ein Liste $(e_1, e_2, \ldots, e_k)$ gibt, so daß für ein $l \in \{0, \ldots, k\}$ die Kanten e_i mit $1 \leq i \leq l$ ausgehende Kanten und die Kanten e_i mit $l < i \leq k$ einfallende Kanten sind. Anschaulich bedeuted dies, daß man den einfallenden und ausgehenden Kanten jeweils zwei, zusammenhängende, disjunkte Bereiche zuordnen kann. In Abb. 3.4(ii) ist eine nichterlaubte Situation dargestellt.

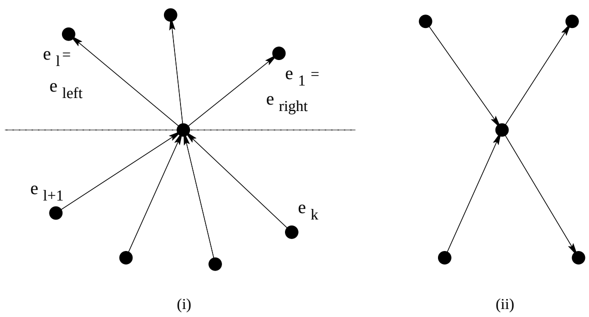

Abbildung 3.4: One-Flow-Direction Bedingung (i) und nichterlaubte Situation (ii)

Offensichtlich sind für ein FVS nur Knoten aus Zykeln von Interesse. Sei $G_C(G) = G - \{v \in V(G) : v \notin \zeta$ für alle Zykel ζ in $G\}$ der Graph, der nur aus Zykel besteht. Im folgenden wird nur der Graph $G_C(G)$ betrachtet, der Einfachheit halber aber wieder mit G benannt.

Aufgrund der OFD-Bedingung gibt es zu jedem $v \in V(G)$ ein eindeutiges Tupel $(e_1, \ldots, e_l)$ von ausgehenden Kanten. Bezeichne $e_{right}(v)$ e_1 und $e_{left}(v)$ e_l (siehe Abb. 3.4(i)).

Def 3.7 (Reduzierter Graph) Sei $G = (V, E)$ ein planarer Graph mit OFD-Bedingung. Es sei der *reduzierte Graph* G^+ definiert durch $G^+ = (V(G^+), E(G^+)$ mit $V(G^+) = V$ und $E(G^+) = \{e_{left}(v) : v \in V(G)\}$. Analog wird der Graph G^- definiert durch $V(G^-) = V(G)$ und $E(G^-) = \{e_{right}(v) : v \in V(G)\}$.

Minimale und maximale Zykel

Bezeichne C^+ die Menge der Zykel in G^+. Da jeder Knoten in $G = G_C(G)$ mindestens einen Nachfolger hat, hat jeder Knoten v in G^+ genau einen Nachfolger. Somit ist durch die Rekursion

$$v_0 = v, \quad v_{i+1} = \text{Nachfolger von } v_i,$$

eine eindeutige, unendliche Folge von Knoten in G^+ gegeben. Da aber $V(G^+)$ endlich ist, gibt es ein j, so daß $v_j = v_i$ für ein $i < j$. Somit ist $v_i, \ldots, v_j$ ein Zyklus in C^+. Da dies für alle Knoten möglich ist, erhält man eine Abbildung von $V(G^+)$ nach C^+. Bezeichne $\zeta(v)$ den jeweiligen, so erhaltenen Zyklus.

Sei $v \in V(G)$ mit Vorgänger v_P und Nachfolger v_S. Alle Kanten, die in dem Winkel zwischen den Kanten $(v, v_S) \in E(G)$ und $(v_P, v) \in E(G)$ (entgegen dem Uhrzeigersinn) liegen, bezeichnet man als *linke Kanten*. Sie bilden die Menge $E_{left}(v, \zeta)$. Die Kanten auf der anderen Seite, die *rechten Kanten*, ergeben die Menge $E_{right}(v, \zeta)$.

Bemerkung 3.1 Sei G stark zusammenhängend und ζ ein Zyklus. Seien $G_1, \ldots, G_k$ die stark zusammenhängenden Komponenten von $G - \zeta$. Es gilt: für alle G_i gibt es einen Knoten $v_i \in V(G_i)$ mit $v_i \notin \zeta$, und einen Knoten $v \in \zeta$, so daß $(v_i, v) \in E(G)$. Falls $(v_i, v) \in E_{left}(v, \zeta)$, gilt auch für alle anderen Kanten (v_j, v) mit $v_j \in G_i$ und $v_j \notin \zeta$, $(v_j, v) \in E_{left}(v, \zeta)$. Somit liegt der Untergraph G_i auf der linken Seite, dem *Inneren* von ζ. Entsprechend gehört G_i zum *Äußeren*, falls $(v_j, v) \in E_{right}(v, \zeta)$.

$\mathrm{Int}(\zeta)$ ist die Vereinigung aller linken G_i und $\mathrm{Ext}(\zeta)$ die Vereinigung aller rechten G_i. Der Abschluß von Int und Ext ist definiert als $\overline{\mathrm{Int}}(\zeta) := \mathrm{Int}(\zeta) \cup \zeta$ und $\overline{\mathrm{Ext}}(\zeta) := \mathrm{Ext}(\zeta) \cup \zeta$.

Die Zykel $\zeta \in C^+$ kann man auch als *minimale* Zykel betrachten, da in ihrem Inneren keine weiteren (kleineren) Zykel enthalten sind, also $\mathrm{Int}(\zeta) = \emptyset$ (siehe [Hac97, Theorem 3.5]).

Analog kann man diese Betrachtungen für $\zeta \in C^-$ wiederholen. Dabei ist C^- die Menge der Zykel in G^-. Es gilt allerdings: $\mathrm{Ext}(\zeta) = \emptyset$ (siehe [Hac97, Theorem 5.1]). Aus diesem Grund werden diese Zykel auch *maximale* Zykel genannt.

Konzentrische Zykel

Algorithmus 3.5 bestimmt sukzessiv die minimalen Zykel $\zeta_0, \ldots, \zeta_k$. Mit diesen Zykeln läßt sich nun wieder ein Graph definieren. Sei Z^+ dieser Graph mit $V(Z^+) = \{\zeta_0, \ldots, \zeta_k\}$ und

```
G_0 := G;
while V(G_i) ≠ ∅ do
        G'_i := G_C(G_i);
        if V(G'_i) = ∅ then stop;
        else
                wähle v ∈ V(G'_i);
                ζ_i := ζ(v);
                G_{i+1} := G'_i − ζ;
end if; end while;
```

Algorithmus 3.5: Algorithmus für minimale Zykel

einer Kantenmenge, die durch folgende partielle Ordnung definiert ist. Seien dazu $\zeta, \zeta' \in V(Z^+)$. Es gelte:

$$\zeta' > \zeta \quad gdw. \quad \overline{\mathrm{Int}}(\zeta) \subset \mathrm{Int}(\zeta') \text{ und } \overline{\mathrm{Ext}}(\zeta') \subset \mathrm{Ext}(\zeta).$$

Für die Kantenmenge gelte: $(\zeta, \zeta') \in E(Z^+)$, falls $\zeta' > \zeta$ und es existiert kein anderes $\zeta'' \in V(Z^+)$, mit $\zeta' > \zeta'' > \zeta$.

Der Algorithmus läßt sich aber auch zum Ermitteln der maximalen Zykel benutzen, indem man G^- anstatt G^+ beim Auffinden der Zykel (der Rekursion) verwendet. Analog zu Z^+ erhält man den Graphen Z^- mit $V(Z^-) = \{\zeta_0, \ldots, \zeta_l\}$, und für die Kantenmenge gelte: $(\zeta, \zeta') \in E(Z^-)$ falls $\zeta > \zeta'$, und es existiert kein anderes $\zeta'' \in V(Z^-)$ mit $\zeta > \zeta'' > \zeta$.

Man kann nun eine Relation zwischen den Zykeln in Z^+ und denen in Z^- einführen. Seien dazu $\zeta_+ \in V(Z^+)$ und $\zeta_- \in V(Z^-)$. Es gelte: ζ_+ *berührt* ζ_- (kurz: $\zeta_+ \circ \zeta_-$), falls

$$\zeta_+ \cap \zeta_- \neq \emptyset \quad \text{ und } \quad \mathrm{Int}(\zeta_+) \cap \zeta_- = \zeta_+ \cap \mathrm{Ext}(\zeta_-) = \emptyset.$$

Es wird nun eine Bewertungsfunktion definiert, durch die "geeignete" FVS-Knoten ermittelt werden können. Sei dazu $v \in V(G)$. Die Bewertungsfunktion setzt sich zusammen aus:

$$\varphi_+(v) := \varphi_+(v, Z^+) := \frac{1}{2}(|\{\zeta \in V(Z^+) : v \in \mathrm{Ext}(\zeta)\}| + |\{\zeta \in V(Z^+) : v \in \overline{\mathrm{Ext}}(\zeta)\}|).$$

$\varphi_+(v)$ ist also die gemittelte Anzahl von minimalen Zykel, für die v außerhalb bzw. außerhalb oder auf dem Rand liegt. Analog bildet man

$$\varphi_-(v) := \varphi_-(v, Z^-) := \frac{1}{2}(|\{\zeta \in V(Z^-) : v \in \mathrm{Int}(\zeta)\}| + |\{\zeta \in V(Z^-) : v \in \overline{\mathrm{Int}}(\zeta)\}|.$$

Def 3.8 Sei $G = (V, E)$ stark zusammenhängend und $v \in V$. Es sei folgende *Bewertungsfunktion* für v definiert:

$$\phi(v) := \phi(v, Z^+, Z^-) := \varphi_+(v, Z^+) + \varphi_-(v, Z^-).$$

Die Bewertungsfunktion gibt somit an, für wieviele minimale plus maximale Zykel sich der Knoten v durschnittlich im Äußeren (Inneren) befindet. In Abb. 3.6 sind verschiedene Werte der Bewertungsfunktion für ein Beispiel aus zwei minimalen und zwei maximalen Zykeln eingetragen.

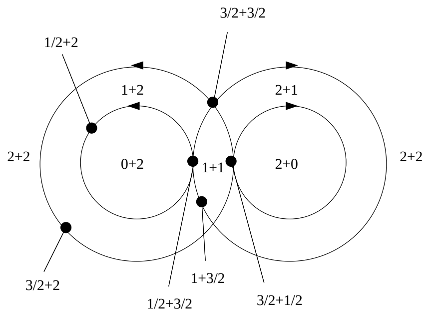

Abbildung 3.6: Bewertungsfunktion $\varphi_+ + \varphi_-$

Man kann zeigen, daß G azyklisch ist gdw. $\phi(v) = 0$ für ein $v \in V(G)$. Somit erhält man ein Abbruchkriterium für den FVS-Algorithmus. Durch das folgende Lemma wird außerdem die Wahl des nächsten FVS-Knoten bestimmt. Das Beispiel in Abb. 3.6 verdeutlicht dies.

Lemma 3.4 Das Minimum von ϕ wird auf der Knotenmenge

$$T := \{v \in V(G) : v \in \zeta_+ \cap \zeta_-, \zeta_+ \circ \zeta_-, \zeta_+ \in V(Z^+), \zeta_- \in V(Z^-)$$

angenommen.

Zum Beweis siehe [Hac97, Paragraph 5.4].

Konstruktion des FVS

In Algorithmus 3.7 wird die Bewertungsfunktion dazu benutzt, den nächsten Knoten für das spätere FVS auszuwählen. Sei dazu

$$\phi(v, G) := \phi(v, Z^+, Z^-),$$

wobei sich Z^+ und Z^- jeweils aus dem Graphen G ergeben. Es gelte außerdem

$$\phi_{min}(G) := \min\{\phi(v, G) : v \in \mathbb{R}^2\}.$$

Dieser Algorithmus ist 1-approximativ und liefert somit im schlechtesten Fall ein FVS, das doppelt so groß wie ein minimales FVS ist. Zum Beweis dieser Approximationseigenschaft siehe [Hac97, Paragraph 6.3]. Es sind dort auch einige Optimierungen nachzulesen, so daß die Komplexität des Algorithmus in $O(n)$ liegt.

3.3. Gewichtete Graphen

In diesem Abschnitt wird das SUBSET-FVS-Problem über gewichteten Graphen $G = (V, E, w)$ betrachtet, wobei w eine Gewichtsfunktion ist und $S \subseteq V$ die Menge der Sonderknoten, die die interessierenden Zykel definieren (siehe Abschnitt 2.2).

```
i = 0; G_i = G; F_i = ∅;
while φ_min(G_i) > 0 do
    v_i = ein v ∈ T(G_i) mit φ(v, G_i) = φ_min(G_i);
    G_{i+1} = G_i − {v_i};
    F_{i+1} = F_i ∪ {v_i};
    i = i + 1;
end while;
FVS = F_i;
```

Algorithmus 3.7: FVS-Algorithmus für planare Graphen

Der nachfolgende Algorithmus berechnet dabei Schnitte in *zirkulierenden* Netzwerken. Anders als bei normalen Netzwerken wird in den zirkulierenden Netzwerken jedes Quelle-Senke-Paar (a, b) mit einer Kante $a \to b$ verbunden. Diese Kante hat eine unendliche Kapazität. Man kann jede Instanz eines SUBSET-FES-Problems (siehe Abschnitt 2.3.1) auf eine Instanz eines Minimalen-Schnitt-Problems (siehe Abschnitt 2.1) in einem zirkulierendem Netzwerk reduzieren. Dabei wird jeder Knoten $s \in S$ in ein Quelle-Senke-Paar (a_s, b_s) transformiert, wobei alle einfallenden Kanten von s zu einfallenden Kanten von b_s werden und alle ausgehenden Kanten von s zu ausgehenden Kanten von a_s. Zusätzlich wird noch die Kante von a_s zu b_s eingefügt.

Im ersten Schritt des Algorithmus wird das Problem des minimalen Schnittes mittels linearer Programmierung (siehe Abschnitt 4.1.2) exakt gelöst. Die dabei berechnete Lösung ist aber nicht, wie benötigt, ganzzahlig. Anstatt zu runden, werden die Werte $d(e)$ für alle Kanten $e \in E$ als Entfernung des End- vom Anfangsknoten der Kante betrachtet.

Um die minimalen Schnitte zu finden, werden im zweiten Schritt um die Quellen und Senken Kugeln mit Radius r gebildet. Im folgenden wird beschrieben, wie eine solche Kugel um Knoten $s \in S$ aufgebaut wird.

Bezeichne $dist(u, v)$ die Länge des kürzesten Pfades p von u nach v, also $dist(u, v) = \sum_{e \in p} d(e)$. Eine Kugel mit Radius r enthält den Untergraphen, der durch alle Knoten u mit $dist(s, u) \leq r$ induziert wird. Bei den Kugeln um die Senken wird hierzu die Richtung der Kanten umgedreht.

Seien alle Knoten in V entsprechend ihrer Entfernung von s geordnet, also $v_0 = s$ und $dist(v_0, v_{i+1}) \geq dist(v_0, v_i)$. Sei $l_i = dist(v_0, v_i)$ und $A_i = \{v_0, \ldots, v_i\}$. Das Gewicht einer Menge A_i sei definiert durch:

$$w(A_i) = \frac{\tau^*}{k} + \sum_{e \in A_i \times A_i} c(e) \cdot d(e) + \sum_{e=(v_j, v_k) \in A_i \times (V - A_i)} c(e) \cdot (l_{i+1} - l_i).$$

Hierbei bezeichne τ^* die Kapazität eines minimalen Schnittes. Die gesuchte Kugel wird durch A_σ definiert, mit $\sigma = \min\{i : c(cut(A_i, V - A_i)) \leq \varepsilon \cdot w(A_i)\}$ und $\varepsilon > 0$. Die Funktion *cut* liefert dabei den Schnitt zwischen den beiden Untergraphen $G(A_i)$ und $G(V - A_i)$.

Die Vereinigung der Untergraph $G(R_1), \ldots, G(R_i)$ aus Algorithmus 3.8 stellt einen Schnitt zwischen den Quelle-Senke-Paaren (a_i, b_i) dar. Da die Reduktion auf eine Instanz

$G' := G$
$i := 0$
while $k' > \frac{k-1}{2}$ **do**
  $i := i + 1;$
  Sei (a_i, b_i) ein Quelle-Senke-Paar in G';
  Berechne die Kugeln um a_i und b_i;
  Sei R_i die Kugel mit der kleinsten Anzahl von Quelle-Senke-Paaren;
  G' := das Netzwerk, das durch die Knoten von N induziert wird, die nicht in R_i sind
  k' := Anzahl der Quelle-Senke-Paare in N'.
end while;
$i := i + 1;$
R_i := die Knoten in N';
$R = \{R_1, R_2, \ldots, R_i\};$

Algorithmus 3.8: Algorithmus zur Berechnung eines minimalen Schnitts

des Minimalen-Schnitt-Problems invertierbar ist, erhält man damit auch eine Lösung des SUBSET-FES-Problems und somit eine Lösung des SUBSET-FVS-Problems. Der Approximationsfaktor beträgt $O(\log^2 k)$ (siehe auch Abschnitt 3 in [ENSS95]).

In [ENSZ96] wird ein Algorithmus für ungerichtete Graphen beschrieben, der auf den gleichen Techniken basiert. Im Unterschied zum gerichteten Fall ist das Approximationsverhältnis dort $O(\log k)$.

4. Algorithmen für ungerichtete Graphen

4.1. FVS

4.1.1. Ein 1-approximativer Algorithmus

Der in diesem Abschnitt beschriebene Algorithmus stammt aus [BBF96] und liefert eine Lösung für das gewichtete FVS-Problem, deren Gewicht höchstens um den Faktor 2 von einer optimalen Lösung abweicht. In der Einführung dieses Artikels wird außerdem darauf hingewiesen, daß ein Erreichen eines wesentlich besseren Faktors, bis auf Spezialfälle, wahrscheinlich nicht mehr in Polynomzeit zu erreichen ist, wenn man Resultate von verwandten Problemen, z.B. Vertex-Cover (siehe Abschnitt 2.3.1), betrachtet.

Sei $G = (V, E, w)$ ein gewichteter Graph mit $w : V \to \mathbb{R}_+$ als Gewichtsfunktion. Sei $V(w) = \{u \in V : w(u) > 0\}$ und $G(w) = G(V(w))$ der durch $V(w)$ induzierte Untergraph von G. Für $U \subseteq V$ und $u \in U$ bezeichne $d(u, U)$ den Grad von u in $G(U)$.

Einige grundlegende Konzepte in diesem Abschnitt sind:

- $\{w_1, w_2, \ldots, w_k\}$ heißen *Zerlegung* von w, falls $\sum_i w_i(v) \leq w(v)$ für alle $v \in V$.
- Ein Zyklus ζ aus G heißt *semi-disjunkt*, falls für alle Knoten v in ζ, bis auf höchstens einen, $d(u, V) = 2$ gilt.

Die Basis des Algorithmus 4.1 bildet das folgende Theorem, in dem das Verhältnis des Gewichtes eines FVS zum Gewicht eines minimalen FVS abgeschätzt werden kann durch das Verhältnis der Gewichte in einem Untergraphen.

Theorem 4.1 Sei $G = (V, E, w)$ ein gewichteter Graph, $w_1, w_2, \ldots, w_k$ eine Zerlegung von w und F ein FVS von G mit $w(F) = \sum_i w_i(F)$. Dann gilt:

$$\frac{w(F)}{w(F_{opt}(G(w)))} \leq \max_i \{ \frac{w_i(F)}{w_i(F_{opt}(G(w_i)))}.$$

Der Beweis kann in [BBF96, Theorem 1] nachgelesen werden.

Die Arbeitsweise des Algorithmus 4.1 ist dann folgendermaßen: Ein Graph G wird in eine Folge von gewichteten Untergraphen zerlegt. In diesen Untergraphen wird eine "gute" Approximation eines minimalen FVS bezüglich des Untergraphen berechnet. Nach Theorem 4.1 erhält man eine "gute" Approximation eines globalen FVS.

Das folgende Theorem gibt die Approximation und das Zeitverhalten des Algorithmus an (siehe auch [BBF96, Theorem 8]).

Theorem 4.2 Algorithmus 4.1 findet ein FVS F von $G = (V, E, w)$ in der Zeit $O(|E| + |V| \log |V|)$ mit $w(F) \leq 2 \cdot w(F_{opt}(G))$.

Eine bessere Approximation eines minimalen FVS läßt sich für Graphen erzielen, bei denen der Grad der Knoten beschränkt ist. Dafür muß man allerdings die Techniken, die im unbeschränkten Fall eingesetzt werden, verfeinern und neue Reduktionen einführen. Diese Reduktionen betreffen zusammenhängende Untergraphen von G, die durch $A \subseteq V$ mit $|A| \leq 9$ induziert werden. Man erhält einen Algorithmus, der in $O(|V|^2)$ arbeitet und dessen Approximationgrad $2 - \frac{2}{3 \cdot \Delta - 2}$ beträgt. Dabei ist Δ der maximale Grad eines Knoten im Graphen. Mehr dazu in [BBF96, Kapitel 4].

4.1.2. Ansatz über lineare Programmierung

Wie bereits erwähnt, handelt es sich beim FVS-Problem um ein Optimierungs- bzw. ein Minimierungsproblem. Solche Probleme kann man mit der Methode der *Linearen Programmierung* (kurz: LP) lösen.

Die Aufgabe der LP besteht darin, eine lineare Funktion unter endlich vielen linearen Nebenbedingungen zu optimieren. Für eine $m \times n$ Matrix A, $c \in \mathbb{R}^n$ und $b \in \mathbb{R}^m$ lautet die Aufgabe:

$$\begin{array}{rl} \text{Maximiere (minimiere) die Zielfunktion} & < c, x > \\ \text{unter den Nebenbedingungen} & Ax \leq b \text{ und } x \geq 0. \end{array}$$

Dabei gilt für $x, y \in \mathbb{R}^n$: $x \geq y$, wenn für alle $1 \leq i \leq n$ $x_i \geq y_i$ gilt.

Durch die Einführung zusätzlicher Variablen $x_{n+1}, \ldots, x_{n+m}$ läßt sich diese Aufgabe umschreiben zu: gesucht ist ein $z \in \mathbb{R}$, so daß mit Hilfe der Zielfunktion das erweiterte

```
procedure FVS(G = (V, E, w));
    i := 0;
    FVS := Feedback(w) ∪ (V − V(w));
end;

procedure Feedback(w);
    i := i + 1
    if V(w) = ∅ then return ∅;
    else
        w'(u) := w(u) für alle u;
        while es ex. ein u ∈ V(w') mit d(u, V(w')) ≤ 1 do w'(u) := 0;
        Reduce(w', w'');
        F := (V(w') − V(w'')) ∪ Feedback(w'');
        for all u ∈ (V(w') − V(w'')) do
            if V(w') − (F − {u}) induziert keinen Zyklus in G(V(w')) then
                F := F − {u};
        end if; end for;
        return F;
end if; end;

procedure Reduce(w', w'');
    if es ex. ein semi-disjunkter Zyklus ζ in G(w') then V_i := ζ;
    else V_i := V(w');
    δ := min{ w'(v) / (d(v, V_i) − 1) : v ∈ V_i };
    if v ∈ V_i then w_i := δ(d(v, V_i) − 1);
    else w_i(v) := 0;
    w'' := w' − w_i;
end;
```

Algorithmus 4.1: 1-approx. FVS-Algorithmus für ungerichtete Graphen

Gleichungssystem

$$\sum_{i=1}^{n} a_{ji}x_i + x_{n+j} = b_j, \ j \in \{1, \ldots, m\} \quad \text{und} \quad -\sum_{i=1}^{n} c_i x_i + z = 0 = z_0$$

eine nichtnegative Lösung besitzt. Diese Form wird auch als *kanonische Form* der LP bezeichnet.

Die LP-Aufgabe selbst kann z.B. mit dem Simplex-Algorithmus gelöst werden. Mehr zu LP und dem Simplex-Algorithmus ist in [Kos91] und [Sch93] zu finden.

Sei $G = (V, E, w)$ ein gewichteter Graph und $t : V \to \{0, 1\}$ eine Indikatorfunktion für die Zugehörigkeit im FVS von G. Die LP-Aufgabe für das FVS-Problem kann dann z.B. wie folgt formuliert werden:

$$\begin{array}{rl} \text{minimiere} & \sum_{v \in V} w(v) \cdot t(v) \\ \text{so daß} & \sum_{v \in \zeta} t(v) \geq 1 \text{ für alle } \zeta \in C. \end{array}$$

Dabei ist C die Menge der Zykel in G. Für das SUBSET-FVS-Problem bleibt die Formulierung die Gleiche, nur wird die Menge der interessierenden Zykel betrachtet.

Berechnet wird eine Lösung über $\mathbb{R}$. Gesucht war aber eine ganzzahlige Lösung. Durch nachträgliches Runden erhält man somit einen Approximationsalgorithmus. Je nach Formulierung ergibt sich ein relativer oder konstanter Faktor (siehe z.B. [BYGNR94], [BG94]). In [Hoc96] ist eine Formulierung mit einem Approximationsverhältnis von 2 angegeben. Für das SUBSET-FVS-Problem (siehe Abschnitt 2.2) kann das Verhältnis bis zu $O(\log k)$ betragen, wobei k die Anzahl der Sonderknoten ist (siehe [ENSZ96]).

4.1.3. Permutationsgraphen

In diesem Abschnitt wird ein Algorithmus zur Bestimmung eines minimalen FVS für eine Klasse von Graphen, den Permutationsgraphen, vorgestellt, der in Polynomzeit arbeitet. Ein erster Algorithmus für den ungewichteten Fall stammt von Brandstädt und Kratsch (siehe [BK87]). Dieser arbeitet in $O(|V|^6)$. Der hier vorgestellte Algorithmus geht zurück auf Y.D. Liang (siehe [Lia94]). Er arbeitet in $O(|V| \cdot |E|)$.

Sei S_n die Menge der Permutationen über $\{1, 2, \ldots, n\}$. Jede Permutation $\pi \in S_n$ definiert einen Graphen $G_\pi = (V, E)$ mit $V = V_n = \{1, 2, \ldots, n\}$ und $E = \{(i, j) \in V \times V : i < j \text{ und } \pi^{-1}(i) > \pi^{-1}(j)\}$ (π^{-1} ist dabei die inverse Permutation).

Def 4.1 (Permutationsgraph) Ein Graph G heißt *Permutationsgraph*, falls es eine Permutation π gibt, so daß G, abgesehen von einer möglichen Vertauschung der Knotenbezeichnungen oder einer anderen Darstellung, identisch mit G_π ist.

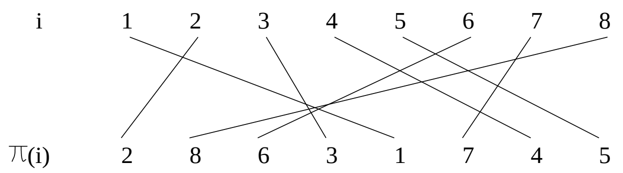

Abbildung 4.2: Das Permutationsdiagramm von G_π

Zur Veranschaulichung der nachfolgenden Definitionen wird eine Darstellung eines Permutationsgraphen G benutzt, die man *Permutationsdiagramm* von G nennt. Dabei werden die Zahlen $1, \ldots, n$ in der ersten Zeile und die Zahlen $\pi(1), \ldots, \pi(n)$ in der zweiten Zeile dargestellt. Jeweils gleiche i werden durch eine Linie verbunden. Diese Linie heißt dann ebenfalls i. In Abb. 4.2 ist das Permutationsdiagramm zur Permutation $\pi = (2, 8, 6, 3, 1, 7, 4, 5)$ dargestellt.

Jedes Paar von Linien i und j in V_n heißt *geordnetes Cross-Paar*, bezeichnet mit X_{ij}, falls $i \leq j$ und $\pi^{-1}(i) \geq \pi^{-1}(j)$. In Abb. 4.2 sind z.B. die Paare $(1, 2), (1, 8)$ und $(3, 3)$ geordnete Cross-Paare, $(2, 3)$ und $(2, 1)$ aber nicht.

Für alle $j \in V_n$ bezeichne L_j die geordnete Liste $(k_0, k_1, \ldots, k_r)$, so daß alle k_t mit $0 < t \leq r$ die Zeile j kreuzen und $\pi^{-1}(j) = \pi^{-1}(k_0) < \pi^{-1}(k_1) < \ldots < \pi^{-1}(k_r)$.

Für alle X_{ij} sei $limit(X_{ij})$ das geordnete Cross-Paar $X_{i'j'}$ mit **(a)** $j' = \max\{j'' : j'' < i \text{ und } \pi^{-1}(j'') < \pi^{-1}(j)\}$ und **(b)** $i' = \max\{i'' : i'' < \pi^{-1}(j) \text{ und } i'' < i\}$. Falls kein solches $X_{i'j'}$ existiert, sei $limit(X_{ij}) = X_{00}$.

Für ein geordnetes Cross-Paar X_{ij} sei $V_{ij} = \{k : k \leq j \text{ und } \pi^{-1}(k) \leq \pi^{-1}(i)\}$. Falls $i \neq j$ bezeichne $pred(X_{ij}) = \max\{i' : i' \in V_{ij} \text{ und } i' < j\}$ und $pred^{\pi}(X_{ij}) = \max\{i' : i' \in V_{ij} \text{ und } i' < \pi^{-1}(i)\}$.

Um ein FVS mit minimalem Gewicht für einen Graphen G zu erhalten, ist es ausreichend ein zykelfreie Menge mit maximalem Gewicht (MWCFS) zu finden, da $F_{opt}(G) = V$ - MWCFS. Es ist außerdem bekannt, daß die einzigen Zykel in Permutationsgraphen die Zykel C_3 und C_4 sind[3] (siehe [Lia94]). Somit ist ein Untergraph G' eines Permutationsgraphen G zykelfrei, falls G' keine Zykel C_3 oder C_4 enthält.

Für $S \subseteq V_n$ sei $W(S) = \sum_{i \in S} w(i)$ für $S \neq \emptyset$ und $W(\emptyset) = 0$. Die folgenden Mengen werden für die Berechnung eines MWCFS benötigt.

- A_{ij} bezeichne ein MWCFS in V_{ij}. $A_{\pi(n)n}$ ist somit ein MWCFS in G, da $V_n = V_{\pi(n)n}$.
- B_{ij} bezeichne ein MWCFS in V_{ij} plus $\{i, j\}$.
- $C_{ij}(k)$ bezeichne ein MWCFS in $V_{ij} \cup \{k\}$ plus $\{k\}$, wobei $\pi^{-1}(k) > \pi^{-1}(i)$.
- $D_{ij}(k)$ bezeichne ein MWCFS in $V_{ij} \cup \{k\}$ plus $\{k\}$, wobei $k > j$.
- $E_{ij}(k)$ bezeichne ein MWCFS in $V_{ij} \cup \{k\}$ plus $\{k, i, j\}$, wobei $\pi^{-1}(k) > \pi^{-1}(i)$ und $k < j$. Falls $k < i < j$ (i, j, k bilden einen C_3) sei $E_{ij}(k) = \emptyset$.
- $F_{ij}(k)$ bezeichne ein MWCFS in $V_{ij} \cup \{k\}$ plus $\{k, i, j\}$, wobei $\pi^{-1}(k) < \pi^{-1}(i)$ und $k < j$. Falls $\pi^{-1}(k) < \pi^{-1}(i) < \pi^{-1}(j)$ (i, j, k bilden einen C_3) sei $F_{ij}(k) = \emptyset$.

Zusätzlich sei $A_{00} = C_{00}(k) = D_{00}(k) = \emptyset$, für $k = 1, \ldots, n$.

Ein einfacher Ansatz für die Berechnung von A_{ij} wäre der Vergleich aller $B_{i'j'}$ für $X_{i'j'} \leq X_{ij}$ und $i', j' \in V_{ij}$ und die Auswahl des MWCFS mit dem größten Gewicht. Der Aufwand dafür ist aber zu groß. Die Lemmata 3.2 - 3.8 in [Lia94] bilden die Grundlage des dort beschriebenen Algorithmus und zeigen einen effizienteren Weg. Sie sind eher technischer Natur und werden, genauso wie der Algorithmus, hier nicht angegeben.

4.2. SUBSET-FVS

4.2.1. Knotengrad abhängige Approximation

In diesem Abschnitt wird ein Algorithmus beschrieben, der eine SUBSET-FVS mit einem Approximationsverhältnis von $2 \cdot \Delta$ zu einem minimalen SUBSET-FVS berechnet. Δ ist dabei der maximale Grad eines Knoten im Graphen. Hierzu wird das gegebene SUBSET-FVS-Problem auf ein SUBSET-FES-Problem (siehe Abschnitt 2.3.1) reduziert, ein SUBSET-FES berechnet und daraus ein SUBSET-FVS bestimmt.

[3] C_n bezeichnet einen Zyklus der Länge n.

Zunächst zum SUBSET-FES-Problem. Sei $G = (V, E, w)$ ein gewichteter Graph. Sei $S = \{s_1, \ldots, s_k\}$ mit $s_i \in V$ die Menge der Sonderknoten. Sei $V_i = V - \{s_i, \ldots, s_k\}$. O.B.d.A. gelten für G die folgenden Bedingungen.

1. Für alle Kanten e die inzident zu Knoten aus S sind, gilt: $w(e) = \infty$.

2. Für $1 \leq i \leq k$ gilt: $d(s_i) = 2$ in $G(V_{i+1})$.

Bedingung (1) wird erfüllt, indem man jede Kante, die inzident zu einem Knoten $s_i \in S$ ist, in zwei neue Kanten teilt. Die neue Kante, die inzident zu s_i ist, erhält unendliches Gewicht, während die andere Kante das Gewicht der ursprünglichen Kante bekommt. Ist eine Kante zu zwei Knoten aus S inzident, wird sie in 3 Kanten unterteilt, wobei die neuen Kanten, die inzident zu den Knoten aus S sind, ein unendliches Gewicht erhalten. Um Bedingung (2) zu erfüllen, wird jeder Knoten $s_i \in S$, dessen Grad größer als zwei ist, zerlegt in eine Kette von $d(s_i) - 1$ neuen Knoten. Angenommen $v_1, \ldots, v_l$ seien die Nachbarn von s_i. s_i wird ersetzt durch die neuen Knoten $s_{i,1}, s_{i,2}, \ldots, s_{i,l-1}$. Die Nachbarn von $s_{i,1}$ sind v_1 und v_2, von $s_{i,2}$ v_3, $s_{i,1}$ und $s_{i,3}$. Bedingung (2) gilt unter Beachtung der Ordnung der neuen Knoten.

Sei G' der Graph, den man mit diesen Modifikationen erhält. Das folgende Theorem zeigt, daß diese Veränderungen nicht das SUBSET-FES-Problem beeinflußen.

Theorem 4.3 Für jedes FES $M \subseteq E$ in G mit endlichem Gewicht exitiert ein korrespondierendes FES $M' \subseteq E'$ in G' (und umgekehrt) mit $w(M) = w(M')$.

Zum Beweis: siehe Theorem 1 in [ENSZ96].

Der SUBSET-FES-Algorithmus 4.3 besteht aus k Iterationen. Auf der i-ten Iterationsstufe wird ein SUBSET-FES ermittelt, das alle interessierenden Zykel überdeckt, die noch nicht in einer früheren Iterationstufe überdeckt worden sind. Der höchste Index eines Knotens aus S ist dabei i. Nach Eigenschaft (2) hat jeder Knoten $s_i \in S$ genau zwei Nachbarn. Seien diese mit x_i und y_i benannt. Das SUBSET-FES in jedem Iterationsschritt wird bestimmt, indem man den minimalen Schnitt (siehe Abschnitt 2.1) zwischen x_i und y_i berechnet.

Eine formelle Beschreibung des Algorithmus kann in Abschnitt 3.2 in [ENSZ96] nachgelesen werden. Dort findet man auch den Beweis für das folgende Theorem.

Theorem 4.4 Algorithmus 4.3 liefert ein SUBSET-FES M von G mit: $w(M) \leq 2 \cdot w(M_{opt})$. Dabei ist M_{opt} ein minimales SUBSET-FES.

```
for i = 1 to k do
    M_i := minimaler Schnitt zwischen x_i und y_i in G_i = (V_i, E − ∪_{j=1}^{i−1} M_j);
end for;
SUBSET-FES = M_1 ∪ ··· ∪ M_k;
```

Algorithmus 4.3: SUBSET-FES Algorithmus

Zurück zu dem SUBSET-FVS-Problem. Für einen Graphen $G = (V, E, w)$ besteht das korrespondierende SUBSET-FES-Problem aus dem Graphen $G' = (V, E, w')$. Dabei ist $w' : E \to \mathbb{R}_+$ eine Gewichtsfunktion über den Kanten. Es gelte: $w(u, v) = \min\{w(u), w(v)\}$ für jede Kante $(u, v) \in E$.

Durch diese Reduktion gelangt man zu dem Faktor Δ im Approximationsverhältnis, denn es gilt:

Bemerkung 4.1 Die Kosten einer optimalen Lösung des SUBSET-FES-Problems betragen höchstens Δ-mal soviel, wie die Kosten einer optimalen Lösung des korrespondierenden SUBSET-FVS-Problems.

Sei M ein durch Algorithmus 4.3 berechnetes SUBSET-FES. Ein SUBSET-FVS F kann man aus M wie folgt gewinnen. Für jede Kante $(u, v) \in M$ füge u zu F, falls $w(u) \leq w(v)$. Ansonsten füge v zu F. Mit Bemerkung 4.1 und Theorem 4.4 erhält man somit:

Theorem 4.5 Das SUBSET-FVS-Problem kann in polynomieller Zeit auf einen Faktor $2 \cdot \Delta$ approximiert werden.

4.2.2. Ein 7-approximativer Algorithmus

Der in diesem Abschnitt vorgestellte Algorithmus stammt aus [ENZ96] und ist der erste Algorithmus zur Berechnung des SUBSET-FVS-Problems, der ein konstantes Approximationsverhältnis hat. Zur Berechnung eines SUBSET-FVS wird hierzu das Problem auf ein spezielles Flußproblem in Netzwerken (siehe Abschnitt 2.1) reduziert.

Zunächst wird eine Transformation, die Gomory-Hu-Transformation, in Netzwerken beschrieben. Sei $N = (V, E, c)$ ein Netzwerk und sei C ein minimaler Schnitt (siehe Abschnitt 2.1) zwischen Quelle x und Senke y in G. Der Schnitt darf dabei sowohl Kanten, als auch Knoten beinhalten. Sei C^e die Menge der Kanten von C und C^v die Menge der Knoten von C. $G - C$ besteht aus zwei disjunkten Untergraphen, X und Y mit $x \in X$ und $y \in Y$.

Sei R_x der Graph, den man aus G erhält, wenn man alle Knoten in Y zu einem einzelnen Knoten zusammenfaßt. Dieser Knoten sei y' und habe ein unendliches Gewicht. Analog bezeichne R_y den Graphen, den man aus G erhält, wenn man alle Knoten in X zu einem Knoten zusammenfaßt. Dieser heiße x' und habe ebenfalls unendliches Gewicht. Jeder Knoten $u \in C^v$ erscheint sowohl in R_x als auch in R_y. Bezeichne u_x die Kopie von u in R_x und u_y die Kopie von u in R_y. Die Knoten u_x und u_y werden im folgenden auch *virtuelle Kopien* genannt. Die Kanten, die mit Kanten aus C^e korrespondieren oder inzident zu virtuellen Kopien von Knoten aus C^v sind, heißen *virtuelle Kanten*.

Im folgenden wird stets von der Annahme ausgegangen, daß die für alle Knoten $u, v \in V$ mit endlichem Gewicht gilt: $(u, v) \notin E$. Dies kann erreicht werden, indem auf jeder Kante ein Knoten mit unendlichem Gewicht eingefügt wird.

Desweiteren können wir o.B.d.A. annehmen, daß für alle $s_i \in S$ die folgenden Bedingungen gelten.

1. s_i hat unendliches Gewicht

2. $d(s_i) = 2$

3. Beide Nachbarn von s_i sind keine Knoten aus S und haben unendliches Gewicht.

Bedingung (3) ist erfüllt, da auf jeder Kante ein Knoten mit unendlichem Gewicht eingefügt wurde. Sei $s \in S$ ein Knoten der die Bedingungen (1) und (2) nicht erfüllt. Auf jeder Kante, die inzident zu s ist, wird ein Knoten mit unendlichem Gewicht eingefügt. Alle neu eingefügten Knoten werden zu S hinzuaddiert und s aus S entfernt.

Ähnlich zu den Bedingungen in Abschnitt 4.2.1 verändern auch diese Forderungen weder die Menge der interessierenden Zykel noch ein optimales SUBSET-FVS.

Sei $N = (V, E, c)$ ein Netzwerk mit zusätzlicher Gewichtsfunktion w über den Knoten. Sei $S = \{s_1, \ldots, s_k\}$ die Menge der Sonderknoten. Die beiden Nachbarn von Knoten $s_i \in S$ seien x_i und y_i für $1 \leq i \leq k$. Bezeichne (a_i, b_i) das i-te Quelle-Senke-Paar. Bezeichne $f_i(e)$ den Wert des Flusses von a_i nach b_i auf Kante e. Sei $f(e) = \max_{1 \leq i \leq k} f_i(e)$. Im folgenden werden die Flüsse $(f_1, \ldots, f_k)$ auch einfach nur mit f bezeichnet.

Def 4.2 Ein Fluß $f = (f_1, \ldots, f_k)$ zwischen den Quelle-Senke-Paaren $\{(a_i, b_i)\}_{i=1}^k$ ist ein *relaxierter Multi-Commodity-Fluß*, welcher den Anforderungsvektor $d = (d_1, \ldots, d_k)$ erfüllt, wenn für alle $1 \leq i \leq k$ der Fluß f_i von a_i nach b_i gleich d_i ist und die folgenden Bedingungen erfüllt sind:

- *Intra-Commodity-Bedingungen*:
  1. für alle $e \in E : f_i(e) \leq c(e)$ und
  2. für alle $v \in V : \sum_{e \in E(v)} f_i(e) \leq 2 \cdot w(v)$.
- *Inter-Commodity-Bedingungen*: für alle $v \in V : \sum_{e \in E(v)} f(e) \leq 4 \cdot w(v)$.

Zusätzlich seien die folgenden Klassen von *gesättigten* Knoten bezüglich f definiert:

- Knoten $u \in V$ heißt *intra-gesättigt* bezüglich Fluß i, wenn u die Intra-Commodity-Bedingung von Fluß i erfüllt.
- Knoten $u \in V$ heißt *inter-gesättigt*, wenn u die Inter-Commodity-Bedingungen erfüllt.

Sei I die Menge der inter-gesättigten Knoten, die in jeder Realisation eines Anforderungsvektors d gesättigt sind. Das folgende Theorem begrenzt das Gewicht dieser Knoten.

Theorem 4.6 Sei $d = (d_1, \ldots, d_k)$ ein Anforderungsvektor der von jedem relaxierten Multi-Commodity-Fluß erfüllt werden kann. Dann gilt:

$$\sum_{v \in I} w(v) \leq \sum_{i=1}^{k-1} d_i.$$

Der Beweis zu diesem Theorem ist in Abschnitt 4.1 in [ENZ96] zu finden.

In Abschnitt 4.2.1 wurde ein SUBSET-FES berechnet, indem die minimalen Schnitte zwischen den Knoten x_i und y_i für alle $1 \leq i \leq k$ bestimmt wurden. Dieser Ansatz läßt sich auch für die Berechnung eines SUBSET-FVS benutzen, liefert aber eine Lösung, die das k-fache Gewicht einer optimalen Lösung haben kann. Der folgende Algorithmus verfeinert diesen Ansatz und versucht, "wichtige" Knoten zu finden, z.B. Knoten, die zu vielen interessierenden Zykel gehören. Grob gesagt wird ein maximaler, erfüllbarer Anforderungsvektor d bezüglich der lexikographischen Ordnung der Sonderknoten berechnet. Es werden dann die Knoten zur Lösung hinzuaddiert, die in jedem d realisierendem relaxierten Multi-Commodity-Fluß inter-gesättigt sind.

Der Algorithmus selbst berechnet einen relaxierten Multi-Commodity-Fluß. Begonnen wird mit einem Graph $H_0 = N - S$. Alle Kanten von H_0 haben eine unendliche Kapazität. Ein Iterationsschritt des Algorithmus wird im folgenden beschrieben. Die Eingabe besteht aus dem Graphen H_i und einem Anforderungsvektor $d = (d_1, \ldots, d_i)$, welcher in H_{i-1} realisiert wurde.

1. Berechne einen relaxierten Multi-Commodity-Fluß $f = (f_1, \ldots, f_{i+1})$ in H_i, die (i) den Anforderungsvektor d für die Flüsse $1, \ldots, i$ für die Quelle-Senke-Paare $x'_j, y'_j, 1 \leq j \leq i$ erfüllt und (ii) d_{i+1} maximiert.

2. Berechne einen minimalen Schnitt C und eine Menge von Knoten $F_{i+1} = A_{i+1} \cup B_{i+1}$ mit: A_{i+1} ist eine Teilmenge der Knoten, die von f_{i+1} intra-gesättigt sind und B_{i+1} ist eine Teilmenge der inter-gesättigten Knoten von f.

3. Konstruiere H_{i+1} aus H_i durch Anwendung der Gomory-Hu-Transformation auf den minimalen Schnitt C zwischen x_{i+1} und y_{i+1} und durch Hinzufügen des Knoten s_{i+1}.

Die Ausgabe des Algorithmus ist ein SUBSET-FVS $F = F_1 \cup \cdots \cup F_k$.

Die Berechnung des relaxierten Multi-Commodity-Flusses in Schritt 1 des Algorithmus kann durch ein Lineares Programm erfolgen (siehe Abschnitt 4.1.2). Die gesuchte Lösung soll d_{i+1} maximieren. Die Nebenbedingungen des linearen Programms entsprechen dem Standard, außer den Inter-Commodity-Bedingungen: für alle Knoten $v \in V$ mit $w(v) < \infty$ gelte:

$$\begin{array}{ll} \forall e \in E(v), & \forall j \in \{1, \ldots, k\} : f_j(e) \leq f(e) \\ \forall v : & \sum_{e \in E(v)} f(e) \leq 4 \cdot w(v) \end{array}$$

Gesucht sind die Knoten in H_i, die von jedem relaxierten Multi-Commodity-Fluß der den Anforderungsvektor $(d_1, \ldots, d_{i+1})$ erfüllt, inter-gesättigt sind. Diese Menge heiße I_{i+1}. Die Knoten in I_{i+1} können folgendermaßen identifiziert werden: für alle $v \in V$ mit $w(v) < \infty$ löse das lineare Programm, welches $\sum_{e \in E(v)} f^v(e)$ minimiert. Die Nebenbedingung für das lineare Programm ist: f^v ist ein relaxierter Multi-Commodity-Fluß, der d erfüllt. Von dem Mittel über alle Flüsse $\{f^v\}_{v \in V}$ werden dann nur Knoten aus I_{i+1} inter-gesättigt.

Zu Schritt 2. Kapazitäten der Kanten wurden bei der Konstruktion der Graphen $H_0, \ldots, H_{i-1}$ zugewiesen und endliche Kapazität haben nur die virtuellen Kanten. Zusätzliche gelte: für alle $v \in I_{i+1}$ und $e \in E(v)$ sei $c(e) = \min\{c(e), f(e)\}$. Da d_{i+1} maximal ist, folgt: f_{i+1} ist ein maximaler Fluß zwischen x_{i+1} und y_{i+1}. Sei C ein minimaler Schnitt zwischen x_{i+1} und y_{i+1}. Knoten in C^v sind intra-gesättigt durch f_{i+1}. Setze $A_{i+1} = C^v$. Kanten in C^e sind entweder virtuell oder inzident zu Knoten in I_{i+1}. Von jeder nicht-virtuellen Kante in C^e fügen wir den Endknoten v aus I_{i+1} zu B_{i+1}, falls $v \notin A_{i+1}$. Damit besteht A_{i+1} aus intra-gesättigten Knoten und B_{i+1} aus inter-gesättigten Knoten bezüglich f_{i+1} bzw. $f = (f_1, \ldots, f_{i+1})$.

In Schritt 3 wird die Gomory-Hu-Transformation auf H_i, Schnitt C und das Quelle-Senke-Paar (x_{i+1}, y_{i+1}) angewandt. Man erhält die Graphen $R_{x_{i+1}}$ mit Knoten y'_{i+1} und $R_{y_{i+1}}$ mit Knoten x'_{i+1}. Verbinde $R_{x_{i+1}}$ und $R_{y_{i+1}}$ durch Einfügen des Knotens s_{i+1} und Verbinden mit seinen beiden Nachbarn x'_{i+1} und y'_{i+1}. Der so erhaltene Graph ist H_{i+1}. Die Kapazitäten der Kanten werden folgendermaßen zugewiesen: falls eine Kante in einem der Graphen $H_1, \ldots, H_i$ eine virtuelle Kante ist, dann behält sie ihre Kapazität. Falls e eine virtuelle Kante in H_{i+1} ist, dann sei $c(e) = f(e)$. Alle anderen Kanten haben unendliche Kapazität.

In Abschnitt 5 in [ENZ96] wird die Korrektheit des Algorithmus gezeigt. In Abschnitt 6 in [ENZ96] wird außerdem das folgende Theorem bewiesen, welches in Zusammenhang mit Theorem 4.6 den Approximationsfaktor des Algorithmus bestimmt.

Theorem 4.7 Sei $F_{opt} \subseteq V$ ein optimales SUBSET-FVS mit: F_{opt} trennt alle Quelle-Senke-Paare $\{(a_i, b_i)\}_{i=1}^k$ in $G-S$. Für alle Anforderungsvektoren $d = (d_1, \ldots, d_k)$, welche in $G-S$ bezüglich der Paare (a_i, b_i) realisierbar sind, gilt:

$$\sum_{i=1}^{k} d_i \leq 4 \cdot w(F_{opt}).$$

Damit folgt: $w(F) \leq 2 \cdot \sum_{i=1}^k d_i$ und $\sum_{i=1}^k d_i \leq 4 \cdot w(F_{opt})$ und somit $w(F) \leq 8 \cdot w(F_{opt})$.

5. Ein Fazit

Das FVS-Problem gehört zu den klassischen und lange Jahre zu den am wenigsten verstandenen **NP**-vollständigen Problemen. Dies scheint sich geändert zu haben. Die Arbeiten auf dem Gebiet der FVS-Probleme haben in den letzten Jahren viel zu deren Verständnis beigetragen. So konnten eine Vielzahl von Algorithmen, die man auf anderen Gebieten entwickelte, zur Berechnung von FVSs genutzt werden, z.B. aus dem Bereich der Netzwerke. Dies zeigt auch, wie eng die einzelnen Problemstellungen miteinander verbunden sind.

Eine Reihe von verschiedenen Spezialfälle wurde gefunden, deren Lösung in polynomieller Zeit erfolgen kann (siehe Abschnitt 3.1). Große Fortschritte sind allerdings auch auf dem Gebiet der Approximationsalgorithmen erzielt worden. So beträgt das Approximationsverhältnis beim FVS-Problem in gerichteten (siehe Abschnitt 3.2 und 3.3), wie in ungerichteten Graphen (4.1.1) mittlerweile 2. Beim SUBSET-FVS-Problem konnte erstmalig ein

Algorithmus mit konstantem Approximationsverhältnis entwickelt werden (siehe Abschnitt 4.2.2).

Literatur

[BBF96] V. Bafna, P. Berman, and T. Fujito. Constant ratio approximation of feedback vertex sets in weighted undirected graphs. *DIMACS Technical Report 96*, 29, 1996.

[BG94] A. Becker and D. Geiger. Approximation algorithms for the loop cutset problem. *13th Conf, on Uncertainty in AI*, pages 60–68, 1994.

[BK87] A. Brandstädt and D. Kratsch. On domination problems for permutation and other graphs. *Theoretical Computer Science*, 54:181–198, 1987.

[BK96] A. Benczur and D.R. Karger. Approximating s-t minimum cuts in $o(n^2)$ time. *28th STOC*, pages 47–55, 1996.

[Bra87] A. Brandstädt. The computational complexity of feedback vertex set, hamiltonian circuits, domination set, steiner tree and bandwidth on special perfect graphs. *J. Inf. Process Cybern.*, EIK 23:471–477, 1987.

[BYGNR94] R. Bar-Yehuda, D. Geiger, J. Naor, and R. Roth. Approximation algorithms for the vertex feedback set problem with applications to constraint satisfaction and bayesian inference. *5th SODA*, pages 344–353, 1994.

[CH94] John Clark and Derek Allan Holton. *Graphentheorie*. Spektrum Akademischer Verlag, Heidelberg, 1994.

[ENSS95] Guy Even, Joseph (Seffi) Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. In *Proceedings of the 4th MPS Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 14–28. University of Copenhagen, 1995.

[ENSZ96] G. Even, J. Naor, B. Schieber, and L. Zosin. Approximating minimum subset feedback sets in undirected graphs with applications. *4th ISTCS*, pages 78–88, 1996.

[ENZ96] Guy Even, Joseph Naor, and Leonid Zosin. An 8-approximation algorithm for the subset feedback vertex set problem. In *37th annual symposium on Foundation of Computer Science (FOCS)*, pages 310–319. Burlington, Vermont, 1996.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[Hac97] Wolfgang Hackbusch. On the feedback vertex set problem for a planar graph. *Computing*, 58(2):129–155, 1997.

[HO92] J. Hao and J.B. Orlin. A faster algorithm for finding the minimum cut in a graph. *Proceedings of the third annual ACM-SIAM symposium on discrete algorithms*, pages 165–174, 1992.

[Hoc96] D.S. Hochbaum. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, MA, 1996.

[HU88] J.E. Hopcroft and J.D. Ullman. *Einführung in Automatentheorie, Formale Sprachen und Komplexitätstheorie.* Adison-Wesley, 1988.

[Kar72] R.M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[Kos91] Peter Kosmol. *Optimierung und Approximation.* de Gryter, Berlin, 1991.

[Lia94] Y.D. Liang. On the feedback vertex set problem in permutation graphs. *Information Processing Letters*, 52:123–129, 1994.

[LSW88] Errol L. Lloyd, Mary Lou Soffa, and Ching-Chy Wang. On locating minimum feedback vertex sets. *Journal of Computer and System Sciences*, 37:292–311, 1988.

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

[Sch93] Hans Rudolf Schwarz. *Numerische Mathematik.* B.G. Teubner, Stuttgart, 1993.

[Sha79] Adi Shamir. A linear time algorithm for finding minimum cutsets in reducible graph. *SIAM J. Comput.*, 8(4):645–655, 1979.

[SW75] G.W. Smith and R.B. Walford. The identification of a minimal feedback vertex set of a directed graph. *IEEE Trans. Circuits and Systems*, CAS-22(1):9–14, 1975.

[Tar72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2), 1972. 146-160.

[WLS85] Ching-Chy Wang, Errol L. Lloyd, and Mary Lou Soffa. Feedback vertex sets and cyclically reducible graphs. *Journal of the Association for Computing Machinery*, 32(2):296–313, 1985.